Texas Instruments

TI-89 Titanium

Graphing Calculator

Important Information

Texas Instruments makes no warranty, either express or implied, including but not limited to any implied warranties of merchantability and fitness for a particular purpose, regarding any programs or book materials and makes such materials available solely on an "as-is" basis. In no event shall Texas Instruments be liable to anyone for special, collateral, incidental, or consequential damages in connection with or arising out of the purchase or use of these materials, and the sole and exclusive liability of Texas Instruments, regardless of the form of action, shall not exceed the purchase price of this product. Moreover, Texas Instruments shall not be liable for any claim of any kind whatsoever against the use of these materials by any other party.

USA FCC Information Concerning Radio Frequency Interference

This equipment has been tested and found to comply with the limits for a Class B digital device, pursuant to Part 15 of the FCC rules. These limits are designed to provide reasonable protection against harmful interference in a residential installation. This equipment generates, uses, and can radiate radio frequency energy and, if not installed and used in accordance with the instructions, may cause harmful interference to radio communications. However, there is no guarantee that interference will not occur in a particular installation.

If this equipment does cause harmful interference to radio or television reception, which can be determined by turning the equipment off and on, you can try to correct the interference by one or more of the following measures:

- Reorient or relocate the receiving antenna.
- Increase the separation between the equipment and receiver.
- Connect the equipment into an outlet on a circuit different from that to which the receiver is connected.
- Consult the dealer or an experienced radio/television technician for help.

Caution: Any changes or modifications to this equipment not expressly approved by Texas Instruments may void your authority to operate the equipment.

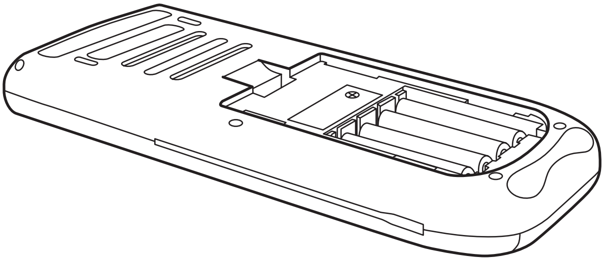
Getting Started

Initial start-up

Installing the AAA Batteries

The TI-89 Titanium uses four AAA alkaline batteries and one button cell backup battery. The backup battery is already installed, and the AAA batteries are provided with the product.

1. Remove the battery cover from the back of the calculator.
2. Unwrap the four AAA batteries provided with your product and insert them in the battery compartment. Arrange the batteries according to the polarity (+ and -) diagram in the battery compartment.



3. Replace the battery cover on the calculator. The cover should snap into place.

Turning on your TI-89 Titanium for the first time

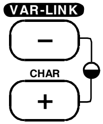
After installing the batteries included with the calculator, press `ON`. The Apps desktop appears.

Note: If your calculator initializes the preinstalled Apps, a progress bar will appear with the message “`Installation in progress . . . Do not interrupt!`” instead of the Apps desktop. To avoid losing Apps, do not remove the batteries during initialization. (You can re-install Apps from either the Product CD-ROM or education.ti.com.)

Progress bar

Adjusting the contrast

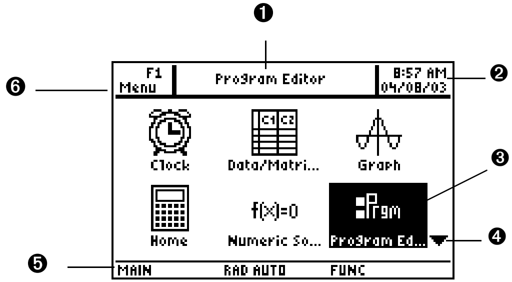
- To lighten the display, press and hold [♦] and tap [−].
- To darken the display, press and hold [♦] and tap [+].



The Apps desktop

The Apps desktop is the starting point for operating your TI-89 Titanium. Your installed Apps appear on the Apps desktop as icons organized in categories for easy access. From the Apps desktop, you can:

- Open Apps.
- Select and edit categories of Apps.
- View all of the Apps installed on your calculator.
- View the full name of the highlighted App.
- View and edit the time and date.
- Check status line information.
- View split-screen mode information.



TI-89 Titanium Apps desktop

❶ View full name of highlighted App.

❷ View time and date.

❸ Press [ENTER] to open highlighted App.

❹ Scroll down to view additional Apps.

❺ Check status line information.

❻ Edit categories.

To return to the Apps desktop at any time, press [APPS]. The last category selected appears with the last open App highlighted.

Turning off the calculator

Press [2nd] [OFF]. The next time you turn on the calculator, the Apps desktop appears with the same settings and memory contents retained. (If you turned off the Apps desktop, the calculator Home screen appears.)

You can use either of the following keys to turn off the TI-89 Titanium.

Press:	Description
[2nd] [OFF] (press [2nd] and then press [OFF])	Settings and memory contents are retained by the Constant Memory™ feature. • You cannot, however, use [2nd] [OFF] if an error message is displayed. • When you turn the TI-89 Titanium on again, it displays either the Home screen or the Apps desktop (regardless of the last application you used).
[♦] [OFF] (press [♦] and then press [OFF])	Similar to [2nd] [OFF] except: • You can use [♦] [OFF] if an error message is displayed. • When you turn the TI-89 TitaniumVoyage™ 200 on again, it will be exactly as you left it.

Note: [OFF] is the second function of the [ON] key.

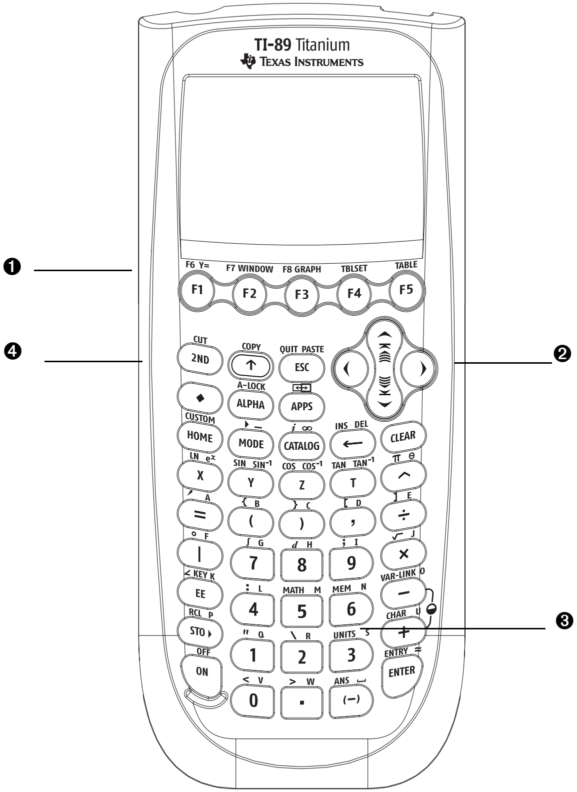
The calculator's Automatic Power Down™ (APD™) feature prolongs battery life by turning the calculator off automatically following several minutes of inactivity. When you turn on the calculator after APD:

- The display, cursor, and any error conditions are exactly the same as before APD.

- All settings and memory contents are retained.

Note: APD does not occur if a calculation or program is in progress, unless the program is paused. If a program is running but waiting for a key press, APD will occur after several minutes of inactivity.

TI-89 Titanium keys



TI-89 Titanium keys

❶ Function keys ([F1]– [F8]) open toolbar menus, access Apps, and edit categories of Apps.

❷ Cursor keys (◀, ▶, ▲, ▼) move the cursor.

❸ Numeric keypad performs math and scientific functions.

❹ Modifier keys ([2nd], [♦], [↑]) add features by increasing the number of key commands.

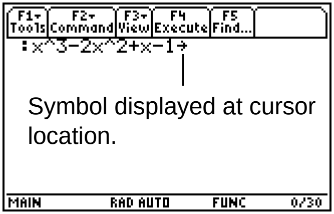
Entering special characters

Use the CHAR (Character) menu and key commands to enter special characters. The CHAR menu lets you access Greek, math, international, and other special characters. An on-screen keyboard map shows the locations of shortcuts used to enter other commonly used characters.

To select characters from the CHAR menu:

1. Press [2nd] [CHAR]. The CHAR menu appears.
2. Use the cursor keys to select a category. A submenu lists the characters in that category.
3. Use the cursor keys to select a character, and press [ENTER].

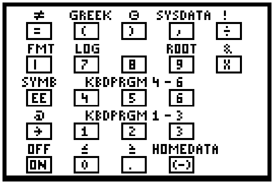
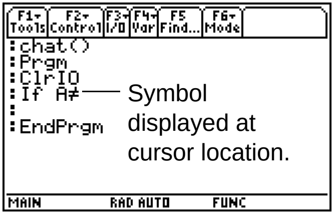
Example: Enter the right arrow symbol (→) in the Text Editor.

Press	Result
[2nd] [CHAR]	
4	 Scroll down for more characters.
9 – or – Press ⊙ repeatedly to select **9:**→ and press [ENTER]	 Symbol displayed at cursor location.

To open the keyboard map, press [♦] [KEY]. The keyboard map appears.

To type most characters, press [◆] and the corresponding key. Press [ESC] to close the map.

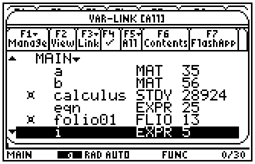
Example: Use the keyboard map to find the “not equal to” symbol (≠) shortcut and enter the symbol in the Program Editor.

Press	Result
[◆] [KEY]	
[◆] [=]	

Modifier keys add features by increasing the number of keyboard operations at your fingertips. To access a modifier function, press a modifier key and then press the key for the corresponding operation.

Keys	Description
[2nd] (Second)	Accesses Apps, menu options, and other operations. Second functions are printed above their corresponding keys in the same color as the [2nd] key.
[◆] (Diamond)	Accesses Apps, menu options, and other operations. Diamond functions are printed above their corresponding keys in the same color as the [◆] key.
[↑] (Shift)	Types an uppercase character for the next letter key you press. Also used with ◀ and ▶ to highlight characters when editing.
[alpha] (Alpha)	Lets you type alphabetic characters without a QWERTY keypad. Alpha characters are printed above their corresponding keys in the same color as the [alpha] key.

Example: Access the VAR-LINK [All] screen, where you can manage variables and Apps.

Press	Result
[2nd] [VAR-LINK]	

Function keys

Use the function keys to perform the following operations:

- On the Apps desktop, open Apps and select or edit Apps categories.
- On the calculator Home screen, open toolbar menus to select math-related operations.
- Within Apps, open toolbar menus to select App options.

Numeric keypad

The numeric keypad lets you enter positive and negative numbers.

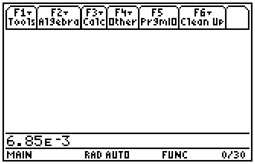
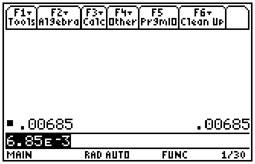
To enter a negative number, press [(-)] before typing the number.

Note: Don’t confuse the negation key ([(-)]) with the subtraction key ([−]).

To enter a number in scientific notation:

1. Type the numbers that precede the exponent. (This value can be an expression.)
2. Press [EE]. The exponent symbol (E) follows the numbers you entered.
3. Type the exponent as an integer with up to three digits. (As the following example shows, you can use a negative exponent.)

Example: On the calculator Home screen, enter 0.00685 using scientific notation.

Press	Result
6 [.] **8 5** [EE] [(-)] **3**	
[ENTER]	

Other important keys

Key Command	Description
[◆] [Y=]	Displays the Y= Editor.
[◆] [WINDOW]	Displays the Window Editor.

Key Command	Description
◆ [GRAPH]	Displays the Graph screen.
◆ [TBLSET]	Sets parameters for the Table screen.
◆ [TABLE]	Displays the Table screen.
◆ [CUT] ◆ [COPY] ◆ [PASTE]	These keys let you edit entered information by performing a cut, copy, or paste operation.
APPS	Displays the Apps desktop.
◆ APPS	With the Apps desktop off, displays the FLASH APPLICATIONS menu.
2nd [◂▸]	Switches between the last two chosen Apps.
2nd [CUSTOM]	Turns the custom menu on and off.
2nd [▶]	Converts measurement units.
◆ [_]	Designates a measurement unit.
←	Deletes the character to the left of the cursor (backspace).
◆ [DEL]	Deletes the character to the right of the cursor.
2nd [INS]	Switches between insert and overwrite modes.
2nd [MEM]	Displays the MEMORY screen.

Key Command	Description
[CATALOG]	Displays a list of commands.
[2nd] [RCL]	Recalls the contents of a variable.
[STO▶]	Stores a value to a variable.
[2nd] [CHAR]	Displays the CHAR menu, which lets you select Greek letters, international accented characters, and other special characters..
[2nd] [QUIT]	• In full-screen mode, displays the Apps desktop. • In split-screen mode, displays the full-screen view of the active App. • With the Apps desktop off, displays the calculator Home screen.

Mode settings

Modes control how the TI-89 Titanium displays and interprets information. All numbers, including elements of matrices and lists, are displayed according to the current mode settings. When the TI-89 Titanium is turned off, the Constant Memory™ feature retains all of the mode settings you have selected.

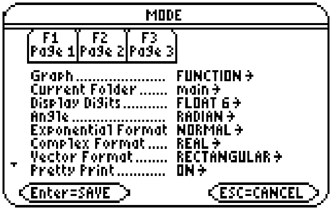
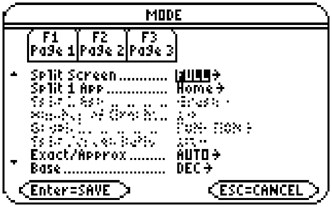
To view the TI-89 Titanium mode settings:

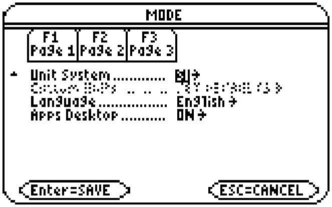
1. Press [MODE]. Page 1 of the MODE dialog box appears.

2. Press [F2] or [F3] to display the modes listed on Page 2 or Page 3.

Note: Modes that are grayed out are available only if other required mode settings are selected. For example, the Custom Units mode listed on Page 3 is available only if the Unit System mode is set to CUSTOM.

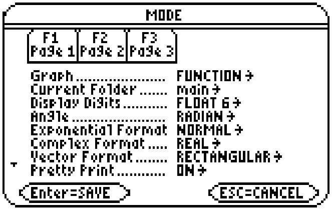
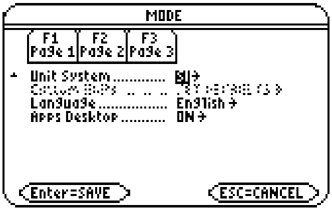
Viewing mode settings

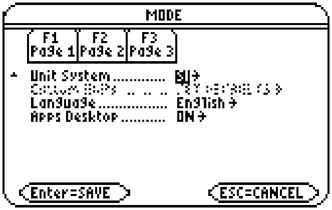
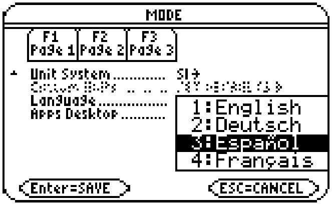
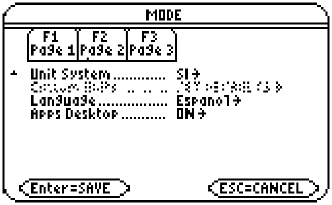
Press	Result
[MODE]	
[F2]	

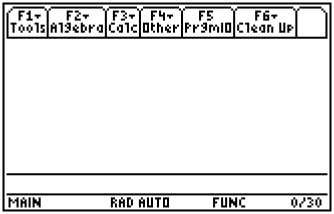
Press	Result
F3	

Changing mode settings

Example: Change the Language mode setting to Spanish (*Español*).

Press	Result
MODE	
F3	

Press	Result
Scroll down to the Language field. ↓	
Press → and then press ↓ until **3:Español** is highlighted. **Note:** Your menu list might vary, depending on the languages installed.	
ENTER	

Press	Result
[ENTER] **Note:** The previous open App appears (in this example, the calculator Home screen).	

To return the Language mode setting to English, repeat the steps, selecting **1:English** in the Language field.

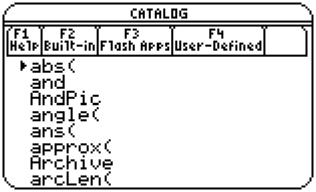
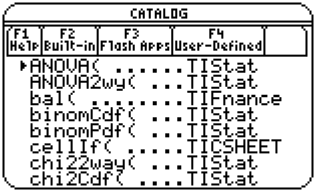
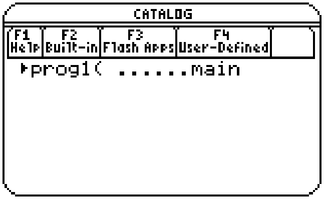
Using the Catalog to access commands

Use the Catalog to access a list of TI-89 Titanium commands, including functions, instructions, and user-defined programs. Commands are listed alphabetically. Commands not beginning with a letter are found at the end of the list (&, /, +, –, etc.).

The Catalog Help App includes details about each command.

Options not currently valid are grayed out. For example, the Flash Apps ([F3]) menu option is grayed out if no Flash applications are installed on your TI-89 Titanium; the User-Defined ([F4]) menu option is grayed out if you have not created a function or program.

Note: Typing a letter takes you to the first command in the list starting with the same letter.

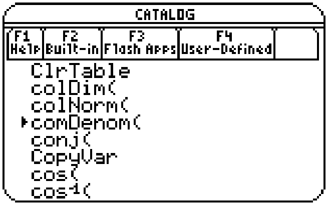
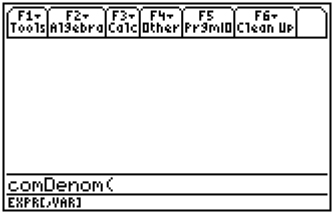
Press	Result
CATALOG (displays Built-in commands)	
F3 (displays Flash Apps commands, if any)	
F4 (displays User-Defined commands, if any)	

Select commands from the Catalog and insert them onto the calculator Home screen entry line or paste them to other Apps, such as the Y= Editor, Text Editor, or CellSheet™ Apps.

Example: Insert the **comDenom(** command on the calculator Home screen entry line.

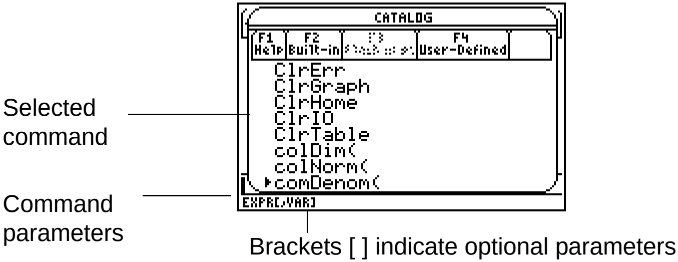
Note: Before selecting a command, position the cursor where you want the command to appear.

Pressing [2nd] [▼] advances the Catalog list one page at a time.

Press	Result
[CATALOG] **C** [2nd] [▼] Then press [▼] until the pointer is at the **comDenom(** function.	
[ENTER]	

The status line displays any required and optional parameters for the selected command. Optional parameters appear in square brackets.

Note: Pressing [F1] will also display the parameters for the selected command.



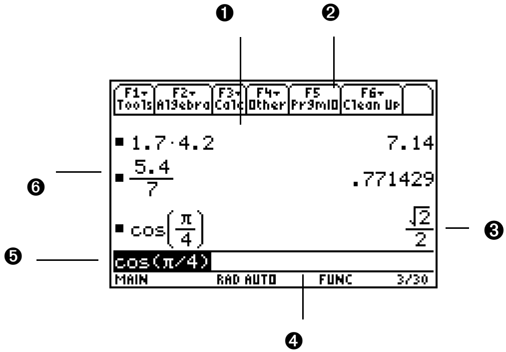
To exit the Catalog without selecting a command, press [ESC].

Calculator Home screen

The calculator Home screen is the starting point for math operations, including executing instructions, evaluating expressions, and viewing results.

To display the calculator Home screen, press: [HOME]

You can also display the calculator Home screen from the Apps desktop by highlighting the Home icon and pressing [ENTER].



❶ History area lists the entry/answer pairs entered.

❷ Tabs display menus for selecting lists of operations. Press [F1], [F2], and so on to display menus.

❸ Result of last entry is displayed here. (Note that results are not displayed on the entry line.)

❹ Status line shows the current state of the calculator.

❺ Entry line displays your current entry.

❻ Your previous entry is displayed here.

To return to the Apps desktop from the calculator Home screen, press [APPS].

The history area displays up to eight entry/answer pairs, depending on the complexity and height of the expressions. When the display is filled, information scrolls off the top of the screen. Use the history area to:

- Review previous entries and answers. Use the cursor keys to view entries and answers that have scrolled off the screen.
- Recall or auto-paste a previous entry or answer onto the entry line to reuse or edit. (For more information, see the electronic *Operating the Calculator* chapter.)

The cursor, which normally rests on the entry line, can be moved into the history area. The following table shows you how to move the cursor around in the history area.

To	Do this
View entries/answers scrolled off the screen	From the entry line, press ⊖ to highlight the last answer. Continue using ⊖ to move the cursor from answer to entry through the history area.
Go to the oldest or newest entry/answer pair	If the cursor is in the history area, press [♦] ⊖ or [♦] ⊖.
View an entry or answer too long for one line (▶ is displayed at the end of the line)	Move the cursor to the entry or answer. Use ⊖ or ⊖ to scroll left or right and [2nd] ⊖ or [2nd] ⊖ to go to the beginning or end.
Return cursor to the entry line	Press [ESC], or press ⊖ until the cursor is back on the entry line.

Interpreting history information on the status line

Use the history indicator on the status line for information about the entry/answer pairs. For example:

If the cursor is on the entry line:

Total number of pairs currently saved	8/30	Maximum number of pairs that can be saved

If the cursor is in the history area:

Pair number of the highlighted entry/answer	8/30	Total number of pairs currently saved

Modifying the history area

To change the number of pairs that can be saved:

1. From the calculator Home screen, press [F1] and select **9:Format**.
2. Press ▶ and use ▲ or ▼ to highlight the new number.
3. Press [ENTER] [ENTER].

To clear the history area and delete all saved pairs:

- From the calculator Home screen, press [F1] and select **8:Clear Home**.

  – or –

- Enter **ClrHome** on the calculator Home screen entry line.

To delete an entry/answer pair, move the cursor to either the entry or answer, and press [←] or [CLEAR].

Working with Apps

The TI-89 Titanium organizes Apps by category on the Apps desktop. To select a category, press a function key ([F2] through [2nd] [F8]). The App icons for the selected category appear on the Apps desktop.

Note: If the name under an Apps desktop icon is truncated, use the cursor keys to highlight the icon. Now view the full name at the top of the Apps desktop.

Opening Apps

Use the cursor keys or press the first letter of the App name to highlight the Apps icon on the Apps desktop and press [ENTER]. The App either opens directly or displays a dialog box. The most common dialog box lists these options for the App:

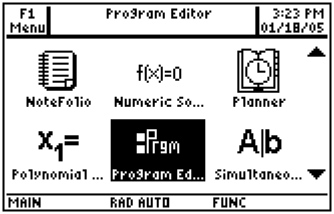
Note: The TI-89 Titanium uses the general term *variable* to refer to the App data files that you create.

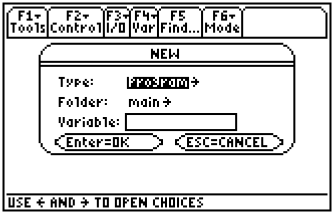
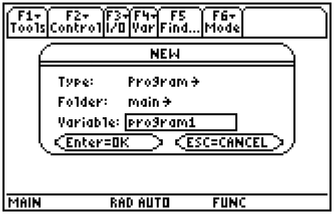
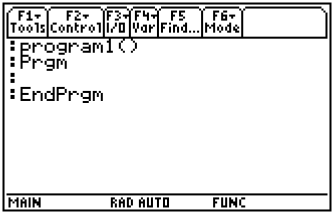
Option	Description
Current	Returns the screen displayed when you last viewed the App. If no current App variable exists, the New dialog box appears.
Open	Lets you open an existing file.

Option	Description
New	Creates a new file with the name typed in the field.

Select an option, enter any required information, and press [ENTER]. The App appears.

Example: Create a new program using the Program Editor.

Press	Result
Use cursor keys to highlight 	
[ENTER]	
3	

Press	Result
[ENTER]	
⊝ ⊝ **program1**	
[ENTER] [ENTER]	

The newly created program variable, *program1*, is saved to the Main folder.

Returning to the Apps desktop from within an App

Press [APPS]. The icons for the last Apps category selected appear on the Apps desktop with the icon for the last App opened highlighted.

You can also return to the Apps desktop by pressing [2nd] [QUIT] in full-screen mode. In split-screen mode, press [2nd] [QUIT] twice.

To return to the last open App from the Apps desktop, press [2nd] [↔].

Selecting an Apps category

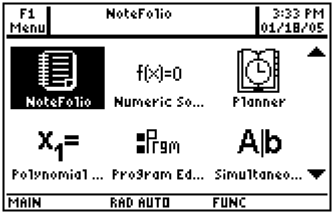
On the TI-89 Titanium, the Apps category names appear only in the **F1** Menu. To select an Apps category, press [F1] **2:Select Category** and use the cursor keys to highlight an Apps category, and then press [ENTER] to select the highlighted category. You can also use the function key shortcuts to select a category from the keypad (use the [2nd] key if necessary). The App icons for the selected category appear on the Apps desktop.

The App icons for the selected category appear on the Apps desktop.

Key	Description
[F2] All	Icons for all installed Apps displayed. Not customizable.
[F3] English	Customizable category. English is the default.
[F4] SocialSt	Customizable category. SocialSt (social studies) is the default.
[F5] Math	Customizable category. Math is the default.

Key	Description
[2nd] [F6] Graphing	Customizable category. Graphing is the default.
[2nd] [F7] Science	Customizable category. Science is the default.
[2nd] [F8] Organizr	Customizable category. Organizr (organizer) is the default.

Example: Select the All category.

Press	Result
[F2]	

If you select an Apps category containing no Apps, a message appears to confirm that the category is empty and point you to the [F1] **1:Edit Categories** menu, where you can add App shortcuts to the category. (See "Customizing the Apps categories" on page 31.)

Press [ENTER] or [ESC] to clear the message and return to the Apps desktop.

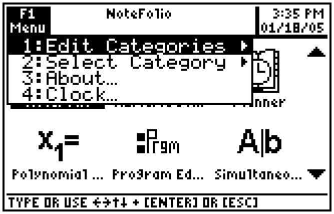
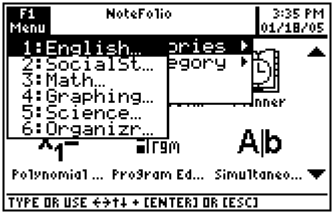
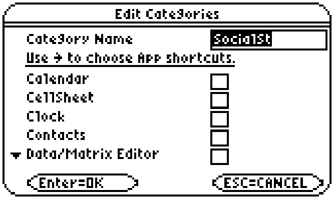
The TI-89 Titanium organizes your Apps into seven categories, six of which you can customize to fit your individual needs. (The All category contains every installed App and cannot be edited.)

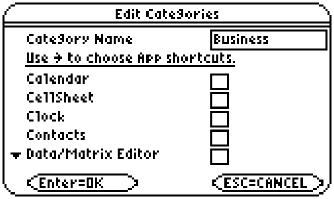
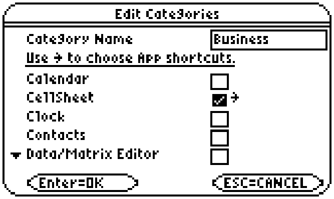
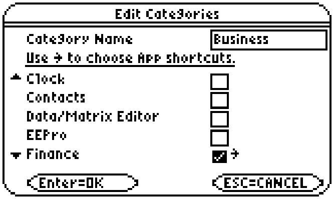
To customize the [F3] through [2nd] [F8] Apps categories:

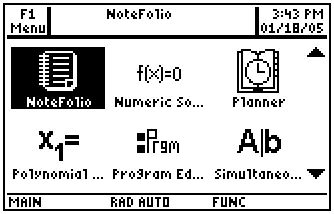
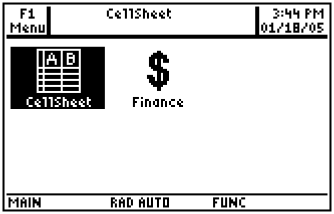
1. Select [F1] **1:Edit Categories.** A submenu displays the six customizable Apps category names. (The All category is not listed.)
2. Highlight an Apps category and press [ENTER]. The Edit Categories dialog box appears with a list of installed Apps and a text box with the category name highlighted.
3. To change the Apps category name, type the desired name.

   Note: Enter a name of up to eight characters, including letters with or without capitalization, numbers, punctuation, and accented characters.
4. To add or remove an App shortcut from the category, press ⊖ as required to highlight the box next to the App, then press ⊳ to add or remove the check mark (✓).
5. To save the changes and return to the Apps desktop, press [ENTER].

Example: Replace the Social Studies category with the Business category and add the CellSheet™ and Finance App shortcuts.

Press	Result
[F1]	
▶	
2 – or – ▼ [ENTER]	

<table>
<tr><th>Press</th><th>Result</th></tr>
<tr><td>[2nd] [a-lock]
[↑] B u s i n e s s</td><td>
</td></tr>
<tr><td>▼
⋮
▶</td><td>
</td></tr>
<tr><td>▼
⋮
▶</td><td>
</td></tr>
</table>

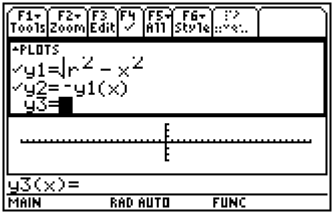
Press	Result
ENTER	
F4	

Open Apps and split-screen status

Your TI-89 Titanium lets you split the screen to view two Apps simultaneously. For example, view the Y= Editor and Graph screens simultaneously to see the list of functions and how they are graphed.

Select the Split Screen mode from Page 2 of the MODE screen. The TI-89 Titanium displays the selected Apps in the split-screen view as shown. Split the screen horizontally (top-bottom) or vertically (left-right).

Top-bottom split screen

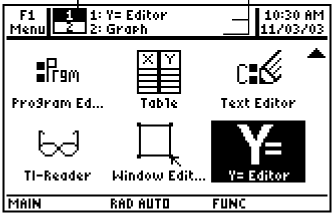


To return to the Apps desktop, press [APPS]. The split-screen status appears at the top of the Apps desktop with the names of the open Apps and the portions of the screen in which each is displayed. The highlighted numeral indicates the split-screen portion where the next App you open will appear.

Note: The Apps desktop always appears in the full-screen view.

Split-screen status (highlight indicates the portion where the next App selected will open.)

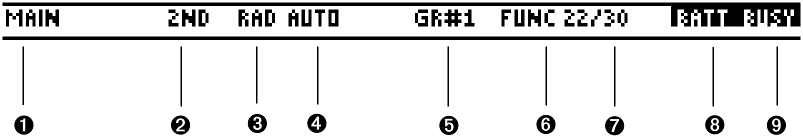
Names of open Apps



More information is available about using split screens. (For more information, see the electronic *Split Screens* chapter.)

Checking status information

Look to the status line, located at the bottom of the screen, for information about the current state of your TI-89 Titanium.



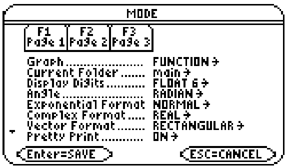
Indicator	Meaning
❶ Current folder	Name of the selected folder (MAIN is the default folder.)
❷ Modifier key	Selected modifier key ([2nd], [♦], [↑]), if any.
❸ Angle mode	Selected units in which angle values are displayed and interpreted (RAD, DEG, GRAD)
❹ Exact/Approx mode	Mode in which answers are calculated and displayed (AUTO, EXACT, APPROX)
❺ Graph number	Active of two independent graphs in split-screen mode (GR#1, GR#2)
❻ Graph mode	Selected type of graph that can be plotted (FUNC, PAR, POL, SEQ, 3D, DE)

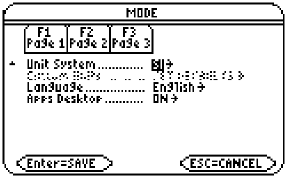
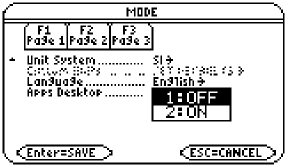
Indicator	Meaning
❼ Entry/Answer pairs	22/30–Number of entry/answer pairs (default is 30, maximum is 99) in the history area of the calculator Home screen.
❽ Replace batteries	Displayed when batteries are low (BATT). If BATT is highlighted with a black background, change the batteries as soon as possible (BATT).
❾ Busy/Pause, Locked/Archived variable	BUSY–Calculation or graph is in progress PAUSE–You paused a graph or program 🔒–Variable opened in the current editor is locked or archived and cannot be modified

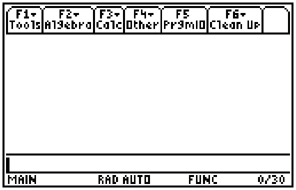
Turning off the Apps desktop

You can turn off the Apps desktop from the MODE dialog box. If you do, open Apps from the APPLICATIONS menu. To open the APPLICATIONS menu, press [APPS].

Example: Turn off the Apps desktop.

Press	Result
[MODE]	

<table>
<tr><th>Press</th><th>Result</th></tr>
<tr><td>[F3]</td><td>
</td></tr>
<tr><td>↓ ↓ → ↑</td><td>
</td></tr>
<tr><td>[ENTER] [ENTER]

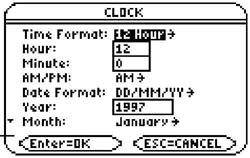
Note: The previous open App appears (in this example, the calculator Home screen).</td><td>
</td></tr>
</table>

To turn on the Apps desktop, repeat the procedure, selecting ON in the Apps Desktop mode field. To return to the Apps desktop from the calculator Home screen, press [APPS].

Using the clock

Use the **CLOCK** dialog box to set the time and date, select the clock display format, and turn the clock off and on.

The clock is turned on by default. If you turn off the clock, all Clock dialog box options except Clock ON/OFF are grayed out.



▼ indicates you can scroll down for more options)

Displaying the CLOCK dialog box

1. Use the cursor keys to highlight the Clock icon on the Apps desktop.
2. Press [ENTER]. The CLOCK dialog box appears with the Time Format field highlighted.

Note: Because the CLOCK dialog box displays the settings current at the time you open the dialog box, you might need to update the time before exiting.

Setting the time

1. Press ▶ to open the list of time formats.
2. Press ▲ or ▼ to highlight an option, then press [ENTER]. The selected format appears in the Time Format field.
3. Press ▼ to highlight the Hour field.
4. Type the hour, then press ▼ to highlight the Minute field.
5. Type the minute(s).

6. If the time format is 24 hours, proceed to step 9.

   — or —

   If the time format is 12 hours, press ⊝ to highlight the AM/PM field.
7. Press ⓘ to open the list of AM/PM options.
8. Press ⊜ or ⊝ to highlight an AM/PM option, then press [ENTER]. The selected AM/PM option appears.
9. Set the date (for procedures, see *Setting the date*).

   — or —

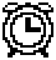
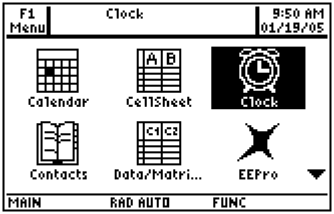
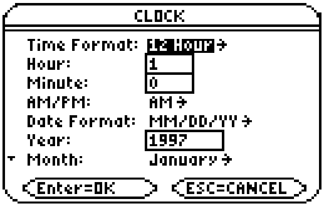
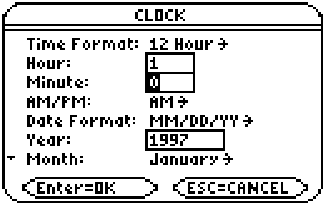
   To save your settings and exit, press [ENTER]. The time is updated in the top right corner of the Apps desktop.

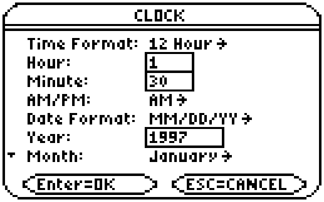
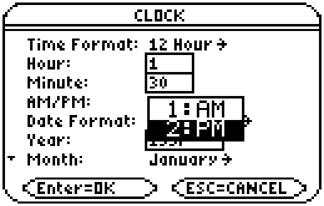
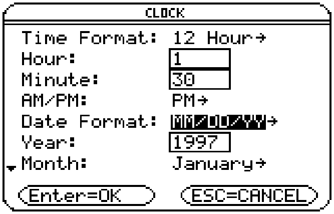
Setting the date

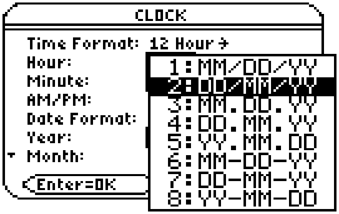
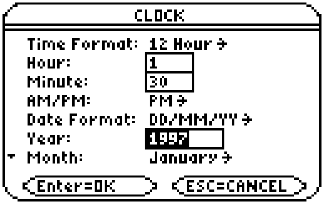
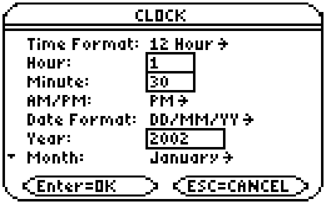
1. Press ⊜ or ⊝ as required to highlight the Date Format field.
2. Press ⓘ to open the list of date formats.
3. Press ⊜ or ⊝ to highlight an option, then press [ENTER]. The selected format appears in the Date Format field.
4. Press ⊝ to highlight the Year field.
5. Type the year, then press ⊝ to highlight the Month field.
6. Press ⓘ to open the list of months.
7. Press ⊜ or ⊝ to highlight an option, then press [ENTER]. The selected month appears in the Month field.
8. Press ⊝ to highlight the Day field.

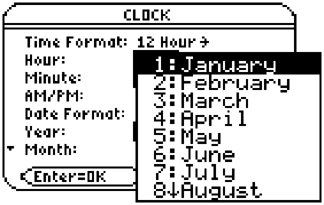
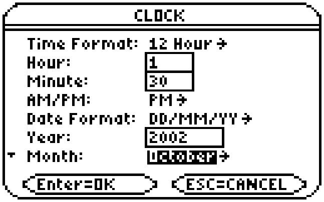
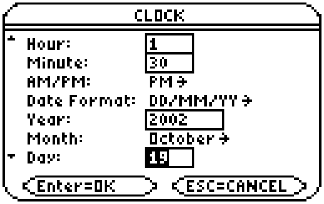
9. Type the day, then press [ENTER] [ENTER] to save your settings and exit. The date is updated in the top right corner of the Apps desktop.

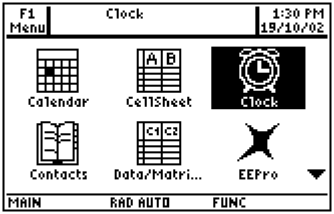
Example: Set the time and date to 19/10/02 (October 19, 2002) at 1:30 p.m.

Press	Result
Use cursor keys to highlight 	Time and date 
[ENTER]	
⊖ **1** ⊖	

Press	Result
3 0 ↓	
→ ↓	
ENTER ↓	

<table>
<tr><th>Press</th><th>Result</th></tr>
<tr><td>▶ ▼</td><td>
</td></tr>
<tr><td>[ENTER] ▼</td><td>
</td></tr>
<tr><td>2 0 0 2</td><td>
</td></tr>
</table>

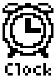
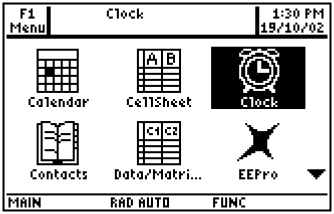
Press	Result
▼ ▶	
Scroll down to October and press [ENTER]	
▼ **1 9**	

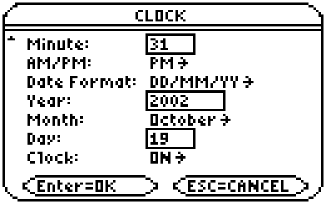
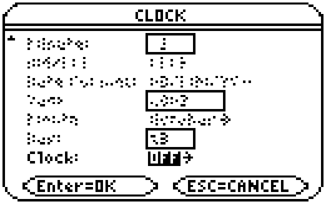
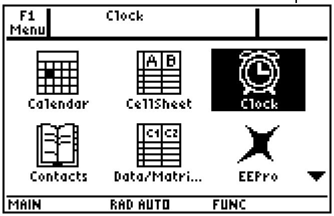
Press	Result
[ENTER] [ENTER]	Revised time and date 

Turning off the clock

From the Apps desktop, open the CLOCK dialog box and select OFF in the Clock field.

Example: Turn off the clock.

Press	Result
Use cursor keys to highlight 	Clock on 

Press	Result
[ENTER] Scroll down to the Clock field.	
▶ ▲ [ENTER]	
[ENTER]	Clock off 

To turn on the clock, repeat the procedure, selecting ON in the Clock field. Remember to reset the time and date.

Using menus

To select most TI-89 Titanium menus, press the function keys corresponding to the toolbars at the top of the calculator Home screen and most App screens. Select other menus using key commands.

Toolbar menus

The starting point for TI-89 Titanium math operations, the calculator Home screen displays toolbar menus that let you choose math-related options.

Toolbar menus also appear at the top of most App screens. These menus list common functions of the active App.

Other menus

Use key commands to select the following menus. These menus contain the same options regardless of the screen displayed or the active App.

Press	To display
2nd [CHAR]	CHAR menu. Lists characters not available on the keyboard; characters are organized by category (Greek, math, punctuation, special, and international).
2nd [MATH]	MATH menu. Lists math operations by category.

Press	To display
[APPS]	APPLICATIONS menu. Lists the installed Apps. (Menu is available only when the Apps desktop is turned off; Apps are normally accessed from the Apps desktop.)
[◆] [APPS]	FLASH APPLICATIONS menu. Lists the installed Flash Apps. (Menu is available only when Apps desktop is turned off; Flash Apps are normally accessed from the Apps desktop.)

Selecting menu options

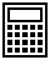
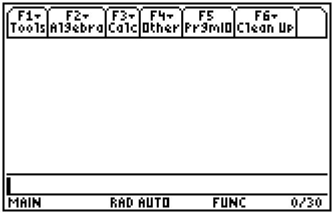
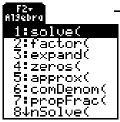
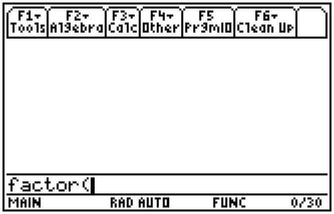
- Press the number or letter to the left of the option you want to select.

  — or —

- Press ⊖ or ⊖ to select the option, and press [ENTER].

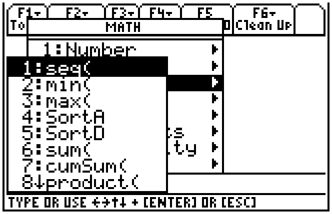
Note: If the first menu option is selected, press ⊖ to select the last option on the menu. If the last menu option is selected, press ⊖ to select the first option on the menu.

Example: Select **factor(** from the Algebra menu on the calculator Home screen.

Press	Result
Press: [HOME] – or – From the Apps desktop, use the cursor keys to highlight  and press [ENTER]	
[F2]	 ▾ indicates Algebra menu will open when you press [F2].
2 – or – ⊝ [ENTER]	

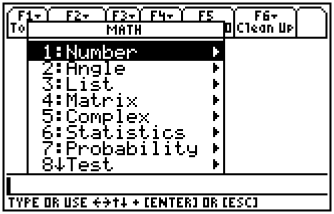
Selecting submenu options

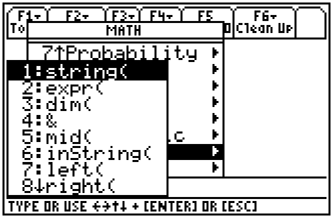
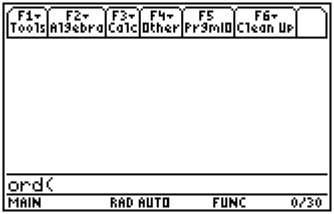
A small arrow symbol (▶) to the right of a menu option indicates that selecting the option will open a submenu.



↓ points to additional options.

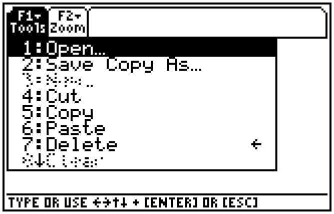
Example: Select **ord(** from the MATH menu on the calculator Home screen.

Press	Result
[2nd] [MATH]	

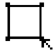
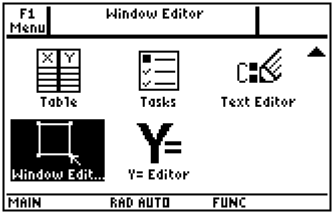
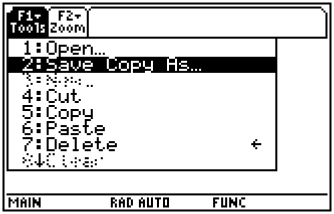
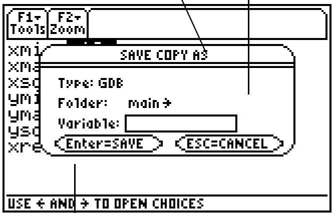
Press	Result
D – or – ↑ ↑ →	
B – or – ↑ ENTER	

Using dialog boxes

An ellipsis (…) at the end of a menu option indicates that choosing the option will open a dialog box. Select the option and press ENTER.



Example: Open the **SAVE COPY AS** dialog box from the Window Editor.

Press	Result
[APPS] Use the cursor keys to highlight  and press [ENTER]	
[F1]	
2 – or – ⊙ [ENTER]	Press ⊙ to display a list of folders. Type the name of the variable.  Press [ENTER] twice to save and close the dialog box.

Note: Pressing the [♦] S key shortcut also opens the SAVE COPY AS dialog box in most Apps.

Canceling a menu

To cancel a menu without making a selection, press [ESC].

Moving among toolbar menus

To move among the toolbar menus without selecting a menu option:

- Press the function key ([F1] through [F8]) of a toolbar menu.
- Press a function key, then press ▶ or ◀ to move from one toolbar menu to the next. Press ▶ from the last menu to move to the first menu. Press ◀ to move from the first menu to the last menu.

Note: If you press ▶ when a menu option with a submenu is selected, the submenu will appear instead of the next toolbar menu. Press ▶ again to move to the next menu.

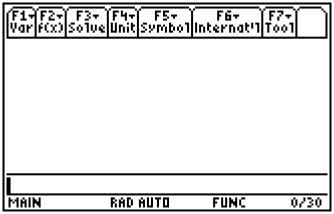
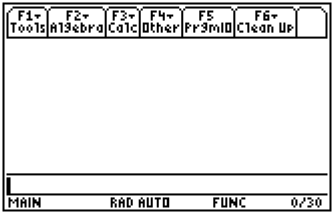
More information is available about menus. (See the electronic *Operating the Calculator* chapter.)

Custom menu

The custom menu provides quick access to your most commonly used options. Use the default custom menu or create your own using the Program Editor. You can include any available TI-89 Titanium command or character.

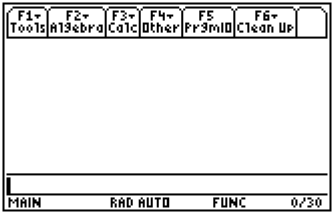
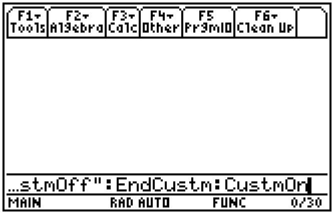
The custom menu replaces the standard toolbar menu on the calculator Home screen. (For details on creating a custom menu, see the electronic *Programming* chapter.) More information is available about custom menus. (See the electronic *Operating the Calculator* chapter.)

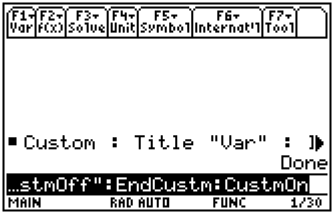
Example: Turn on and turn off the custom menu from the calculator Home screen.

Press	Result
[2nd] [CUSTOM]	Default custom menu 
[2nd] [CUSTOM]	Normal toolbar menu 

Example: Restore the default custom menu.

Note: Restoring the default custom menu erases the previous custom menu. If you created the previous custom menu with a program, you can run the program again to reuse the menu.

Press	Result
[2nd] [CUSTOM] (to turn off the custom menu and turn on the standard toolbar menu)	
[2nd] [F6]	
3 – or – ⊖ ⊖ [ENTER]	

Press	Result
[ENTER]	

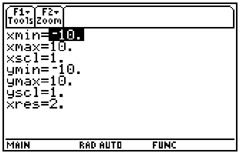
Opening Apps with the Apps desktop turned off

If you turn off the Apps desktop, use the APPLICATIONS menu to open Apps. To open the APPLICATIONS menu with the Apps desktop off, press [APPS].

Note: If you press [APPS] with the Apps desktop turned on, the Apps desktop will appear instead of the APPLICATIONS menu.

Example: With the Apps desktop turned off, open the Window Editor from the APPLICATIONS menu.

Press	Result
[APPS]	

Press	Result
3 – or – ▼ ▼ [ENTER]	

To access Apps not listed on the APPLICATIONS menu, select **1:FlashApps**.

Using split screens

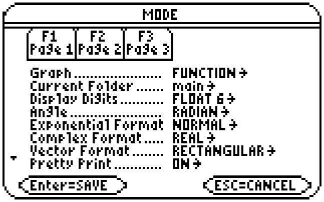
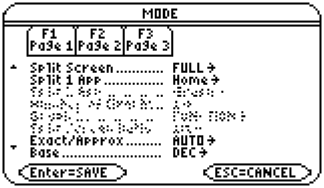
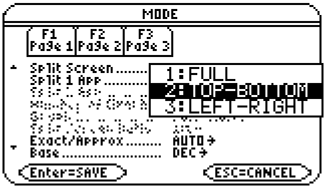
The TI-89 Titanium lets you split the screen to show two Apps at the same time. For example, display both the Y= Editor and Graph screens to compare the list of functions and how they are graphed.

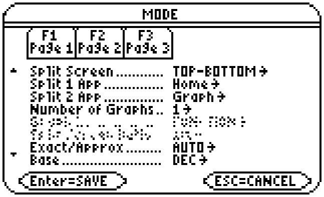
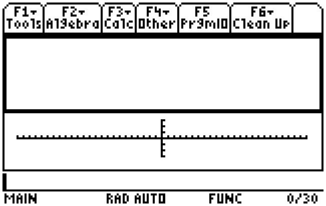
Setting split-screen mode

You can split the screen either top to bottom or left to right from the MODE dialog box. The split-screen setting stays in effect until you change it.

1. Press [MODE] to display the MODE dialog box.
2. Press [F2] to display the Split Screen mode setting.
3. Press ▶ to open the Split Screen mode menu.
4. Press ▼ as required to highlight either TOP-BOTTOM or LEFT-RIGHT.
5. Press [ENTER]. The Split Screen mode setting displays the option you selected.
6. Press [ENTER] again to save this change and display the split screen.

Example: Set split-screen mode to TOP-BOTTOM.

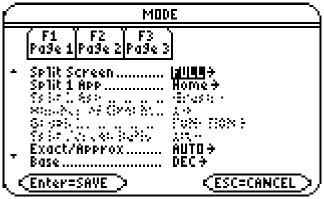
Press	Result
MODE	
F2	
▶ ▼	

Press	Result
[ENTER]	
[ENTER]	

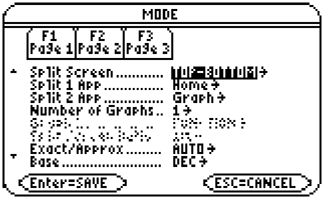
Setting the initial Apps for split screen

After you select either TOP-BOTTOM or LEFT-RIGHT split-screen mode, additional mode settings become available.

Full-screen mode



Split-screen mode

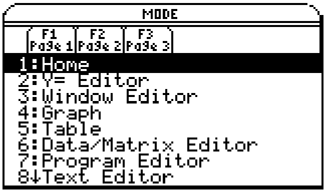


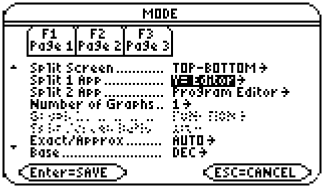
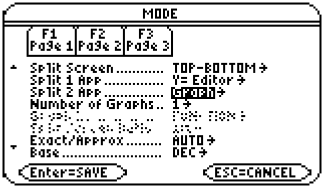
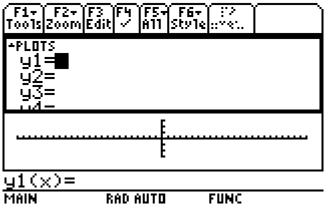
Mode	Description
Split 2 App	Lets you specify the App displayed in the bottom or right portion of the split screen. Works together with Split 1 App, which lets you specify the App displayed in the top or left portion of the split screen.
Number of Graphs	Lets you set up and display two independent graphs.

To set the initial App for each split-screen portion:

1. Select the Split 1 App mode setting and press ▶ to display a menu of available Apps. (See “Setting split-screen mode” on page 57.)
2. Press ▼ or ▲ to highlight the App and press [ENTER].
3. Repeat steps 1 and 2 for the Split 2 App mode setting.

Example: Display the Y= Editor in the top screen and the Graph App in the bottom screen.

Press	Result
▼ ▶	

Press	Result
2	
▼ ▶	
4	
ENTER	

If you set Split 1 App and Split 2 App to the same nongraphing App or to the same graphing App with Number of Graphs set to 1, the TI-89 Titanium exits split-screen mode and displays the App in full-screen mode.

Selecting the active App

In split-screen mode, only one App can be active at a time.

- To switch between active Apps, press [2nd] [⇹].
- To open a third App, press [APPS] and select the App. This App replaces the active split-screen App.

Exiting split-screen mode

Exit split-screen mode in any of the following ways:

- Press [2nd] [QUIT] to close the active App and display the full-screen view of the other open App.
- If the Apps desktop is turned off, pressing [2nd] [QUIT] replaces the active split-screen App with the calculator Home screen. Pressing [2nd] [QUIT] again turns off the split-screen mode and displays the calculator Home screen in full-screen mode.
- Select Split Screen on Page 2 of the MODE dialog box, set split-screen mode to FULL, and press [ENTER].
- Press [2nd] [QUIT] twice to display the Apps desktop

More information is available about using split screens. (See the electronic *Split Screens* chapter.)

Managing Apps and operating system (OS) versions

Using the TI-89 Titanium connectivity features, you can download Apps from:

- The TI Educational & Productivity Solutions (E&PS) Web site at: education.ti.com/latest
- The CD-ROM included with your TI-89 Titanium.
- A compatible graphing calculator.

Adding Apps to your TI-89 Titanium is like loading software on a computer. All you need is TI Connect™ software and the USB computer cable that came with your TI-89 Titanium.

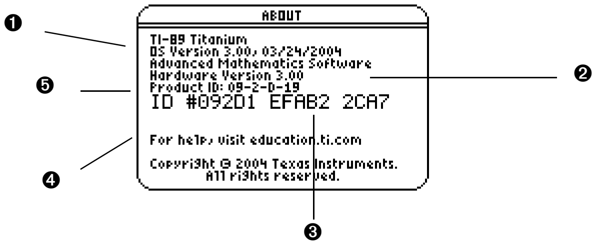
For system requirements and instructions to link to compatible calculators and download TI Connect software, Apps, and OS versions, see the TI E&PS Web site.

Before downloading Apps to your TI-89 Titanium, please read the license agreement on the CD-ROM or TI Web site.

Finding the OS version and identification (ID) numbers

If you purchase software from the TI E&PS Web site or call the customer support number, you will be asked to provide information about your TI-89 Titanium. You will find this information on the ABOUT screen.

To display the ABOUT screen, press [F1] **3:About** from the Apps desktop. The ABOUT screen displays the following information about your TI-89 Titanium:



❶ OS version

❷ Hardware version

❸ Unit ID (required to obtain certificates for installing purchased Apps). Similar to a serial number. Write this number down and keep it in a safe place in case the calculator is ever lost or stolen.

❹ Apps certificate revision number (Cert. Rev.)

❺ Product identifier (Product ID). Similar to a model number.

Note that your screen will be different than the one shown above.

Deleting an application removes it from the TI-89 Titanium and increases space for other applications. Before deleting an application, consider storing it on a computer for reinstallation later.

1. Quit the application.
2. Press [2nd] [VAR-LINK] to display the VAR-LINK (All) screen.
3. Press [2nd] [F7] to display the list of installed applications.
4. Select the application you want to delete by pressing [F4]. (Press [F4] again to deselect.)
5. Press [F1] **1:Delete**. The VAR-LINK delete confirmation dialog box displays.
6. Press [ENTER] to delete the application.

Note: Only Flash Apps can be deleted.

Connecting your TI-89 Titanium to other devices

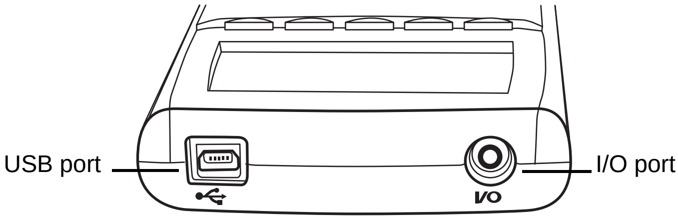
The TI-89 Titanium includes both a mini-USB port and a standard I/O port. Ports are used to link two compatible graphing calculators or connect to a computer or peripheral device.

In addition, the teacher model of the TI-89 Titanium includes an accessory port. This port is used to output visual data so that a classroom can view the calculator’s display on a video device or overhead screen.

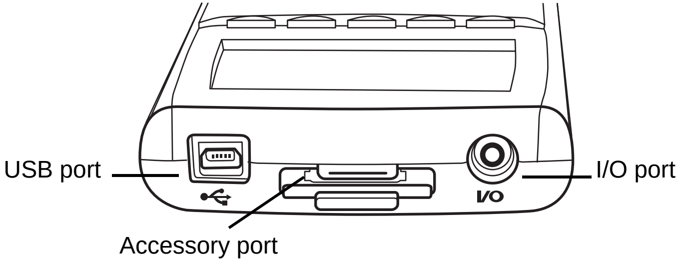
To connect your calculator to a computer – Connect your TI-89 Titanium using the USB port and the included USB computer cable.

To connect your calculator to another calculator – Use the USB unit-to-unit cable or an I/O unit-to-unit cable to connect the TI-89 Titanium to a compatible graphing calculator or peripheral device, such as a TI-89 or TI-92 Plus graphing calculator or the CBL 2™ and CBR™ systems.

To show your calculator's display to the classroom – Use the accessory port to connect the TI-Presenter™ video adapter to the teacher model of the TI-89 Titanium. The TI-Presenter video adapter provides a video interface between the calculator and video display or recording devices. Or use the accessory port to connect the TI ViewScreen™ overhead panel to your calculator. The TI ViewScreen overhead panel enlarges and projects the display so an entire class can view it. For more information about the TI-Presenter video adapter and TI ViewScreen panel, see the TI E&PS Web site at education.ti.com.



TI-89 Titanium ports



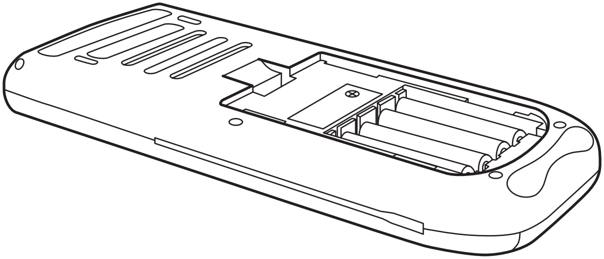
TI-89 Titanium ports (teacher model)

Batteries

The TI-89 Titanium uses four AAA alkaline batteries and a button cell backup battery. The backup battery is already installed, and the AAA batteries are provided with your product.

Installing the AAA Batteries

1. Remove the battery cover from the back of the calculator.
2. Unwrap the four AAA batteries provided with your product and insert them in the battery compartment. Arrange the batteries according to the polarity (+ and -) diagram in the battery compartment.



3. Replace the battery cover on the calculator. The cover should snap into place.

Replacing the AAA (alkaline) batteries

As the batteries lose power, the display begins to dim, especially during calculations. If you find yourself increasing the contrast frequently, replace the AAA alkaline batteries.

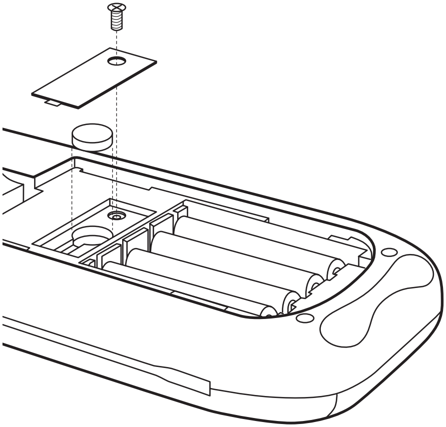
The status line also gives battery information.

Indicator	Meaning
BATT	Batteries are low.
BATT (highlighted)	Replace batteries as soon as possible.

Before replacing the batteries, turn off the TI-89 Titanium by pressing [2nd] [OFF] to avoid losing information stored in memory. Do not remove both the back-up battery and the AAA alkaline batteries at the same time.

Replacing the backup battery

1. To replace the backup battery, remove the battery cover and unscrew the tiny screw holding the BACK UP BATTERY cover in place.



2. Remove the old battery and install a new button cell battery, positive (+) side up. Replace the cover and the screw.

Important OS download information

New batteries should be installed before beginning an OS download.

When in OS download mode, the APD™ feature does not function. If you leave your calculator in download mode for an extended time before you actually start the download, your batteries may become depleted. You will then need to replace the depleted batteries with new batteries before downloading.

You can also transfer the OS to another TI-89 Titanium using a USB unit-to-unit cable . If you accidentally interrupt the transfer before it is complete, you will need to reinstall the OS via a computer. Again, remember to install new batteries before downloading.

Please contact Texas Instruments as described in Service & Support Information, if you experience a problem.

Battery Precautions

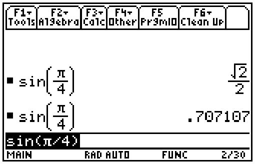
Take these precautions when replacing batteries:

- Do not leave batteries within the reach of children.
- Do not mix new and used batteries. Do not mix brands (or types within brands) of batteries.
- Do not mix rechargeable and non-rechargeable batteries.
- Install batteries according to polarity (+ and –) diagrams.
- Do not place non-rechargeable batteries in a battery recharger.
- Properly dispose of used batteries immediately.
- Do not incinerate or dismantle batteries.

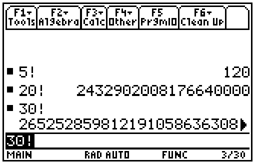
Performing Computations

This section provides several examples for you to perform from the Calculator Home screen that demonstrate some of the computational features of the TI-89 Titanium. The history area in each screen was cleared by pressing [F1] and selecting **8:Clear Home**, before performing each example, to illustrate only the results of the example’s keystrokes.

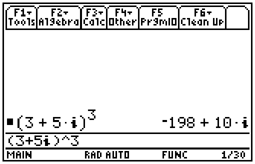
Showing Computations

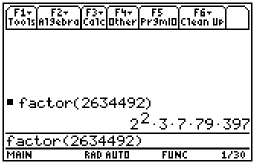
Steps and keystrokes	Display
Compute **sin(π/4)** and display the result in symbolic and numeric format. To clear the history area of previous calculations, press [F1] and select **8:Clear Home**. [2nd] [SIN] [2nd] [π] [÷] 4 [)] [ENTER] [♦] [≈]	

Finding the Factorial of Numbers

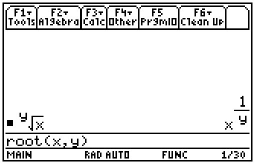
Steps and keystrokes	Display
Compute the factorial of several numbers to see how the TI-89 Titanium handles very large integers. To get the factorial operator (!), press [2nd] [MATH], select **7:Probability**, and then select **1:!**. 5 [2nd] [MATH] 7 1 [ENTER] 20 [2nd] [MATH] 7 1 [ENTER] 30 [2nd] [MATH] 7 1 [ENTER]	

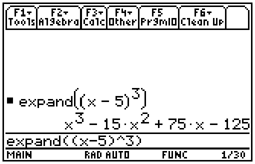
Expanding Complex Numbers

Steps and keystrokes	Display
Compute $(3+5i)^3$ to see how the TI-89 Titanium handles computations involving complex numbers. Press [(] 3 [+] 5 [2nd] [i] [)] [^] 3 [ENTER]	

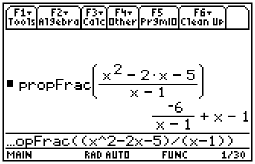
Steps and keystrokes	Display
Compute the factors of the rational number 2634492. You can enter “factor” on the entry line by typing **FACTOR** on the keyboard, or by pressing [F2] and selecting **2:factor(**. Press [F2] 2 2634492 [)] [ENTER] (*Optional*) Enter other numbers on your own.	

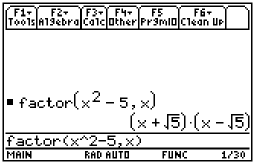
Finding Roots

Steps and keystrokes	Display
Find the root of the expression (x,y). You can enter “root” on the entry line by typing **ROOT** on the keyboard, or by pressing [♦] 9. This example illustrates using the root function and how the expression is displayed in “pretty print” in the history area. Press [♦] 9 X [,] Y [)] [ENTER]	

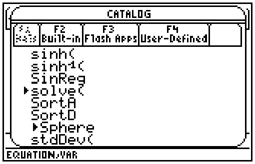
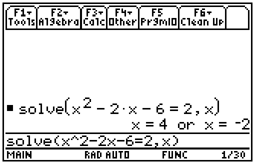
Steps and keystrokes	**Display**
Expand the expression $(x-5)^3$. You can enter “expand” on the entry line by typing **EXPAND** on the keyboard, or by pressing [F2] and selecting **3:expand(.** Press [F2] 3 [(] X [−] 5 [)] [^] 3 [)] [ENTER] (*Optional*) Enter other expressions on your own.	

Reducing Expressions

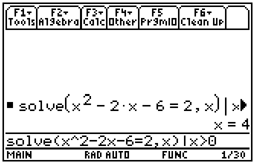
Steps and keystrokes	**Display**
Reduce the expression $(x^2-2x-5)/(x-1)$ to its simplest form. You can enter “propFrac” on the entry line by typing **PROPFRAC** on the keyboard, or by pressing [F2] and selecting **7:propFrac(**. Press [F2] 7 [(] X [^] 2 [−] 2 X [−] 5 [)] [÷] [(] X [−] 1 [)] [)] [ENTER]	

Steps and keystrokes	Display
Factor the polynomial (x^2-5) with respect to x. You can enter “factor” on the entry line by typing **FACTOR** on the keyboard or by pressing F2 and selecting **2:factor(**. Press F2 2 X ^ 2 − 5 , X) ENTER	

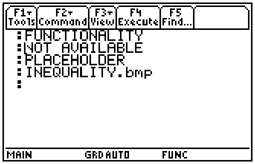
Solving Equations

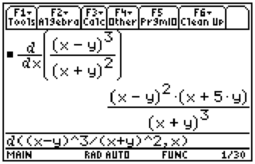
Steps and keystrokes	Display
Solve the equation $x^2-2x-6=2$ with respect to x. You can enter “**solve(**” on the entry line by selecting “**solve(**” from the Catalog menu, by typing **SOLVE(** on the keyboard, or by pressing F2 and selecting **1:solve(**. The status line area shows the required syntax for the marked item in the **Catalog** menu. Press F2 1 X ^ 2 − 2 X − 6 = 2 , X) ENTER	 

Solving Equations with a Domain Constraint

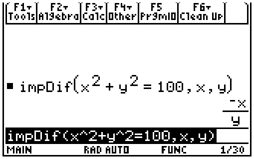
Steps and keystrokes	Display
Solve the equation $x^2-2x-6=2$ with respect to x where x is greater than zero. The "*with*" (I) operator provides domain constraint. [F2] 1 X [^] 2 [-] 2 X [-] 6 [=] 2 [,] X [)] [I] X [2nd] [>] 0 [ENTER]	

Solving Inequalities

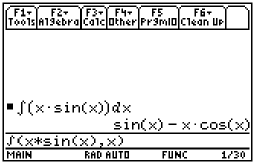
Steps and keystrokes	Display
Solve the inequality $(x^2>1,x)$ with respect to x. Press[F2] 1 X [^] 2 [2nd] [>] 1 [)] [ENTER]	

Steps and keystrokes	Display
Find the derivative of $(x-y)^3/(x+y)^2$ with respect to x. This example illustrates using the calculus differentiation function and how the function is displayed in “pretty print” in the history area. Press [2nd] [d] [(] X [−] Y [)] [^] 3 [÷] [(] X [+] Y [)] [^] 2 [,] X [)] [ENTER]	

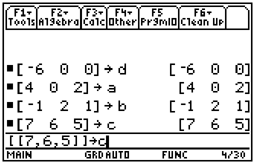
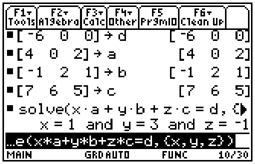
Finding Implicit Derivatives

Steps and keystrokes	Display
Compute implicit derivatives for equations in two variables in which one variable is defined implicitly in terms of another. This example illustrates using the calculus implicit derivative function. Press [F3] D X [^] 2 [+] Y [^] 2 [=] 100 [,] X [,] Y [)] [ENTER]	

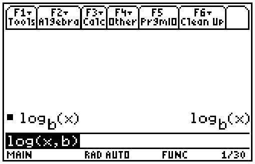
Finding the Integral of Functions

Steps and keystrokes	Display
Find the integral of **x∗sin(x)** with respect to x. This example illustrates using the calculus integration function. [2nd] [∫] X [×] [2nd] [SIN] X [)] [,] X [)] [ENTER]	

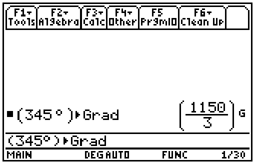
Solving Problems Involving Vectors

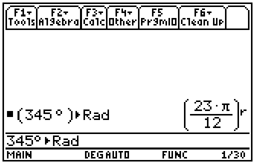
Steps and keystrokes	Display
1. Input a row or column of vectors. Press [[] [(-)] 6 [,] 0 [,] 0 []] [STO▸] d [[] 4 [,] 0 [,] 2 []] [STO▸] a [[] [(-)] 1 [,] 2 [,] 1 []] [STO▸] b [[] 7 [,] 6 [,] 5 []] [STO▸] c	
2. Solve (x* a+y*b+z*c=d {x,y,z}) Press [F2] 1 X [×] a [+] y [×] b [+] z [×] c [=] d [{] X [,] Y [,] Z [}] [)] [ENTER]	

Log to Any Base

Steps and keystrokes	Display
Find log (x,b). You can enter “log” on the entry line by typing **LOG** on the keyboard, or by pressing [♦] 7. Press[♦] 7 X , [alpha] b [)] [ENTER]	

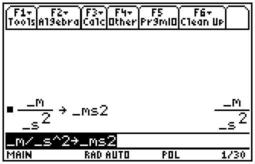
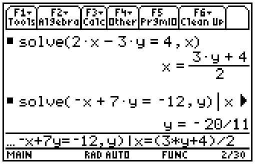
Converting Angle Measures

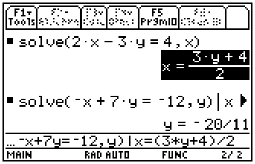
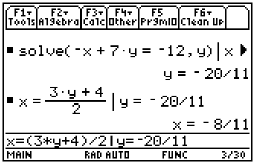
Steps and keystrokes	Display
1. Display the **MODE** dialog box. For **Angle** mode select **DEGREE**. Convert 345 degrees to Gradian angle measure. You can enter “▶**GRAD**” on the entry line by selecting “▶**GRAD**” from the Catalog menu, or from the Math menu by pressing [2nd] [MATH] and selecting **2:angle, A:▶GRAD.** Press [MODE] ⊖ ⊖ ⊖ ⊙ 2 [ENTER] 345 [2nd] [°] [2nd] [MATH] 2 [alpha] A [ENTER]	

Steps and keystrokes	Display
2. Convert 345 degrees to Radian angle measure. You can enter “▶**RAD**” on the entry line by selecting “▶**RAD**” from the Catalog menu, or from the Math menu by pressing [2nd] [MATH] and selecting **2:angle, B:▶RAD.** **Note:** You can also use °, ^{r}, or ^{G} to override the angle mode setting temporarily.	

Symbolic Manipulation

Solve the system of equations $2x - 3y = 4$ and $^-x + 7y = {}^-12$. Solve the first equation so that x is expressed in terms of y. Substitute the expression for x into the second equation, and solve for the value of y. Then substitute the y value back into the first equation to solve for the value of x.

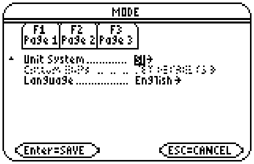
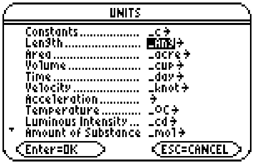
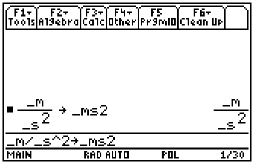
Steps and keystrokes	Display
1. Display the Home screen and clear the entry line. Solve the equation 2x − 3y = 4 for x. [F2] **1** selects **solve(** from the Algebra menu. You can also type **solve(** directly from the keyboard or select it from the **Catalog.** [HOME] [CLEAR] [CLEAR] [F2] 1 2 X [-] 3 Y [=] 4 [,] X [)] [ENTER]	
2. Begin to solve the equation ⁻x + 7y = ⁻12 for y, but do not press [ENTER] yet. Press [F2] 1 [(-)] X [+] 7 Y [=] [(-)] 12 [,] Y [)]	
3. Use the "*with*" operator to substitute the expression for x that was calculated from the first equation. This gives the value of y. The "*with*" operator is displayed as **|** on the screen. Use the auto-paste feature to highlight the last answer in the history area and paste it to the entry line. [|] [⬆] [ENTER] [ENTER]	

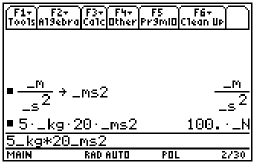
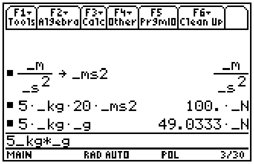
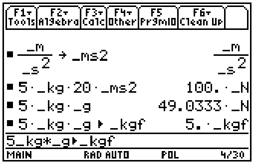
Steps and keystrokes	Display
4. Highlight the equation for x in the history area. Press [▲] [▲] [▲]	
5. Auto-paste the highlighted expression to the entry line. Then substitute the value of y that was calculated from the second equation. [ENTER] [|] [▲] [ENTER] [ENTER] The solution is: x = ⁻8/11 and y = ⁻20/11	

This example is a demonstration of symbolic manipulation. A one-step function is available for solving systems of equations.

Constants and Measurement Units

Using the equation f = m∗a, calculate the force when m = 5 kilograms and a = 20 meters/second^2. What is the force when a = 9.8 meters/second^2. (This is the acceleration due to gravity, which is a constant named _g). Convert the result from newtons to kilograms of force.

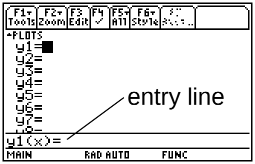
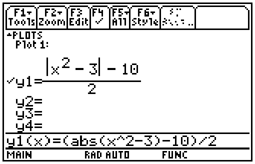
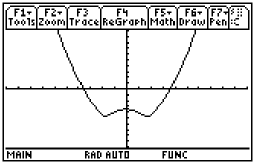
Steps and keystrokes	Display
1. Display the **MODE** dialog box, Page 3. For **Unit System** mode, select **SI** for the metric system of measurements. Results are displayed according to these default units. Press [MODE] [F3] ▶ 1 [ENTER]	
2. Create an acceleration unit for meters/second^2 named _ms2. The **UNITS** dialog box lets you select units from an alphabetical list of categories. You can use [2nd] ▼ and [2nd] ▲ to scroll one page at a time through the categories. If you use the **UNITS** dialog box to select a unit, the _ is entered automatically. Now, instead of re-entering _m/_s^2 each time you need it, you can use _ms2. Also, you can now use the **UNITS** dialog box to select _ms2 from the Acceleration category. [2nd] [UNITS] ▼ ▶ M [ENTER] [÷] [2nd] [UNITS] ▼ ▼ ▼ ▼ ▶ S [ENTER] [^] 2 [STO▶] [♦] [_] [2nd] [a-lock] MS [alpha] 2 [ENTER]	 

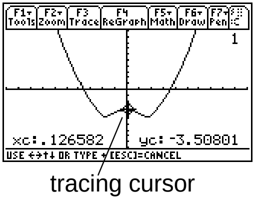
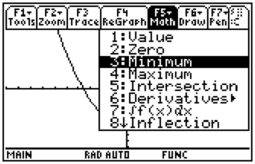
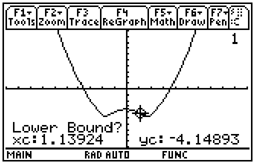
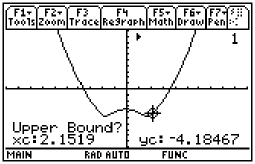
Steps and keystrokes	Display
3. Calculate the force when m = 5 kilograms (_kg) and a = 20 meters/second^2 (_ms2). If you know the abbreviation for a unit, you can type it from the keyboard. 5 [♦] [_] [2nd] [a-lock] KG [alpha] [×] 20 [♦] [_] [2nd] [a-lock] MS [alpha] 2 [ENTER]	
4. Using the same m, calculate the force for an acceleration due to gravity (the constant _g). For *_g*, you can use the pre-defined constant available from the **UNITS** dialog box or you can type *_g*. 5 [♦] [_] [2nd] [a-lock] KG [alpha] [×] [2nd] [UNITS] [▶] [alpha] G [ENTER] [ENTER]	
5. Convert to kilograms of force (_kgf). [2nd] [▶] displays the ▶ conversion operator. [▶] [2nd] [▶] [♦] [_] [2nd] [a-lock] KGF [alpha] [ENTER]	

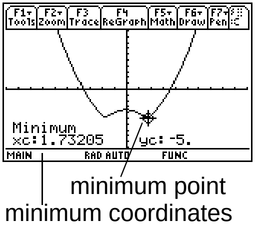
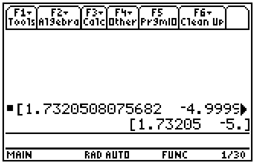
Basic Function Graphing I

The example in this section demonstrates some of the graphing capabilities of the TI-89 Titanium keystrokes. It illustrates how to graph a function using the **Y= Editor.** You will learn how to enter a function, produce a graph of the function, trace a curve, find a minimum point, and transfer the minimum coordinates to the Home screen.

Explore the graphing capabilities of the TI-89 Titanium by graphing the function $y=(|x^2-3|-10)/2$.

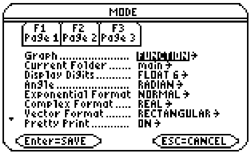
Steps and keystrokes	Display
1. Display the **Y= Editor.** Press [♦] [Y=]	
2. Enter the function **(abs(x^2−3)−10)/2**. The screen shot shows the “pretty print” display at **y1=**. [(] [CATALOG] A [ENTER] X [^] 2 [−] 3 [)] [−] 1 0 [)] [÷] 2 [ENTER]	
3. Display the graph of the function. Select **6:ZoomStd** by pressing **6** or by moving the cursor to **6:ZoomStd** and pressing [ENTER]. Press [F2] 6	

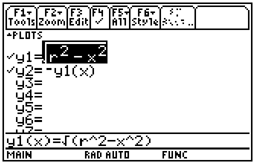
Steps and keystrokes	Display
4. Turn on **Trace.** The tracing cursor, and the x and y coordinates are displayed. Press [F3]	
5. Open the **MATH** menu and select **3:Minimum.** Press [F5] ↓ ↓ [ENTER]	
6. Set the lower bound. Press → (right cursor) to move the tracing cursor until the lower bound for x is just to the left of the minimum node before pressing [ENTER] the second time. Press → ... → [ENTER]	
7. Set the upper bound. Press → (right cursor) to move the tracing cursor until the upper bound for x is just to the right of the minimum node. Press → ... →	

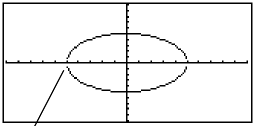
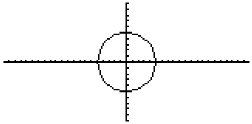
Steps and keystrokes	Display
8. Find the minimum point on the graph between the lower and upper bounds. Press [ENTER]	
9. Transfer the result to the Home screen, and then display the Home screen. [♦] [(-)] [HOME]	

Basic Function Graphing II

Graph a circle of radius 5, centered on the origin of the coordinate system. View the circle using the standard viewing window (**ZoomStd**). Then use **ZoomSqr** to adjust the viewing window.

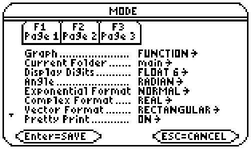
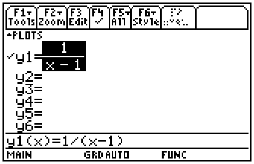
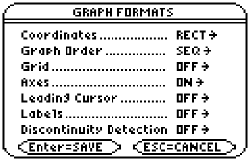
Steps and keystrokes	Display
1. Display the **MODE** dialog box. For **Graph** mode, select **FUNCTION.** Press [MODE] ⓘ 1 [ENTER]	

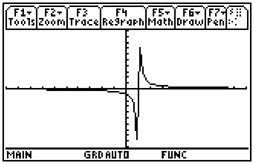
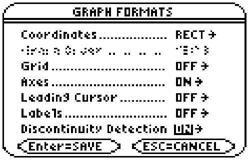
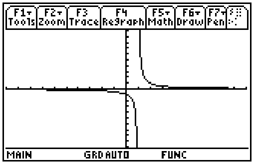
Steps and keystrokes	Display
2. Display the Home screen. Then store the radius, 5, in variable r. [HOME] 5 [STO▸] [alpha] R [ENTER]	5→r
3. Display and clear the **Y= Editor.** Then define y1(x) = $\sqrt{(r^2 - x^2)}$, the top half of a circle. In function graphing, you must define separate functions for the top and bottom halves of a circle. [♦] [Y=] [F1] 8 [ENTER] [ENTER] [2nd] [√] [alpha] R [^] 2 [−] X [^] 2 [)] [ENTER]	
4. Define y2(x) = $-\sqrt{r^2 - x^2}$, the function for the bottom half of the circle. The bottom half is the negative of the top half, so you can define y2(x) = ⁻y1(x). Use the full function name **y1(x)**, not simply y1. Press [ENTER] [(-)] Y 1 [(] X [)] [ENTER]	

Steps and keystrokes	Display
5. Select the **ZoomStd** viewing window, which automatically graphs the functions. In the standard viewing window, both the x and y axes range from ⁻10 to 10. However, this range is spread over a longer distance along the x axis than the y axis. Therefore, the circle appears as an ellipse. Press [F2] 6	 Notice slight gap between top and bottom halves.
6. Select **ZoomSqr**. **ZoomSqr** increases the range along the x axis so that circles and squares are shown in correct proportion. Press [F2] 5	

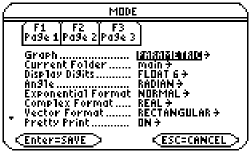
Note: There is a gap between the top and bottom halves of the circle because each half is a separate function. The mathematical endpoints of each half are (⁻5,0) and (5,0). Depending on the viewing window, however, the *plotted* endpoints for each half may be slightly different from their *mathematical* endpoints.

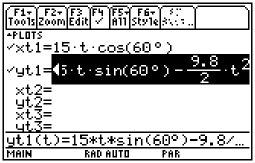
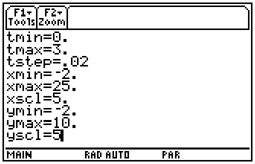
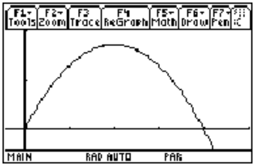
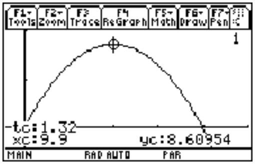
Use the “Detect Discontinuities” graph format to eliminate faux asymptotes and connections in a jump discontinuity.

Steps and keystrokes	Display
1. Display the **MODE** dialog box. For **Graph** mode, select **FUNCTION.** For **Angle** mode, select **RADIAN.** Press [MODE] ▶ 1 ▼ ▼ ▼ ▶ 1 [ENTER]	
2. Open the Y= Editor and enter y1(x)=1/(x-1). Press [♦] [Y=] 1 [÷] [(] X [−] 1 [)] [ENTER]	
3. Display the Graph Formats dialog box and set “Detect Discontinuities” to OFF **Note:** The second item on the Graph Format dialog is not greyed out, which means it can be set to sequential “Seq” or simultaneous “Simul”. [♦] [I] ▼ ▼ ▼ ▼ ▼ ▼ ▶ 1 [ENTER]	

Steps and keystrokes	Display
4. Execute the **Graph** command, which automatically displays the Graph screen. Observe the “faux” asymptotes contained in the graph. Press [◆] [GRAPH]	
5. Display the Graph Formats dialog box and set “Detect Discontinuities” to ON. **Note:** The second item on the Graph Format dialog is greyed out, which means the graph order is set to sequential “Seq”. [◆] [I] ▼ ▼ ▼ ▼ ▼ ▼ ▶ 2 [ENTER]	
6. Execute the **Graph** command, which automatically displays the Graph screen. No”faux” asymptotes are present on the graph. **Note:** Graphing speed may slow considerably when “Detect Discontinuities” is set to ON. Press [◆] [GRAPH]	

Graph the parametric equations describing the path of a ball kicked at an angle (θ) of 60° with an initial velocity (v_0) of 15 meters/sec. The gravity constant g = 9.8 meters/sec^2. Ignoring air resistance and other drag forces, what is the maximum height of the ball and when does it hit the ground?

Steps and keystrokes	Display
1. Display the **MODE** dialog box. For **Graph** mode, select **PARAMETRIC.** Press [MODE] [▶] 2 [ENTER]	
2. Display and clear the **Y= Editor.** Then define the horizontal component xt1(t) = v_0t cos θ. Enter values for v_0 and θ. [♦] [Y=] [F1] 8 [ENTER] [ENTER] 15T [×] [2nd] [COS] 60 [2nd] [°] [)] [ENTER] Type T [×] [2nd] [COS], not T [2nd] [COS]. Enter a ° symbol by typing either [2nd] [°] or [2nd] [MATH] 2 1. This ensures a number is interpreted as degrees, regardless of the angle mode.	xt1(t)=15t∗cos(60°)

Steps and keystrokes	Display
3. Define the vertical component yt1(t) = $v_0 t \sin \theta - (g/2)t^2$. Enter values for v_0, θ, and g. [ENTER] 15T [×] [2nd] [SIN] 60 [2nd] [°] [)] [−] [(] 9.8 [÷] 2 [)] T [^] 2 [ENTER]	
4. Display the **Window Editor.** Enter Window variables appropriate for this example. You can press either ▼ or [ENTER] to enter a value and move to the next variable. Press [♦] [WINDOW] 0 ▼ 3 ▼ .02 ▼ [(-)] 2 ▼ 25 ▼ 5 ▼ [(-)] 2 ▼ 10 ▼ 5	
5. Graph the parametric equations to model the path of the ball. Press [♦] [GRAPH]	
6. Select **Trace**. Then move the cursor along the path to find the: • y value at maximum height. • t value where the ball hits the ground. Press [F3] ► or ◄ as necessary	

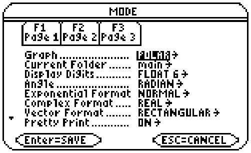
Polar Graphing

The graph of the polar equation r1(θ) = A sin Bθ forms the shape of a rose. Graph the rose for A=8 and B=2.5. Then explore the appearance of the rose for other values of A and B.

Steps and keystrokes | **Display**

1. Display the **MODE** dialog box. For **Graph** mode, select **POLAR**. For **Angle** mode, select **RADIAN**.

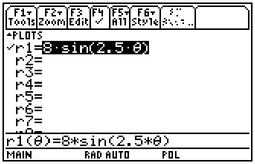
   Press [MODE] ▶ 3 ▼ ▼ ▼ ▶ 1 [ENTER]



2. Display and clear the **Y= Editor.** Then define the polar equation r1(θ) = A sin Bθ.

   Enter 8 and 2.5 for A and B, respectively.

   [♦] [Y=] [F1] 8 [ENTER] [ENTER] 8 [2nd] [SIN] 2.5 [♦] [θ] [)] [ENTER]

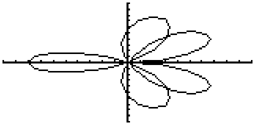


Steps and keystrokes	**Display**

3. Select the **ZoomStd** viewing window, which graphs the equation.

   - The graph shows only five rose petals.
     - In the standard viewing window, the Window variable θmax = 2π. The remaining petals have θ values greater than 2π.
   - The rose does not appear symmetrical.
     - Both the x an y axes range from ⁻10 to 10. However, this range is spread over a longer distance along the x axis than the y axis.

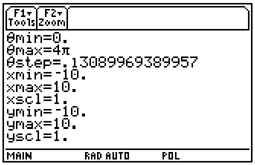
   Press [F2] 6

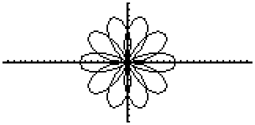


4. Display the **Window Editor**, and change θmax to 4π.

   4π will be evaluated to a number when you leave the **Window Editor**.

   Press [♦] [WINDOW] ⊖ 4 [2nd] [π]



Steps and keystrokes	Display
5. Select **ZoomSqr**, which regraphs the equation. **ZoomSqr** increases the range along the x axis so that the graph is shown in correct proportion. Press [F2] 5	
You can change values for *A* and *B* as necessary and regraph the equation.	

Sequence Graphing

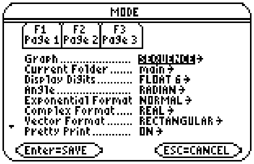
A small forest contains 4000 trees. Each year, 20% of the trees will be harvested (with 80% remaining) and 1000 new trees will be planted. Using a sequence, calculate the number of trees in the forest at the end of each year. Does it stabilize at a certain number?

Initially	After 1 Year	After 2 Years	After 3 Years	. . .
4000	.8 x 4000 + 1000	.8 x (.8 x 4000 + 1000) + 1000	.8 x (.8 x (.8 x 4000 + 1000) + 1000) + 1000	. . .

Steps and keystrokes	Display

1. Display the **MODE** dialog box. For **Graph** mode, select **SEQUENCE**.

   Press [MODE] ▶ 4 [ENTER]

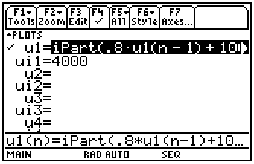


2. Display and clear the **Y= Editor.** Then define the sequence as u1(n) = iPart(.8∗u1(n−1)+1000).

   Use **iPart** to take the integer part of the result. No fractional trees are harvested.

   To access **iPart(**, you can use [2nd] [MATH], simply type it, or select it from the **CATALOG.**

   [♦] [Y=] [F1] 8 [ENTER] [ENTER] [2nd] [MATH] 1 4 . 8 [alpha] U1 [(] [alpha] N [−] 1 [)] [+] 1000 [)] [ENTER]



3. Define ui1 as the initial value that will be used as the first term.

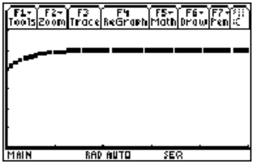
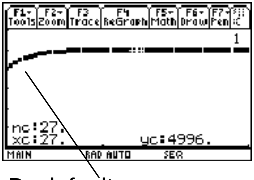
   Press [ENTER] 4000 [ENTER]

4. Display the **Window Editor.** Set the n and plot Window variables.

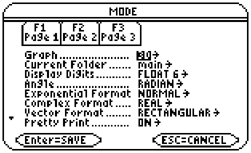
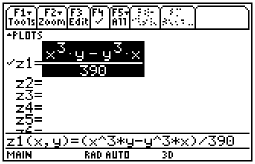
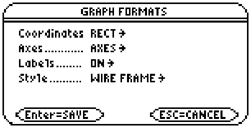
   nmin=0 and **nmax=50** evaluate the size of the forest over 50 years.

   Press [♦] [WINDOW] 0 ⊙ 50 ⊙ 1 ⊙ 1 ⊙

```
nmin=0.
nmax=50.
plotStrt=1.
plotStep=1.
xmin=0.
xmax=50.
xscl=10.
ymin=0.
ymax=6000.
yscl=1000.
```

Steps and keystrokes	Display
5. Set the x and y Window variables to appropriate values for this example. Press 0 ⊖ 50 ⊖ 10 ⊖ 0 ⊖ 6000 ⊖ 1000	
6. Display the Graph screen. Press [♦] [GRAPH]	
7. Select **Trace**. Move the cursor to trace year by year. How many years (nc) does it take the number of trees (yc) to stabilize? Trace begins at nc=0. nc is the number of years. xc = nc since n is plotted on the x axis. yc = u1(n), the number of trees at year n. Press [F3] ⊳ and ⊲ as necessary	 By default, sequences use the Square display style.

Graph the 3D equation z(x,y) = $(x^3y - y^3x) / 390$. Animate the graph by using the cursor to interactively change the eye Window variable values that control your viewing angle. Then view the graph in different graph format styles.

Steps and keystrokes	Display
1. Display the **MODE** dialog box. For **Graph** mode, select **3D**. Press [MODE] ▶ 5 [ENTER]	
2. Display and clear the **Y= Editor.** Then define the 3D equation z1(x,y) = $(x^3y - y^3x) / 390$. Notice that implied multiplication is used in the keystrokes. Press [◆] [Y=] [F1] 8 [ENTER] [ENTER] [(] X [^] 3 Y [−] Y [^] 3 X [)] [÷] 390 [ENTER]	
3. Change the graph format to display and label the axes. Also set **Style = WIRE FRAME.** You can animate any graph format style, but **WIRE FRAME** is fastest. [◆] [I] ▼ ▶ 2 ▼ ▶ 2 ▼ ▶ 1 [ENTER]	

Steps and keystrokes	Display

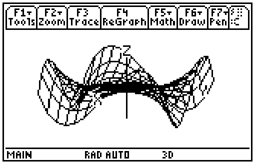
4. Select the **ZoomStd** viewing cube, which automatically graphs the equation.

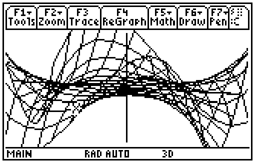
   As the equation is evaluated (before it is graphed), “evaluation percentages” are shown in the upper-left part of the screen.

   Press [F2] 6

   Note: If you have already used 3D graphing, the graph may be shown in expanded view. When you animate the graph, the screen returns to normal view automatically. (Except for animation, you can do the same things in normal and expanded view.)

   Press [×] (press [×] to switch between expanded and normal view)



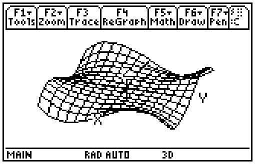


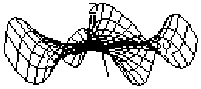
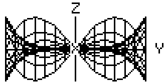
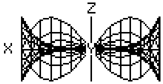
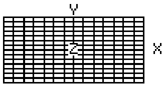
5. Animate the graph by decreasing the eyeϕ Window variable value.

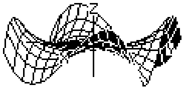
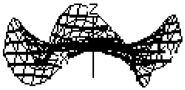
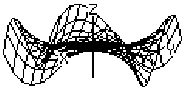
   ⊖ or ⊕ may affect eyeθ and eyeψ, but to a lesser extent than eyeϕ**.**

   To animate the graph continuously, press and hold the cursor for about 1 second and then release it. To stop, press [ENTER].

   Press ⊖ eight times



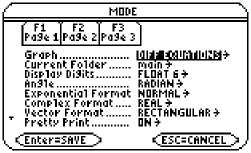
Steps and keystrokes	Display
6. Return the graph to its initial orientation. Then move the viewing angle along the “viewing orbit” around the graph. Press 0 (zero, not the letter O) ◀ ◀ ◀	
7. View the graph along the x axis, the y axis, and then the z axis. Press X This graph has the same shape along the y axis and x axis. Press Y Press Z	  
8. Return to the initial orientation. Press 0 (zero)	

Steps and keystrokes	Display
9. Display the graph in different graph format styles. [I] (press [I] to switch from each style to the next)	 HIDDEN SURFACE  CONTOUR LEVELS (may require extra time to calculate contours)  WIRE AND CONTOUR  WIRE FRAME

Note: You can also display the graph as an implicit plot by using the **GRAPH FORMATS** dialog box ([♦] [I]). If you press [I] to switch between styles, the implicit plot is not displayed.

Differential Equation Graphing

Graph the solution to the logistic 1st-order differential equation y' = .001y∗(100−y). Start by drawing only the slope field. Then enter initial conditions in the **Y= Editor** and interactively from the Graph screen.

Steps and keystrokes	**Display**
1. Display the **MODE** dialog box. For **Graph** mode, select **DIFF EQUATIONS.** Press [MODE] ► 6 [ENTER]	

Steps and keystrokes	Display

2. Display and clear the **Y= Editor.** Then define the 1st-order differential equation:

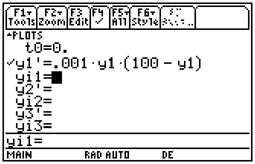
   y1'(t)=.001y1∗(100−y1)

   Press [×] to enter the ∗ shown above. Do not use implied multiplication between the variable and parentheses. If you do, it is treated as a function call.

   Leave the initial condition **yi1** blank.

   Note: With y1' selected, the device will graph the **y1** solution curve, not the derivative y1'.

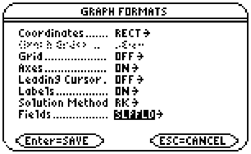
   Press [◆] [Y=] [F1] 8 [ENTER] [ENTER] .001 Y1 [×] [(] 100 [−] Y1 [)] [ENTER]

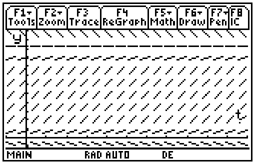
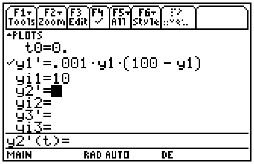
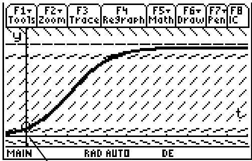


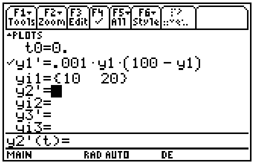
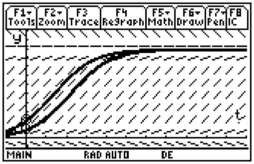
3. Display the **GRAPH FORMATS** dialog box. Then set **Axes = ON**, **Labels = ON**, **Solution Method = RK**, and **Fields = SLPFLD.**

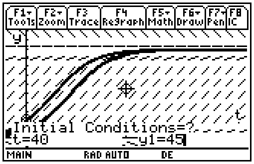
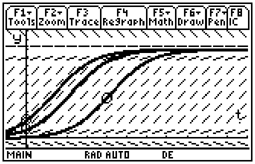
   Note: To graph one differential equation, *Fields* must be set to **SLPFLD** or **FLDOFF.** If **Fields=DIRFLD**, an error occurs when you graph.

   [◆] [I] [▼] [▼] [▶] 2 [▼] [▼] [▶] 2 [▼] [▶] 1 [▼] [▶] 1 [ENTER]

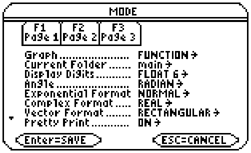
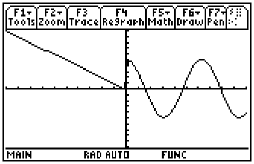


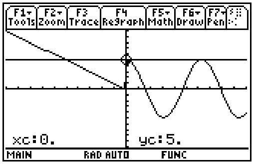
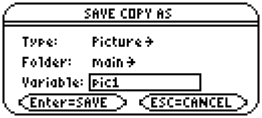
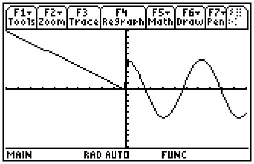
Steps and keystrokes	Display
4. Display the **Window Editor**, and set the Window variables as shown to the right. Press [◆] [WINDOW] 0 ▼ 10 ▼ .1 ▼ 0 ▼ [(-)] 10 ▼ 110 ▼ 10 ▼ [(-)] 10 ▼ 120 ▼ 10 ▼ 0 ▼ .001 ▼ 20	t0=0. tmax=10. tstep=.1 tplot=0. xmin=-10. xmax=110. xscl=10. ymin=-10. ymax=120. yscl=10. ncurves=0. diftol=.001 fldres=20.
5. Display the Graph screen. Because you did not specify an initial condition, only the slope field is drawn (as specified by **Fields=SLPFLD** in the **GRAPH FORMATS** dialog box). Press [◆] [GRAPH]	
6. Return to the **Y= Editor** and enter an initial condition: **yi1=10** Press [◆] [Y=] [ENTER] 10 [ENTER]	
7. Return to the Graph screen. Initial conditions entered in the **Y= Editor** always occur at t_0. The graph begins at the initial condition and plots to the right. Then it plots to the left. Press [◆] [GRAPH]	 The initial condition is marked with a circle.

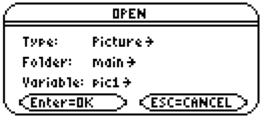
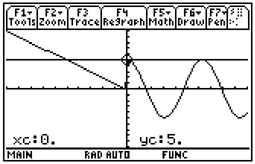
Steps and keystrokes	Display
8. Return to the **Y= Editor** and change **yi1** to enter two initial conditions as a list: **yi1={10,20}** Press [♦] [Y=] ⏶ [ENTER] [2nd] [{] 10 [,] 20 [2nd] [}] [ENTER]	
9. Return to the Graph screen. Press [♦] [GRAPH]	

Steps and keystrokes	Display
10. To select an initial condition interactively, press: [2nd] [F8] When prompted, enter t=40 and y1=45. When selecting an initial condition interactively, you can specify a value for t other than the t_0 value entered in the **Y= Editor** or **Window Editor.** Instead of entering **t** and **y1** after pressing [2nd] [F8] you can move the cursor to a point on the screen and then press [ENTER]. You can use [F3] to trace curves for initial conditions specified in the **Y= Editor.** However, you cannot trace the curve for an initial condition selected interactively. [2nd] [F8] 40 [ENTER] 45 [ENTER]	 

From the Home screen, graph the piecewise defined function: y = ⁻x when x < 0 and y = 5 cos(x) when x ≥ 0. Draw a horizontal line across the top of the cosine curve. Then save a picture of the displayed graph.

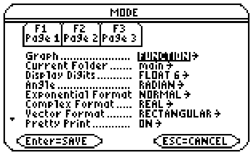
Steps and keystrokes	Display
1. Display the **MODE** dialog box. For **Graph** mode, select **FUNCTION**. For **Angle** mode, select **RADIAN**. Press [MODE] ▶ 1 ▼ ▼ ▼ ▶ 1 [ENTER]	
2. Display the Home screen. Use the **Graph** command and the **when** function to specify the piecewise defined function. [F4] **2** selects **Graph** from the **Other** toolbar menu and automatically adds a space. [HOME] [F4] 2 [2nd] [a-lock] WHEN [alpha] [(] X [2nd] [<] 0 [,] [(-)] X [,] 5 [×] [2nd] [COS] X [)] [)]	Graph when(x<0,⁻x, 5∗cos(x))
3. Execute the **Graph** command, which automatically displays the Graph screen. The graph uses the current Window variables, which are assumed to be their standard values ([F2] **6**) for this example. Press [ENTER]	

Steps and keystrokes	Display
4. Draw a horizontal line across the top of the cosine curve. The calculator remains in “horizontal” mode until you select a different operation or press [ESC]. [2nd] [F7] 5 ↑ (until the line is positioned) [ENTER]	
5. Save a picture of the graph. Use **PIC1** as the variable name for the picture. Be sure to set **Type = Picture**. By default, it is set to **GDB.** [F1] 2 → 2 ↓ ↓ PIC [alpha] 1 [ENTER] [ENTER]	
6. Clear the drawn horizontal line. You can also press [F4] to regraph. [2nd] [F6] 1	

Steps and keystrokes	Display
7. Open the saved picture variable to redisplay the graph with the line. Be sure to set **Type = Picture**. By default, it is set to **GDB.** Press [F1] 1 ► 2 (if not already shown, also set Variable = pic1) [ENTER]	 

Tables

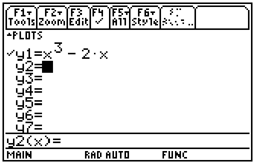
Evaluate the function $y=x^3-2x$ at each integer between -10 and 10. How many sign changes are there, and where do they occur?

Steps and keystrokes	Display
1. Display the **MODE** dialog box. For the **Graph** mode, select **FUNCTION.** Press [MODE] ► 1 [ENTER]	

Steps and keystrokes	Display

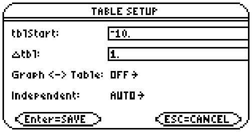
2. Display and clear the **Y= Editor.** Then define y1(x) = x^3 − 2x.

   Press [♦] [Y=] [F1] 8 [ENTER] [ENTER] X [^] 3 [−] 2 X [ENTER]



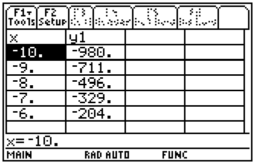
3. Set the table parameters to:
   tblStart = ⁻10
   ∆tbl = 1
   Graph < - > Table = OFF
   Independent = AUTO

   Press [♦] [TBLSET] [(-)] 10 ⊖ 1 ⊖ ⊕ 1 ⊖ ⊕ 1 [ENTER]



4. Display the Table screen.

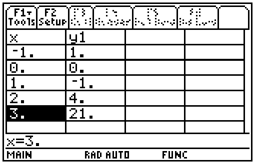
   Press [♦] [TABLE]

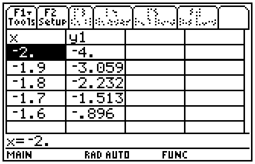


5. Scroll through the table. Notice that **y1** changes sign at x = ⁻1, 1, and 2.

   To scroll one page at a time, use [2nd] ⊖ and [2nd] ⊕.

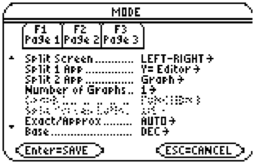
   Press ⊖ and ⊕ as necessary

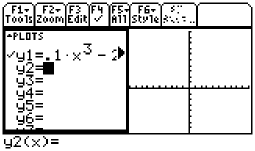
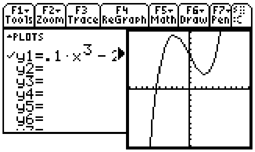


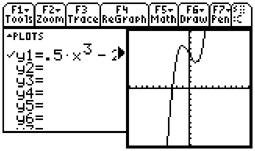
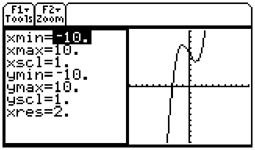
Steps and keystrokes		Display
6.	Zoom in on the sign change between x = $^{-}2$ and x = $^{-}1$ by changing the table parameters to: **tblStart = $^{-}$2** **Δtbl = .1** Press [F2] [(-)] 2 ▼ .1 [ENTER] [ENTER]	

Split Screens

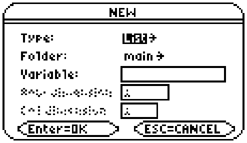
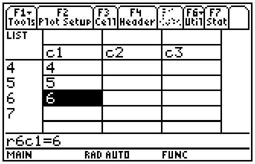
Split the screen to show the **Y= Editor** and the Graph screen. Explore the behavior of a polynomial as its coefficients change.

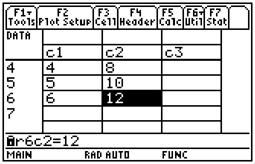
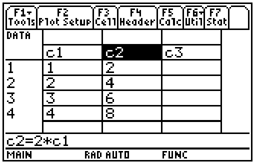
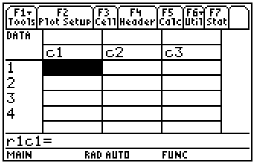
Steps and keystrokes		Display
1.	Display the **MODE** dialog box. For **Graph**, select **FUNCTION**. For **Split Screen**, select **LEFT-RIGHT.** For **Split 1 App**, select **Y= Editor.** For **Split 2 App**, select **Graph**. Press [MODE] ► 1 [F2] ► 3 ▼ ► 2 ▼ ► 4 [ENTER]	

Steps and keystrokes	Display
2. Clear the **Y= Editor** and turn off any stat data plots. Define y1(x) = .1x^3−2x+6. A thick border around the **Y= Editor** indicates it is active. When active, its entry line goes all the way across the display. Press [F1] 8 [ENTER] [F5] 5 [ENTER] .1 X [^] 3 [−] 2 X [+] 6 [ENTER]	
3. Select the **ZoomStd** viewing window, which switches to the Graph screen and graphs the function. The thick border is now around the Graph screen. Press [F2] 6	
4. Switch to the **Y= Editor** and edit **y1(x)** to change .1x^3 to .5x^3. [2nd] [↔] is the second function of [APPS]. The thick border is around the **Y= Editor.** Press [2nd] [↔] ↑ [ENTER] ← → → [←] 5 [ENTER]	

Steps and keystrokes	Display
5. Switch to the Graph screen, which regraphs the edited function. The thick border is around the Graph screen. Press [2nd] [⊟]	
6. Switch to the **Y= Editor** and open the **Window Editor** in its place. Press [2nd] [⊟] [♦] [WINDOW]	
7. Open the Home screen and then exit to a full-sized Home screen. Press: [2nd] [QUIT] [HOME]	

Use the **Data/Matrix Editor** to create a one-column list variable. Then add a second column of information. Notice that the list variable (which can have only one column) is automatically converted into a data variable (which can have multiple columns).

Steps and keystrokes	Display
1. Use [APPS] to display the **Data/Matrix Editor.** Create a new list variable named **TEMP**. Press 3 ⓡ 3 ⓓ ⓓ TEMP [ENTER] [ENTER]	
2. Enter a column of numbers. Then move the cursor up one cell (just to see that a highlighted cell's value is shown on the entry line). **LIST** is shown in the upper-left corner to indicate a list variable. You can use ⓓ instead of [ENTER] to enter information in a cell. Press 1 [ENTER] 2 [ENTER] 3 [ENTER] 4 [ENTER] 5 [ENTER] 6 [ENTER] ⓤ	

Steps and keystrokes	Display
3. Move to column 2, and define its column header so that it is twice the value of column 1. **DATA** is shown in the upper-left corner to indicate that the list variable was converted to a data variable. ▶ [F4] 2 [×] [alpha] C 1 [ENTER]	 🔒 means the cell is in a defined column.
4. Move to the column 2 header cell to show its definition in the entry line. When the cursor is on the header cell, you do not need to press [F4] to define it. Simply begin typing the expression. Press [2nd] ▲ ▲	
5. Clear the contents of the variable. Simply clearing the data does not convert the data variable back into a list variable. Press [F1] 8 [ENTER]	

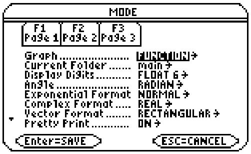
Note: If you don’t need to save the current variable, use it as a *scratchpad*. The next time you need a variable for temporary data, clear the current variable and re-use it. This lets you enter temporary data without wasting memory by creating a new variable each time.

Based on a sample of seven cities, enter data that relates population to the number of buildings with more than 12 stories. Using Median-Median and linear regression calculations, find and plot equations to fit the data. For each regression equation, predict how many buildings of more than 12 stories you would expect in a city of 300,000 people.

Steps and keystrokes | **Display**

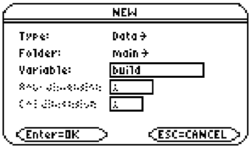
1. Display the **MODE** dialog box. For **Graph** mode, select **FUNCTION.**

   Press [MODE] ▶ 1 [ENTER]



2. Use [APPS] to display the **Data/Matrix Editor.** Create a new data variable named **BUILD**.

   Press 3 ▼ ▼ BUILD [ENTER] [ENTER]

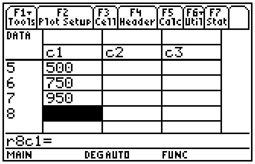


Steps and keystrokes	Display

3. Using the sample data below, enter the population in column 1.

Pop. (in 1000s)	Bldgs > 12 stories
150	4
500	31
800	42
250	9
500	20
750	55
950	73

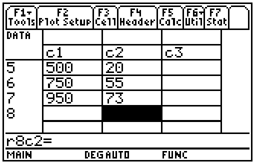
Press 150 [ENTER] 500 [ENTER] 800 [ENTER] 250 [ENTER] 500 [ENTER] 750 [ENTER] 950 [ENTER]



4. Move the cursor to row 1 in column 2 (r1c2). Then enter the corresponding number of buildings.

[♦] ⊕ moves the cursor to the top of the page. After typing data for a cell, you can press [ENTER] or ⊖ to enter the data and move the cursor down one cell. Pressing ⊕ enters the data and moves the cursor up one cell.

⊙ [♦] ⊕ 4 [ENTER] 31 [ENTER] 42 [ENTER] 9 [ENTER] 20 [ENTER] 55 [ENTER] 73 [ENTER]



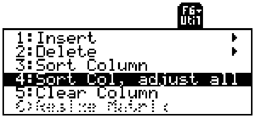
Steps and keystrokes	Display

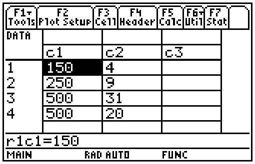
5. Move the cursor to row 1 in column 1 (r1c1). Sort the data in ascending order of population.

   This sorts column 1 and then adjusts all other columns so that they retain the same order as column 1. This is critical for maintaining the relationships between columns of data.

   To sort column 1, the cursor can be anywhere in column 1. This example has you press
   [calculator icon] [◆] [↑]
   so that you can see the first four rows.

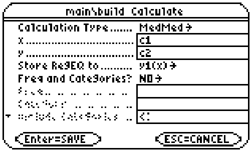
   [←] [◆] [↑] [2nd] [F6] 4





6. Display the **Calculate** dialog box. Set
   Calculation Type = MedMed
   x = C1
   y = C2
   Store RegEQ to = y1(x)

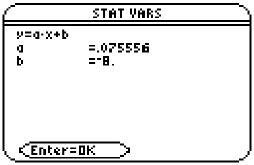
   [F5] [→] 7 [↓] C [alpha] 1 [↓] [alpha] C2 [↓] [→] [↓] [ENTER]



7. Perform the calculation to display the MedMed regression equation.

   As specified on the **Calculate** dialog box, this equation is stored in **y1(x)**.

   Press [ENTER]



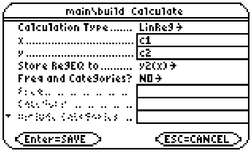
Steps and keystrokes	Display

8. Close the **STAT VARS** screen. The **Data/Matrix Editor** displays.

   Press [ENTER]

9. Display the **Calculate** dialog box. Set:
   Calculation Type = LinReg
   x = C1
   y = C2
   Store RegEQ to = y2(x)

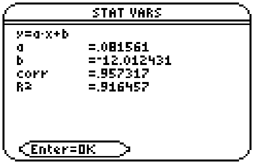
   Press [F5] ▶ 5 ▼ ▼ ▼ ▶ ▼ [ENTER]



10. Perform the calculation to display the LinReg regression equation.

    This equation is stored in **y2(x).**

    Press [ENTER]



11. Close the **STAT VARS** screen. The **Data/Matrix Editor** displays.

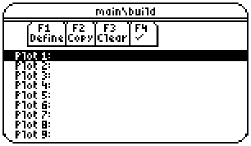
    Press [ENTER]

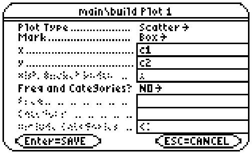
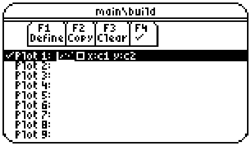
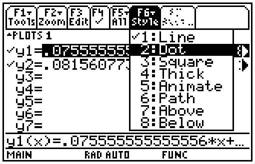
12. Display the Plot Setup screen.

    Plot 1 is highlighted by default.

    [F3] lets you clear highlighted Plot settings.

    Press [F2]



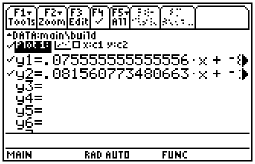
Steps and keystrokes	**Display**
13. Define **Plot 1** as: **Plot Type = Scatter** **Mark = Box** **x = C1** **y = C2** Notice the similarities between this and the **Calculate** dialog box. [F1] ▶ 1 ▼ ▶ 1 ▼ C [alpha] 1 ▼ [alpha] C2	
14. Save the plot definition and return to the Plot Setup screen. Notice the shorthand notation for **Plot 1's** definition. Press [ENTER] twice	
15. Display the **Y= Editor.** For **y1(x)**, the MedMed regression equation, set the display style to **Dot.** **Note:** Depending on the previous contents of your **Y= Editor**, you may need to move the cursor to **y1**. **PLOTS 1** at the top of the screen means that **Plot 1** is selected. Notice that **y1(x)** and **y2(x)** were selected when the regression equations were stored. ♦ [Y=] [2nd] [F6] 2	

Steps and keystrokes | **Display**

16. Scroll up to highlight **Plot 1.**

    The displayed shorthand definition is the same as on the Plot Setup screen.

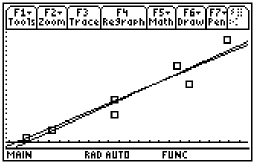
    Press ▲



17. Use **ZoomData** to graph **Plot 1** and the regression equations **y1(x)** and **y2(x).**

    ZoomData examines the data for all selected stat plots and adjusts the viewing window to include all points.

    Press [F2] 9



18. Return to the current session of the **Data/Matrix Editor.**

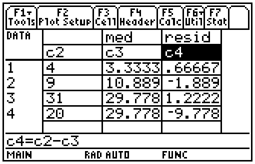
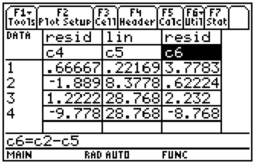
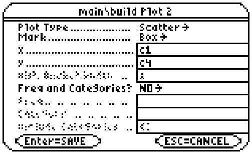
    Press [APPS] D [ENTER] [ENTER]

19. Enter a title for column 3. Define column 3’s header as the values predicted by the MedMed line.

    To enter a title, the cursor must highlight the title cell at the very top of the column.

    [F4] lets you define a header from anywhere in a column. When the cursor is on a header cell, pressing [F4] is not required.

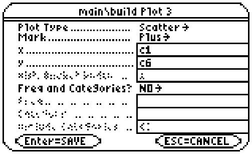
    ▶ ▶ ▲ ▲ [2nd] [a-lock] MED [alpha] [ENTER] [F4] Y1 [(] [alpha] C1 [)] [ENTER]

Steps and keystrokes	Display
20. Enter a title for column 4. Define column 4's header as the residuals (difference between observed and predicted values) for MedMed. ▶ ▲ [2nd] [a-lock] RESID [alpha] [ENTER] [alpha] C2 [−] [alpha] C3 [ENTER]	
21. Enter a title for column 5. Define column 5's header as the values predicted by the LinReg line. ▶ ▲ ▲ [2nd] [a-lock] LIN [alpha] [ENTER] [F4] Y2 [(] [alpha] C1 [)] [ENTER]	
22. Enter a title for column 6. Define column 6's header as the residuals for LinReg. ▶ ▲ [2nd] [a-lock] RESID [alpha] [ENTER] [F4] [alpha] C2 [−] [alpha] C5 [ENTER]	
23. Display the Plot Setup screen and deselect **Plot 1**. Press [F2] [F4]	
24. Highlight **Plot 2** and define it as: **Plot Type = Scatter** **Mark = Box** **x = C1** **y = C4** (MedMed residuals) ▼ [F1] ▼ ▼ C [alpha] 1 ▼ [alpha] C4 [ENTER] [ENTER]	

Steps and keystrokes | **Display**

25. Highlight **Plot 3** and define it as:
Plot Type = Scatter
Mark = Plus
x = C1
y = C6 (LinReg residuals)

↓ [F1] ↓ → 3 ↓ C [alpha] 1 ↓ [alpha] C6
[ENTER] [ENTER]

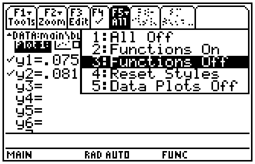


26. Display the **Y= Editor** and turn all the **y(x)** functions off.

From [F5], select **3:Functions Off**, not **1:All Off.**

Plots 2 and 3 are still selected.

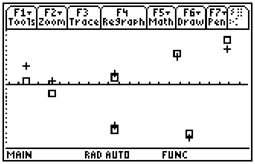
Press [♦] [Y=] [F5] 3



27. Use **ZoomData** to graph the residuals.

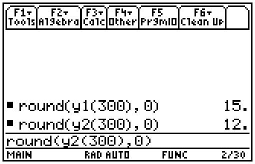
☐ marks the MedMed residuals;
+ marks the LinReg residuals.

Press [F2] 9



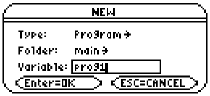
28. Display the Home screen.

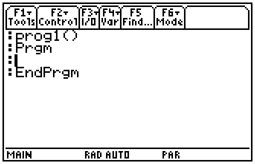
[HOME]

Steps and keystrokes	Display
29. Use the MedMed (**y1(x)**) and LinReg (**y2(x)**) regression equations to calculate values for x = 300 (300,000 population). The **round** function *(*[2nd] [MATH] 1 3*)* ensures that results show an integer number of buildings. After calculating the first result, edit the entry line to change **y1** to **y2.** Press [2nd] [MATH] 1 3 Y1 [(] 300 [)] [,] 0 [)] [ENTER] ⓡ ⓛ (eight times) [←] 2 [ENTER]	

Programming

Write a program that prompts the user to enter an integer, sums all integers from 1 to the entered integer, and displays the result.

Steps and keystrokes	Display
1. Use [APPS] to display the **Program Editor.** Create a new program. Press 3	
2. Type **PROG1** (with no spaces) as the name of the new program variable. ⓓ ⓓ PROG [alpha] 1	

Steps and keystrokes	Display
3. Display the “template” for a new program. The program name, **Prgm**, and **EndPrgm** are shown automatically. After typing in an input box such as Variable, you must press [ENTER] twice. Press [ENTER] twice	

Steps and keystrokes	Display

4. Type the following program lines.

`Request "Enter an integer",n`
Displays a dialog box that prompts "Enter an integer", waits for the user to enter a value, and stores it (as a string) to variable n.

`expr(n)→n`
Converts the string to a numeric expression.

`0→temp`
Creates a variable named temp and initializes it to **0**.

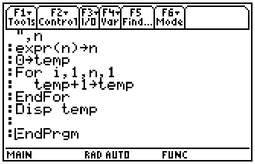
`For i,1,n,1`
Starts a *For loop* based on variable i. First time through the loop, i = 1. At end of loop, i is incremented by 1. Loop continues until i > n.

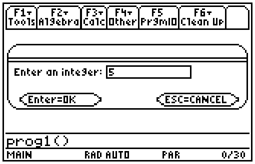
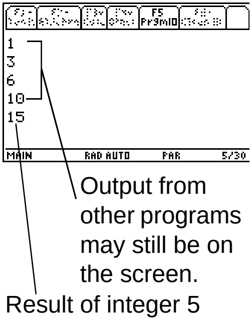
`temp+i→temp`
Adds current value of i to temp.

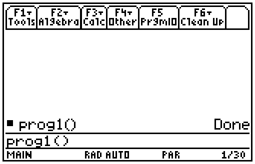
`EndFor`
Marks the end of the *For loop*.

`Disp temp`
Displays the final value of temp.

Type the program lines as shown. Press [ENTER] at the end of each line.

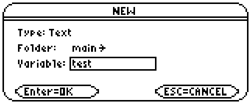


Steps and keystrokes	Display
5. Go to the Home screen. Enter the program name, followed by a set of parentheses. You must include ( ) even when there are no arguments for the program. The program displays a dialog box with the prompt specified in the program. [HOME] [2nd] [a-lock] PROG [alpha] 1 [(] [)] [ENTER]	prog1()
6. Type 5 in the displayed dialog box. Press 5	
7. Continue with the program. The **Disp** command displays the result on the Program I/O screen. The result is the sum of the integers from 1 through 5. Although the Program I/O screen looks similar to the Home screen, it is for program input and output only. You cannot perform calculations on the Program I/O screen. Press [ENTER] twice	

Steps and keystrokes	Display
8. Leave the Program I/O screen and return to the Home screen. You can also press [ESC], [2nd] [QUIT], or [HOME] to return to the Home screen. Press [F5]	

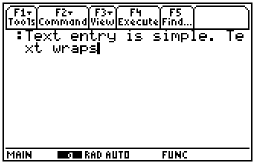
Text Operations

Start a new **Text Editor** session. Then practice using the **Text Editor** by typing whatever text you want. As you type, practice moving the text cursor and correcting any typos you may enter.

Steps and keystrokes	Display
1. Start a new session of the **Text Editor.** Press 3	
2. Create a text variable called **TEST**, which will automatically store any text you enter in the new session. Use the **MAIN** folder, shown as the default on the **NEW** dialog box. After typing in an input box such as **Variable**, you must press [ENTER] twice. Press ⊖ TEST [ENTER] [ENTER]	

Steps and keystrokes | **Display**

3. Type some sample text.



   - To type a single uppercase letter, press [↑] and then the letter.
     - To type a space, press [alpha] [␣] (alpha function of the [(-)] key).
     - To type a period, press [alpha] to turn alpha-lock off, press [.], and then press [2nd] [a-lock] to turn alpha-lock on again.

   Practice editing your text by using:

   - The cursor pad to move the text cursor.
   - [←] or [♦] [DEL] to delete the character to the left or right of the cursor, respectively.

   [2nd] [a-lock] type anything you want

4. Leave the **Text Editor** and display the Home screen.

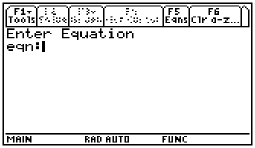
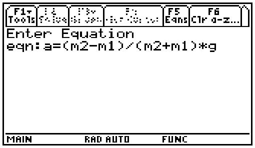
   Your text session was stored automatically as you typed. Therefore, you do not need to save the session manually before exiting the **Text Editor**.

   [HOME]

Steps and keystrokes	Display
5. Return to the current session on the **Text Editor**. Notice that the displayed session is exactly the same as you left it. Press [2nd] [↔]	

Numeric Solver

Consider the equation a=(m2−m1)/(m2+m1)∗g, where the known values are m2=10 and g=9.8. If you assume that a=1/3 g, find the value of m1.

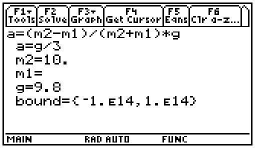
Steps and keystrokes	Display
1. Use [APPS] to display the **Numeric Solver.**	
2. Enter the equation. When you press [ENTER] or [▼], the screen lists the variables used in the equation. [alpha] A [=] [(] [alpha] M2 [−] [alpha] M1 [)] [÷] [(] [alpha] M2 [+] [alpha] M1 [)] [×] [alpha] G [ENTER]	

Steps and keystrokes	Display

3. Enter values for each variable, except the unknown variable m1.

Define m2 and g first. Then define a. (You must define g before you can define a in terms of g.) Accept the default for bound. If a variable has been defined previously, its value is shown as a default.

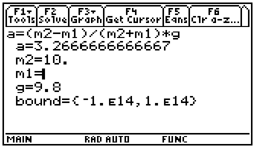
▼ 10 ▼ ▼ 9.8 ▲ ▲ ▲ [alpha] G [÷] 3



4. Move the cursor to the unknown variable m1.

Optionally, you can enter an initial guess for m1. Even if you enter a value for all variables, the Numeric Solver solves for the variable marked by the cursor.

Press ▼ ▼

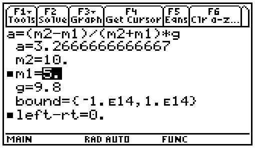


g/3 is evaluated when you move the cursor off the line.

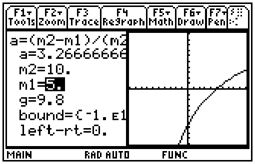
5. Solve for the unknown variable.

To check the solution's accuracy, the left and right sides of the equation are evaluated separately. The difference is shown as left-rt. If the solution is precise, left-rt=0.

Press [F2]

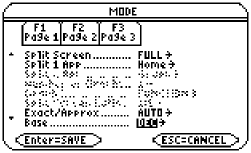
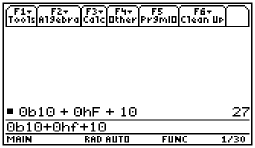


■ marks the calculated values.

Steps and keystrokes	Display
6. Graph the solution using a **ZoomStd** viewing window. The graph is displayed in a split screen. You can explore the graph by tracing, zooming, etc. The variable marked by the cursor (unknown variable m1) is on the x axis, and left-rt is on the y axis. Press [F3] 3	
7. Return to the **Numeric Solver** and exit the split screen. You can press [ENTER] or ⊖ to redisplay the list of variables. Press [2nd] [↔] [F3] 2	

Number Bases

Calculate 10 binary (base 2) + F hexadecimal (base 16) + 10 decimal (base 10). Then, use the ▸ operator to convert an integer from one base to another. Finally, see how changing the Base mode affects the displayed results.

Steps and keystrokes	Display
1. Display the **MODE** dialog box, Page 2. For **Base** mode, select **DEC** as the default number base. Integer results are displayed according to the **Base** mode. Fractional and floating-point results are always displayed in decimal form. Press [MODE] [F2] (use ⊝ to move to **Base** mode) ⊙ 1 [ENTER]	
2. Calculate 0b10+0hF+10. To enter a binary or hex number, you must use the 0b or 0h prefix (zero and the letter B or H). Otherwise, the entry is treated as a decimal number. **Note:** The 0b or 0h prefix is a zero, not the letter O, followed by B or H. 0 [alpha] B 10 [+] 0 [2nd] [a-lock] HF [alpha] [+] 10 [ENTER]	

Steps and keystrokes | **Display**

3. Add 1 to the result and convert it to binary.

   [2nd] [▶] displays the ▶ conversion operator.

   [+] 1 [2nd] [▶] [2nd] [a-lock] BIN [alpha] [ENTER]

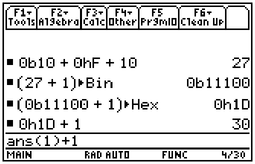
4. Add 1 to the result and convert it to hexadecimal.

   [+] 1 [2nd] [▶] [2nd] [a-lock] HEX [alpha] [ENTER]

5. Add 1 to the result and leave it in the default decimal base.

   Results use the 0b or 0h prefix to identify the base.

   Press [+] 1 [ENTER]



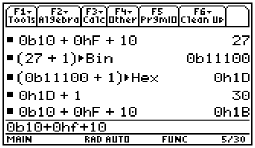
6. Change the **Base** mode to **HEX**.

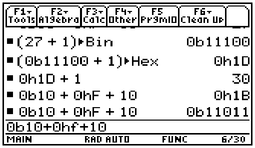
   When **Base = HEX** or **BIN**, the magnitude of a result is restricted to certain size limitations.

   Press [MODE] [F2] (use ⊖ to move to **Base** mode) ⊙ 2 [ENTER]

7. Calculate 0b10+0hF+10.

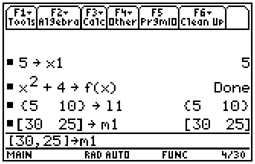
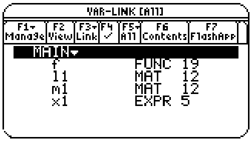
   0 [alpha] B 10 [+] 0 [2nd] [a-lock] HF [alpha] [+] 10 [ENTER]

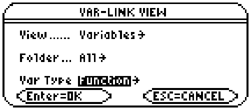
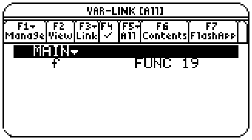
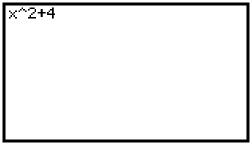


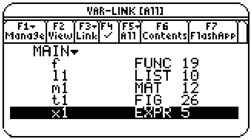
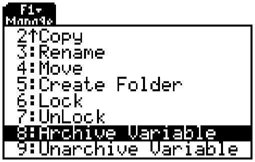
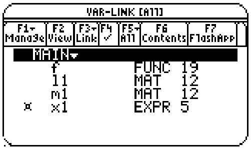
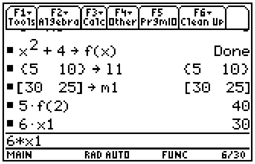
Steps and keystrokes	Display
8. Change the **Base** mode to **BIN**. Press [MODE] [F2] (use ⊖ to move to **Base** mode) ⊕ 3 [ENTER]	
9. Re-enter 0b10+0hF+10. Press [ENTER]	

Memory and Variable Management

Assign values to a variety of variable data types. Use the **VAR-LINK** screen to view a list of the defined variables. Then move a variable to the user data archive memory and explore the ways in which you can and cannot access an archived variable. (Archived variables are locked automatically.) Finally, unarchive the variable and delete the unused variables so that they will not take up memory.

Steps and keystrokes	Display
1. From the Home screen, assign variables with the following variable types. Expression: 5 →x1 Function: x^2+4 →f(x) List: {5,10} →L1 Matrix: [30,25] →m1 HOME CLEAR 5 STO▸ X1 ENTER X ^ 2 + 4 STO▸ alpha F (X) ENTER 2nd [{] 5 , 10 2nd [}] STO▸ alpha L1 ENTER 2nd [[] 30 , 25 2nd []] STO▸ alpha M1 ENTER	
2. Suppose you start to perform an operation using a function variable but can't remember its name. Press 5 ×	5∗
3. Display the **VAR-LINK** screen. This example assumes that the variables assigned above are the only ones defined. Press 2nd [VAR-LINK]	

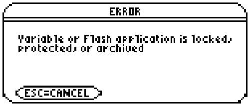
Steps and keystrokes	Display
4. Change the screen's view to show only function variables. Although this may not seem particularly useful in an example with four variables, consider how useful it could be if there were many variables of all different types. Press [F2] ↓ ↓ → 5 [ENTER]	 
5. Highlight the **f** function variable, and view its contents. Notice that the function was assigned using **f(x)** but is listed as **f** on the screen. ↓ [2nd] [F6]	
6. Close the Contents window. Press [ESC]	
7. With the **f** variable still highlighted, close **VAR-LINK** to paste the contents of the variable to the entry line. Notice that "(" is pasted. Press [ENTER]	5∗f(
8. Complete the operation. Press 2 [)] [ENTER]	5∗f(2)

Steps and keystrokes	Display
1. Redisplay **VAR-LINK**, and highlight the variable you want to archive. The previous change in view is no longer in effect. The screen lists all defined variables. Press [2nd] [VAR-LINK] (use ⊖ to highlight **x1**)	
2. Use the [F1] **Manage** toolbar menu to archive the variable. ¤ indicates the variable is archived. Press [F1] 8	 
3. Return to the Home screen and use the archived variable in a calculation. [HOME] 6 [×] X1 [ENTER]	

Steps and keystrokes — **Display**

4. Attempt to store a different value to the archived variable.

   Press 10 [STO▸] X1 [ENTER]



5. Cancel the error message.

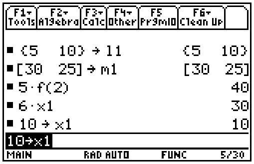
   Press [ESC]

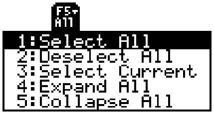
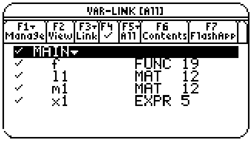
6. Use **VAR-LINK** to unarchive the variable.

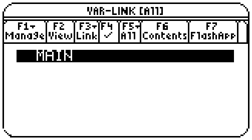
   Press [2nd] [VAR-LINK] (use ⊖ to highlight **x1**) [F1] 9

7. Return to the Home screen and store a different value to the unarchived variable.

   [HOME] [ENTER]



Steps and keystrokes	Display
1. Display **VAR-LINK**, and use the F5 **All** toolbar menu to select all variables. A ✓ mark indicates items that are selected. Notice that this also selected the **MAIN** folder. **Note:** Instead of using F5 (if you don't want to delete all your variables), you can select individual variables. Highlight each variable to delete and press F4. Press F5 1	 
2. Use F1 to delete. **Note:** You can press ← (instead of F1 **1**) to delete the marked variables. Press F1 1	
3. Confirm the deletion. Press ENTER	

Steps and keystrokes	Display
4. Because [F5] **1** also selected the **MAIN** folder, an error message states that you cannot delete the **MAIN** folder. Acknowledge the message. When **VAR-LINK** is redisplayed, the deleted variables are not listed. Press [ENTER]	
5. Close **VAR-LINK** and return to the current application (Home screen in this example). When you use [ESC] (instead of [ENTER]) to close **VAR-LINK**, the highlighted name is not pasted to the entry line. Press [ESC]	

Operating the Calculator

Turning the Calculator On and Off

You can turn your graphing calculator on and off manually by using the [ON] and [2nd] [OFF] (or [♦] [OFF]) keys. To prolong battery life, the APD™ (Automatic Power Down™) feature lets the calculator turn itself off automatically.

Turning the Calculator On

Press [ON].

- If you turned the unit off by pressing [2nd] [OFF], the unit returns to either the Apps desktop or the Home screen.
- If you turned the unit off by pressing [♦] [OFF] or if the unit turned itself off through APD, the unit returns to whichever application you used last.

Turning the Calculator Off

You can use either of the following keys to turn off your graphing calculator.

Press:	Description
[2nd] [OFF] (press [2nd] and then press [OFF])	Settings and memory contents are retained by the Constant Memory™ feature. However: • You cannot use [2nd] [OFF] if an error message is displayed. • When you turn the calculator on again, it displays either the Home screen or the Apps desktop (regardless of the last application you used).
[♦] [OFF] (press [♦] and then press [OFF])	Similar to [2nd] [OFF] except: • You can use [♦] [OFF] if an error message is displayed. • When you turn the calculator on again, it will be exactly as you left it.

Note: [OFF] is the second function of the [ON] key.

APD (Automatic Power Down)

After several minutes without any activity, the calculator turns itself off automatically. This feature is called APD.

When you press [ON], the calculator will be exactly as you left it.

- The display, cursor, and any error conditions are exactly as you left them.
- All settings and memory contents are retained.

APD does not occur if a calculation or program is in progress, unless the program is paused. If a program is running, but waiting for a key press, APD will occur after several minutes of inactivity.

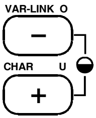
Setting the Display Contrast

The brightness and contrast of the display depend on room lighting, battery freshness, viewing angle, and the adjustment of the display contrast. The contrast setting is retained in memory when the graphing calculator is turned off.

Adjusting the Display Contrast

You can adjust the display contrast to suit your viewing angle and lighting conditions.

To:	Press and hold both:
Decrease (lighten) the contrast	[♦] and [−]
Increase (darken) the contrast	[♦] and [+]



Contrast keys

If you press and hold [♦] [−] or [♦] [+] too long, the display may go completely black or blank. To make finer adjustments, hold [♦] and then tap [−] or [+].

When to Replace Batteries

As the batteries get low, the display begins to dim (especially during calculations) and you must increase the contrast. If you have to increase the contrast frequently, replace the four alkaline batteries.

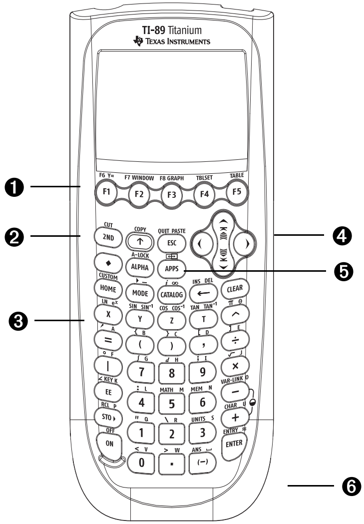
Note: The display may be very dark after you change batteries. Use [♦] [−] to lighten the display.

The status line along the bottom of the display also gives battery information.

Indicator in status line	Description
BATT	Batteries are low.
BATT (reverse video)	Replace batteries as soon as possible.

The TI-89 Titanium Keyboard

Most keys can perform two or more functions, depending on whether you first press a modifier key.



❶ [F1] – [2nd] [F8] open toolbar menus. Select applications (when used with [♦])
❷ [2nd], [♦], [↑], and [alpha] add functionality by increasing the available key commands.
❸ X, Y, and Z are often used in symbolic calculations.
❹ ◁, ▷, △, and ▽ move the cursor.
❺ [APPS] lets you select an application.
❻ [ENTER] evaluates an expression, executes an instruction, selects a menu item, etc.

Modifier Keys

Modifier	Description
[2nd] (second)	Accesses the second function of the next key you press. On the keyboard, these are printed in the same color as the [2nd] key.
[♦] (diamond)	Activates keys that select certain applications, menu items, and other operations from the keyboard. On the keyboard, these are printed in the same color as the [♦] key.
[↑] (shift)	Types an uppercase character for the next letter key you press. [↑] is also used with ▶ and ◀ to highlight characters in the entry line for editing purposes.
[alpha]	Used to type alphabetic letters, including a space character. On the keyboard, these are printed in the same color as the [alpha] key.

Note: Information is available about using [↑] and [alpha].

Examples of [2nd] and [diamond] Modifiers

The [ESC] key is one of several keys that can perform three operations, depending on whether you first press [2nd] or [♦].

The following TI-89 Titanium example shows using the [2nd] or [♦] modifier key with the [ESC] key.

[2nd] [QUIT] accesses QUIT, which is the same color as the [2nd] key.

QUIT PASTE
ESC

[♦] [PASTE] accesses PASTE, which is the same color as the [♦] key.

[ESC] accesses the key's primary function.

Some keys perform only one additional operation, which may require either [2nd] or [♦], depending on the color in which the operation is printed on the keyboard and where it is positioned above the key.

CUT
2nd

On the TI-89 Titanium, [♦] [CUT] accesses CUT, which is the same color as the [♦] key.

When you press a modifier such as [2nd] or [♦], a 2ND or ♦ indicator appears in the status line at the bottom of the display. If you press a modifier by accident, press it again (or press [ESC]) to cancel its effect.

Other Important Keys You Need to Be Familiar With

Key	Description
[♦] [Y=]	Displays the Y= Editor.
[♦] [WINDOW]	Displays the Window Editor.
[♦] [GRAPH]	Displays the Graph screen.
[♦] [TBLSET]	Sets parameters for the Table screen.

Key	Description
[◆] [TABLE]	Displays the Table screen.
[calculator icon]: [◆] [CUT] [◆] [COPY] [◆] [PASTE]	These keys let you edit entered information by performing a cut, copy, or paste operation.
[2nd] [⊣⊢]	Toggles between the last two chosen Apps or between split screen portions.
[2nd] [CUSTOM]	Toggles the custom menu on and off.
[2nd] [▶]	Converts measurement units.
[calculator icon] [◆] [_]	Designates a measurement unit.
[←]	Deletes the character to the left of the cursor (backspaces).
[2nd] [INS]	Toggles between insert and overtype mode for entering information.
[◆] [DEL]	Deletes the character to the right of the cursor.
[calculator icon] [|]	Enters the "*with*" operator, which is used in symbolic calculations.
[2nd] [∫], [2nd] [*d*]	Performs integrations and derivatives.
[2nd] [∠]	Designates an angle in polar, cylindrical, and spherical coordinates.
[2nd] [MATH]	Displays the MATH menu.

Key	Description
2nd [MEM]	Displays the MEMORY screen.
2nd [VAR-LINK]	Displays the VAR-LINK screen for managing variables and Flash applications.
2nd [RCL]	Recalls the contents of a variable.
2nd [UNITS]	Displays the UNITS dialog box.
2nd [CHAR]	Displays the CHAR menu, which lets you select Greek letters, international accented characters, etc.
2nd [ENTRY], 2nd [ANS]	Recalls the previous entry and the last answer, respectively.

Entering Alphabetic Characters

Alphabetic characters are used in expressions such as $\mathbf{x^2+y^2}$ to enter variable names and in the Text Editor (*Text Editor* module).

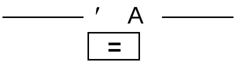
Entering a Letter Character on the TI-89 Titanium

The letters x, y, z, and t are commonly used in algebraic expressions. So that you can type them quickly, these letters are primary keys on the TI-89 Titanium keyboard.

X Y Z T

Other letters are available as the [alpha] function of another key, similar to the [2nd] and [♦] modifiers described in the previous section. For example:

[2nd] [′] types ′, which is the same color as the [2nd] key.



[alpha] [A] displays an A, which is the same color as the [alpha] key.

Typing Alphabetic Characters on the TI-89 Titanium

To:	Press:
Type a single lowercase alpha character.	[alpha] and then the letter key (status line shows)
Type a single uppercase alpha character.	[↑] and then the letter key (status line shows)
Type a space.	[alpha] [␣] (alpha function of the [(-)] key)
Turn on lowercase alpha-lock.	[2nd] [a-lock] (status line shows)
Turn on uppercase ALPHA-lock.	[↑] [a-lock] (status line shows)
Turn off alpha-lock.	[alpha] (turns off upper- and lowercase lock)

Notes:

- On the TI-89 Titanium, you do not need [alpha] or alpha-lock to type x, y, z, or t. But you must use [↑] or uppercase ALPHA-lock for X, Y, Z, or T.
- On the TI-89 Titanium, alpha-lock is always turned off when you change applications, such as going from the Text Editor to the Home screen.

On the TI-89 Titanium, while either type of alpha-lock is on:

- To type a period, comma, or other character that is the primary function of a key, you must turn alpha-lock off.
- To type a second function character such as [2nd] [{], you do not need to turn alpha-lock off. After you type the character, alpha-lock remains on.

Automatic Alpha-Lock in TI-89 Titanium Dialog Boxes

There are certain times when you do not need to press [alpha] or [2nd] [a-lock] to type alphabetic characters on the TI-89 Titanium. Automatic alpha-lock is turned on whenever a dialog box is first displayed. The automatic alpha-lock feature applies to these dialog boxes:

Dialog box	Alpha-lock
Catalog dialog box	All commands are listed in alphabetical order. Press a letter to go to the first command that begins with that letter.
Units dialog box	In each unit category, type the first letter of a unit or constant. See *Constants and Measurement Units* for more information.
Dialog boxes with entry fields	These include, but are not limited to: Create New Folder, Rename, and Save Copy As.

Note: To type a number, press [alpha] to turn alpha-lock off. Press [alpha] or [2nd] [a-lock] to resume typing letters.

Alpha-lock is *not* turned on in dialog boxes that require numeric-only entries. The dialog boxes that accept only numeric entries are: Resize Matrix, Zoom Factors, and Table Setup.

For Special Characters

Use the [2nd] [CHAR] menu to select from a variety of special characters. For more information, refer to “Entering Special Characters” in the *Text Editor* module.

Entering Numbers

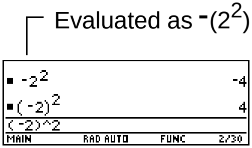
The keypad lets you enter positive and negative numbers for your calculations. You can also enter numbers in scientific notation.

Entering a Negative Number

1. Press the negation key [(-)]. (Do not use the subtraction key [−].)
2. Type the number.

To see how your graphing calculator evaluates a negation in relation to other functions, refer to the Equation Operating System (EOS™) hierarchy in the *Technical Reference* module. For example, it is important to know that functions such as x^2 are evaluated before negation.

Use [(] and [)] to include parentheses if you have any doubt about how a negation will be evaluated.



If you use [−] instead of [(-)] (or vice versa), you may get an error message or you may get unexpected results. For example:

- 9 [×] [(-)] 7 = -63
  – but –
  9 [×] [−] 7 displays an error message.
- 6 [−] 2 = 4
  – but –
  6 [(-)] 2 = -12 since it is interpreted as 6(-2), implied multiplication.
- [(-)] 2 [+] 4 = 2
  – but –
  [−] 2 [+] 4 subtracts 2 from the previous answer and then adds 4.

Important: Use [−] for subtraction and use [(-)] for negation.

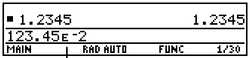
Entering a Number in Scientific Notation

1. Type the part of the number that precedes the exponent. This value can be an expression.

2. Press:
   [EE]
   E appears in the display.
3. Type the exponent as an integer with up to 3 digits. You can use a negative exponent.

Entering a number in scientific notation does not cause the answers to be displayed in scientific or engineering notation.

The display format is determined by the mode settings and the magnitude of the number.



Represents 123.45×10^{-2}

Entering Expressions and Instructions

You perform a calculation by evaluating an expression. You initiate an action by executing the appropriate instruction. Expressions are calculated and results are displayed according to the mode settings.

Expression	Consists of numbers, variables, operators, functions, and their arguments that evaluate to a single answer. For example: πr^2+3. • Enter an expression in the same order that it normally is written. • In most places where you are required to enter a value, you can enter an expression.
Operator	Performs an operation such as +, –, ∗, ^. • Operators require an argument before and after the operator. For example: 4+5 and 5^2.
Function	Returns a value. • Functions require one or more arguments (enclosed in parentheses) after the function. For example: √(5) and **min(5,8)**.
Instruction	Initiates an action. • Instructions cannot be used in expressions. • Some instructions do not require an argument. For example: ClrHome. • Some require one or more arguments. For example: **Circle 0,0,5**. **Note:** For instructions, do not put the arguments in parentheses.

Notes:

- The *Technical Reference* module describes all of the built-in functions and instructions.

- This guidebook uses the word command as a generic reference to both functions and instructions.

Implied Multiplication

The graphing calculator recognizes implied multiplication, provided it does not conflict with a reserved notation.

	If you enter:	The calculator interprets it as:
Valid	2π 4 sin(46) 5(1+2) or (1+2)5 [1,2]a 2(a)	2∗π 4∗sin(46) 5∗(1+2) or (1+2)∗5 [a 2a] 2∗a
Invalid	xy a(2) a[1,2]	Single variable named xy Function call Matrix index to element a[1,2]

Parentheses

Expressions are evaluated according to the Equation Operating System (EOS™) hierarchy described in the *Technical Reference* module. To change the order of evaluation or just to ensure that an expression is evaluated in the order you require, use parentheses.

Calculations inside a pair of parentheses are completed first. For example, in 4(1+2), EOS first evaluates (1+2) and then multiplies the answer by 4.

Entering an Expression

Type the expression, and then press [ENTER] to evaluate it. To enter a function or instruction name on the entry line, you can:

- Press its key, if available. For example, press: [2nd] [SIN]

  – or –

- Select it from a menu, if available. For example, select **2:abs** from the Number submenu of the MATH menu.

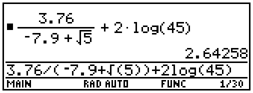
  – or –

- Type the name letter-by-letter from the keyboard. (On the TI-89 Titanium, use [alpha] and [2nd] [a-lock] to type letters.) You can use any mixture of uppercase or lowercase letters. For example, type **sin(** or **Sin(**.

Example

Calculate 3.76 ÷ (⁻7.9 + √5) + 2 log 45.

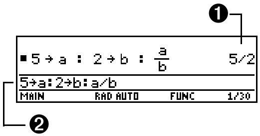
Type the function name in this example.

Press	Display
3.76 [÷] [(] [(-)] 7.9 [+] [2nd] [√]	3.76/(-7.9+√([2nd] [√] inserts √(because its argument must be in parentheses.
5 [)] [)]	3.76/(-7.9+√(5)) Use [)] once to close √(5) and again to close (-7.9 + √5).
[+] 2 [2nd] [a-lock] LOG [alpha] [(] 45 [)]	3.76/(-7.9+√(5))+2log(45) log requires ( ) around its argument.
[ENTER]	

Note: You can also select **log** by using [CATALOG]

Entering Multiple Expressions on a Line

To enter more than one expression or instruction at a time, separate them with a colon by pressing [2nd] [:].

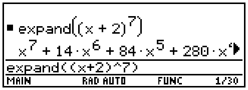


❶ Displays last result only.
❷ → is displayed when you press [STO▶] to store a value to a variable.

If an Entry or Answer Is Too Long for One Line

In the history area, if both the entry and its answer cannot be displayed on one line, the answer is displayed on the next line.

If an entry or answer is too long to fit on one line, ▶ is displayed at the end of the line.



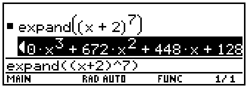
To view the entire entry or answer:

1. Press ⊙ to move the cursor from the entry line up into the history area. This highlights the last answer.

2. As necessary, use ⏶ and ⏷ to highlight the entry or answer you want to view. For example, ⏶ moves from answer to entry, up through the history area.

3. Use ⏵ and ⏴ or [2nd] ⏵ and [2nd] ⏴ to scroll right and left.

   Note: When you scroll to the right, ◀ is displayed at the beginning of the line.



4. To return to the entry line, press [ESC].

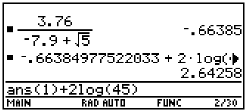
Continuing a Calculation

When you press [ENTER] to evaluate an expression, the graphing calculator leaves the expression on the entry line and highlights it. You can continue to use the last answer or enter a new expression.

If you press:	The calculator:
[+], [−], [×], [÷], [^], or [STO▸]	Replaces the entry line with the variable **ans(1)**, which lets you use the last answer as the beginning of another expression.
Any other key	Erases the entry line and begins a new entry.

Example

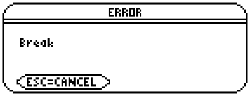
Calculate $3.76 \div ({}^{-}7.9 + \sqrt{5})$. Then add 2 log 45 to the result.

Press	Display
3.76 [÷] [(] [(-)] 7.9 [+] [2nd] [√] 5 [)] [)] [ENTER]	
[+] 2 [2nd] [a-lock] LOG [alpha] [(] 45 [)] [ENTER]	When you press [+], the entry line is replaced with the variable **ans(1)**, which contains the last answer.

Stopping a Calculation

When a calculation is in progress, BUSY appears on the right end of the status line. To stop the calculation, press [ON].

There may be a delay before the Break message is displayed.



Press [ESC] to return to the current application.

Formats of Displayed Results

A result may be calculated and displayed in any of several formats. This section describes the modes and their settings that affect the display formats. You can check or change your current mode settings.

Pretty Print Mode

By default, **Pretty Print = ON**. Exponents, roots, fractions, etc., are displayed in the same form in which they are traditionally written. You can use [MODE] to turn pretty print off and on.

Pretty Print	
ON	**OFF**
π^2, $\frac{\pi}{2}$, $\sqrt{\frac{x-3}{2}}$	π^2, π/2, √((x–3)/2)

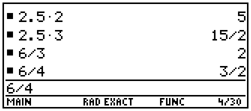
The entry line does not show an expression in pretty print. If pretty print is turned on, the history area will show both the entry and its result in pretty print after you press [ENTER].

Exact/Approx Mode

By default, **Exact/Approx = AUTO**. You can use [MODE] to select from three settings.

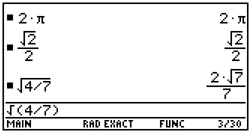
Because AUTO is a combination of the other two settings, you should be familiar with all three settings.

EXACT — Any result that is not a whole number is displayed in a fractional or symbolic form (1/2, π, $\sqrt{2}$, etc.).



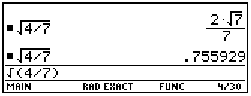
Shows whole-number results.

Shows simplified fractional results.



Shows symbolic π.

Shows symbolic form of roots that cannot be evaluated to a whole number.

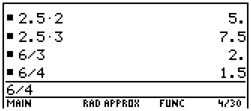


Press [♦] [ENTER] to temporarily override the EXACT setting and display a floating-point result.

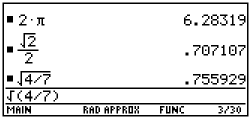
Note: By retaining fractional and symbolic forms, EXACT reduces rounding errors that could be introduced by intermediate results in chained calculations.

APPROXIMATE — All numeric results, where possible, are displayed in floating-point (decimal) form.

> **Note:** Results are rounded to the precision of your graphing calculator and displayed according to current mode settings.



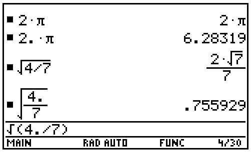
Fractional results are evaluated numerically.



Symbolic forms, where possible, are evaluated numerically

> Because undefined variables cannot be evaluated, they are treated algebraically. For example, if the variable r is undefined, $\pi r^2 = 3.14159 \cdot r^2$.

AUTO — Uses the EXACT form where possible, but uses the APPROXIMATE form when your entry contains a decimal point. Also, certain functions may display APPROXIMATE results even if your entry does not contain a decimal point.



A decimal in the entry forces a floating-point result.

Note: To retain an EXACT form, use fractions instead of decimals. For example, use 3/2 instead of 1.5.

The following chart compares the three settings.

Entry	Exact Result	Approximate Result	Auto Result
8/4	2	2.	2
8/6	4/3	1.33333	4/3
8.5∗3	51/2	25.5	25.5
√(2)/2	$\frac{\sqrt{2}}{2}$	.707107	$\frac{\sqrt{2}}{2}$
π∗2	2·π	6.28319	2·π
π∗2.	2·π	6.28319	6.28319

— A decimal in the entry forces a floating-point result in AUTO.

Note: To evaluate an entry in APPROXIMATE form, regardless of the current setting, press [♦] [ENTER].

Display Digits Mode

By default, **Display Digits = FLOAT 6**, which means that results are rounded to a maximum of six digits. You can use [MODE] to select different settings. The settings apply to all exponential formats.

Internally, the calculator calculates and retains all decimal results with up to 14 significant digits (although a maximum of 12 are displayed).

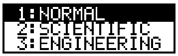
Setting	Example		Description
FIX (0–12)	123. 123.5 123.46 123.457	(FIX 0) (FIX 1) (FIX 2) (FIX 3)	Results are rounded to the selected number of decimal places.
FLOAT	123.456789012		Number of decimal places varies, depending on the result.
FLOAT (1–12)	1.E 2 1.2E 2 123. 123.5 123.46 123.457	(FLOAT 1) (FLOAT 2) (FLOAT 3) (FLOAT 4) (FLOAT 5) (FLOAT 6)	Results are rounded to the total number of selected digits.

Notes:

- Regardless of the **Display Digits** setting, the full value is used for internal floating-point calculations to ensure maximum accuracy.
- A result is automatically shown in scientific notation if its magnitude cannot be displayed in the selected number of digits.

Exponential Format Mode

By default, **Exponential Format = NORMAL**. You can use [MODE] to select from three settings.



Setting	Example	Description
NORMAL	12345.6	If a result cannot be displayed in the number of digits specified by the Display Digits mode, the calculator switches from NORMAL to SCIENTIFIC for that result only.
SCIENTIFIC	1.23456E 4 (❶ under 1.23456, ❷ under 4)	1.23456×10^4
ENGINEERING	1.23456E 3 (❸ under 1.23456, ❹ under 3)	12.3456×10^3

❶ Always 1 digit to the left of the decimal point.
❷ Exponent (power of 10).

❸ May have 1, 2, or 3 digits to the left of the decimal point.
❹ Exponent is a multiple of 3.

Note: In the history area, a number in an entry is displayed in SCIENTIFIC if its absolute value is less than .001.

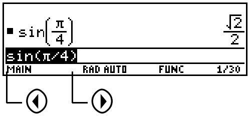
Editing an Expression in the Entry Line

Knowing how to edit an entry can be a real time-saver. If you make an error while typing an expression, it’s often easier to correct the mistake than to retype the entire expression.

Removing the Highlight from the Previous Entry

After you press [ENTER] to evaluate an expression, the calculator leaves that expression on the entry line and highlights it. To edit the expression, you must first remove the highlight; otherwise, you may clear the expression accidentally by typing over it.

To remove the highlight, move the cursor toward the side of the expression you want to edit.



◄ moves the cursor to the beginning.
► moves the cursor to the end of the expression.

Moving the Cursor

After removing the highlight, move the cursor to the applicable position within the expression.

To move the cursor:	Press:	
Left or right within an expression.	◄ or ►	Hold the pad to repeat the movement.
To the beginning of the expression.	[2nd] ◄	
To the end of the expression.	[2nd] ►	

Note: If you accidentally press ▲ instead of ◄ or ►, the cursor moves up into the history area. Press [ESC] or press ▼ until the cursor returns to the entry line.

Deleting a Character

To delete:	Press:	
The character to the left of the cursor.	[←]	Hold [←] to delete multiple characters.
The character to the right of the cursor.	[♦] [←]	
All characters to the right of the cursor.	[CLEAR] (once only)	If there are no characters to the right of the cursor, [CLEAR] erases the entire entry line.

Clearing the Entry Line

To clear the entry line, press:

- [CLEAR] if the cursor is at the beginning or end of the entry line.

  – or –

- [CLEAR] [CLEAR] if the cursor is not at the beginning or end of the entry line. The first press deletes all characters to the right of the cursor, and the second clears the entry line.

Inserting or Overtyping a Character

The calculator has both an insert and an overtype mode. By default, the calculator is in the insert mode. To toggle between the insert and overtype modes, press [2nd] [INS].

If in:	The next character you type:
Insert mode Thin cursor between characters	Will be inserted at the cursor.
Overtype mode Cursor highlights a character	Will replace the highlighted character.

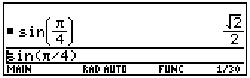
Note: Look at the cursor to see if you're in insert or overtype mode.

Replacing or Deleting Multiple Characters

First, highlight the applicable characters. Then, replace or delete all the highlighted characters.

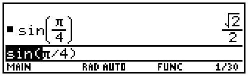
To highlight multiple characters:

1. Move the cursor to either side of the characters you want to highlight.



To replace **sin(** with **cos(**, place the cursor beside **sin**.

2. Hold [↑] and press ◀ or ▶ to highlight characters left or right of the cursor.

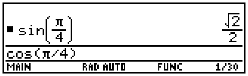


Hold [↑] and press ▶ ▶ ▶ ▶.

To replace or delete the highlighted characters:

1. Type the new characters.
2. Press [←].

   Note: When you highlight characters to replace, remember that some function keys automatically add an open parenthesis.



Menus

To leave the keyboard uncluttered, the calculator uses menus to access many operations. This section gives an overview of how to select an item from any menu. Specific menus are described in the appropriate modules.

Displaying a Menu

Press:	To display:
[F1], [F2], etc.	A toolbar menu — Drops down from the toolbar at the top of most application screens. Lets you select operations useful for that application.
[APPS]	**Apps desktop** or **APPLICATIONS** menu — Lets you select from a list of applications.
[2nd] [CHAR]	**CHAR** menu — Lets you select from categories of special characters (Greek, math, etc.).
[2nd] [MATH]	**MATH** menu — Lets you select from categories of math operations.
[CATALOG]	**CATALOG** menu — Lets you select from a complete, alphabetic list of built-in functions and instructions. Also lets you select user-defined functions or Flash application functions (if any have been defined or loaded).
[2nd] [CUSTOM]	**CUSTOM** menu — Lets you access a menu that you can customize to list any available function, instruction, or character. The calculator includes a default custom menu, which you can modify or redefine. Refer to the *Calculator Home Screen* and/or the *Programming* module for more information on the custom menu.

To select an item from the displayed menu, either:

- Press the number or letter shown to the left of that item. For a letter on the TI-89 Titanium, press [alpha] and then a letter key.

  – or –

- Use the cursor pad ↓ and ↑ to highlight the item, and then press [ENTER]. (Note that pressing ↑ from the first item moves the highlight to the last item, and vice versa.)

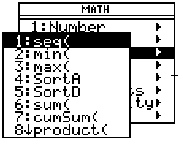
▼ indicates that a menu will drop down from the toolbar when you press [F2].

To select **factor**, press 2 or ↓ [ENTER]. This closes the menu and inserts the function at the cursor location.

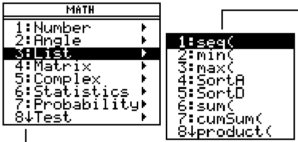
factor(

Items Ending with ▶ (Submenus)

If you select a menu item ending with ▶, a submenu is displayed. You then select an item from the submenu.



Because of limited screen size, the TI-89 Titanium overlaps these menus.



For example, List displays a submenu that lets you select a specific List function.

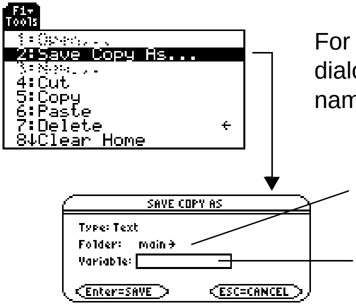
↓ indicates that you can use the cursor pad to scroll down for additional items.

For items that have a submenu, you can use the cursor pad as described below.

- To display the submenu for the highlighted item, press ▶. (This is the same as selecting that item.)
- To cancel the submenu without making a selection, press ◀. (This is the same as pressing [ESC].)
- To wrap to the last menu item directly from the first menu item, press ▲. To wrap to the first menu item directly from the last menu item, press ▼.

Items Containing “. . .” (Dialog Boxes)

If you select a menu item containing “**...**” (ellipsis marks), a dialog box is displayed for you to enter additional information.



For example, **Save Copy As ...** displays a dialog box that prompts you to select a folder name and type a variable name.

→ indicates that you can press ▶ to display and select from a menu.

An input box indicates that you must type a value. (Alpha-lock is automatically turned on for the TI-89 Titanium.)

After typing in an input box such as Variable, you must press [ENTER] twice to save the information and close the dialog box.

Canceling a Menu

To cancel the current menu without making a selection, press [ESC]. Depending on whether any submenus are displayed, you may need to press [ESC] several times to cancel all displayed menus.

Moving from One Toolbar Menu to Another

To move from one toolbar menu to another without making a selection, either:

- Press the key ([F1], [F2], etc.) for the other toolbar menu.

  – or –

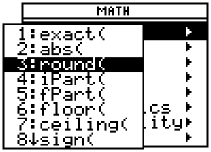
- Use the cursor pad to move to the next (press ▶) or previous (press ◀) toolbar menu. Pressing ▶ from the last menu moves to the first menu, and vice versa.

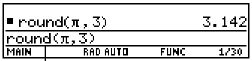
When using ▶, be sure that an item with a submenu is not highlighted. If so, ▶ displays that item's submenu instead of moving to the next toolbar menu.

Example: Selecting a Menu Item

Round the value of π to three decimal places. Starting from a clear entry line on the Home screen:

1. Press [2nd] [MATH] to display the **MATH** menu.
2. Press **1** to display the **Number** submenu. (Or press [ENTER] since the first item is automatically highlighted.)
3. Press **3** to select **round**. (Or press ▼ ▼ and [ENTER].)
4. Press [2nd] [π] [,] **3** [)] and then [ENTER] to evaluate the expression.





❶

❶ Selecting the function in Step 3 automatically typed **round(** on the entry line.

Selecting an Application

The graphing calculator has different applications that let you solve and explore a variety of problems. You can select an application from a menu, the Apps desktop, or you can access commonly used applications directly from the keyboard.

From the APPLICATIONS Menu

1. If the Apps desktop is off, press [APPS] to display a menu that lists the applications.

   Note: To cancel the menu without making a selection, press [ESC].

2. Select an application. Either:

   - Use the cursor pad ▼ or ▲ to highlight the application and then press [ENTER].

     – or –

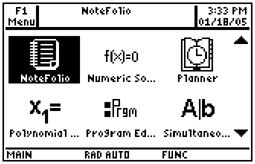
   - Press the number or letter for that application.

Application:	Lets you:
FlashApps	Display a list of Flash applications, if any.
Y= Editor	Define, edit, and select functions or equations for graphing.
Window Editor	Set window dimensions for viewing a graph.
Graph	Display graphs.

Application:	Lets you:
Table	Display a table of variable values that correspond to an entered function.
Data/Matrix Editor	Enter and edit lists, data, and matrices. You can perform statistical calculations and graph statistical plots.
Program Editor	Enter and edit programs and functions.
Text Editor	Enter and edit a text session.
Numeric Solver	Enter an expression or equation, define values for all but one variable, and then solve for the unknown variable.
Home	Enter expressions and instructions, and perform calculations.

From the Apps Desktop

Press the first letter of the application name, or use the cursor keys to highlight an application icon on the Apps desktop and press ENTER. (If you press the first letter of the application and more than one application begins with that letter, the first one alphabetically is highlighted). The application either opens directly or displays a dialog box. (Your Apps desktop may vary from the one shown below.)



The most common dialog box lists these options for the application:

Option	Description
Current	Returns the screen displayed when you last viewed the App. (If there is no current file/variable for the selected App, this option defaults to New if you press [ENTER].)
Open	Lets you select an existing file.
New	Creates a new file with the name typed in the field.

Select an option and press [ENTER]. The application appears.

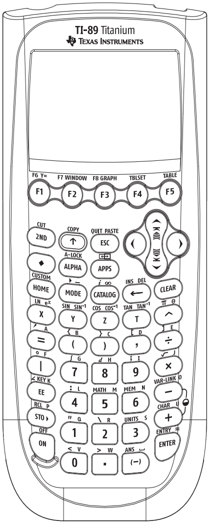
Note: The general term *variable* is used to refer to the application data files that you create.

Use any of these methods to return to the Apps desktop from within an application:

- Press [APPS].
- In full-screen mode, press [2nd] [QUIT].
- In split-screen mode, press [2nd] [QUIT] to open the full-screen view of the active application, then press [2nd] [QUIT] again.

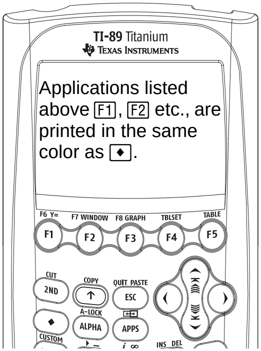
To return to the last open application from the Apps desktop, press [2nd] [⇔].

From the Keyboard



You can access commonly used applications from the keyboard. On the TI-89 Titanium for example, [♦] [Y=] is the same as pressing [♦] and then [F1]. This guidebook uses the notation [♦] [Y=], similar to the notation used in second functions.

Application:	Press:
Home	[HOME] [CALC HOME]
Y= Editor	[♦] [Y=]
Window Editor	[♦] [WINDOW]
Graph	[♦] [GRAPH]
Table Setup	[♦] [TBLSET]
Table Screen	[♦] [TABLE]

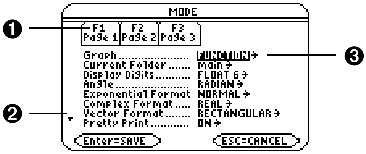


Setting Modes

Modes control how numbers and graphs are displayed and interpreted. Mode settings are retained by the Constant Memory™ feature when the graphing calculator is turned off. All numbers, including elements of matrices and lists, are displayed according to the current mode settings.

Checking Mode Settings

Press [MODE] to display the MODE dialog box, which lists the modes and their current settings.



❶ There are three pages of mode listings. Press [F1], [F2], or [F3] to quickly display a particular page.

❷ Indicates you can scroll down to see additional modes.

❸ → indicates that you can press ► or ◄ to display and select from a menu.

Note: Modes that are not currently valid are dimmed. For example, on **Page 2, Split 2 App** is not valid when **Split Screen = FULL**. When you scroll through the list, the cursor skips dimmed settings.

Changing Mode Settings

From the MODE dialog box:

1. Highlight the mode setting you want to change. Use ▼ or ▲ (with [F1], [F2], or [F3]) to scroll through the list.
2. Press ► or ◄ to display a menu that lists the valid settings. The current setting is highlighted.
3. Select the applicable setting. Either:
   - Use ▼ or ▲ to highlight the setting and press [ENTER].

     – or –

- Press the number or letter for that setting.

  Note: To cancel a menu and return to the **MODE** dialog box without making a selection, press [ESC].

4. Change other mode settings, if necessary.
5. When you finish all your changes, press [ENTER] to save the changes and exit the dialog box.

Important: If you press [ESC] instead of [ENTER] to exit the **MODE** dialog box, any mode changes you made will be canceled.

Overview of the Modes

Note: For detailed information about a particular mode, look in the applicable section of this guidebook.

Mode	Description
Graph	Type of graphs to plot: FUNCTION, PARAMETRIC, POLAR, SEQUENCE, 3D, or DE.
Current Folder	Folder used to store and recall variables. Unless you have created additional folders, only the MAIN folder is available. Refer to “Using Folders to Store Independent Sets of Variables” in *Calculator Home Screen.*
Display Digits	Maximum number of digits (FLOAT) or fixed number of decimal places (FIX) displayed in a floating-point result. Regardless of the setting, the total number of displayed digits in a floating-point result cannot exceed 12.
Angle	Units in which angle values are interpreted and displayed: RADIAN, DEGREE or GRADIAN.

Mode	Description
Exponential Format	Notation used to display results: NORMAL, SCIENTIFIC, or ENGINEERING.
Complex Format	Format used to display complex results, if any: REAL (complex results are not displayed unless you use a complex entry), RECTANGULAR, or POLAR.
Vector Format	Format used to display 2- and 3-element vectors: RECTANGULAR, CYLINDRICAL, or SPHERICAL.
Pretty Print	Turns the pretty print display feature OFF or ON.
Split Screen	Splits the screen into two parts and specifies how the parts are arranged: FULL (no split screen), TOP-BOTTOM, or LEFT-RIGHT. Refer to the *Split Screens* module.
Split 1 App	Application in the top or left side of a split screen. If you are not using a split screen, this is the current application.
Split 2 App	Application in the bottom or right side of a split screen. This is active only for a split screen.
Number of Graphs	For a split screen, lets you set up both sides of the screen to display independent sets of graphs.
Graph 2	If **Number of Graphs = 2**, selects the type of graph in the Split 2 part of the screen. Refer to *Calculator Home Screen.*
Exact/Approx	Calculates expressions and displays results in numeric form or in rational/symbolic form: AUTO, EXACT, or APPROXIMATE.
Base	Lets you perform calculations by entering numbers in decimal (DEC), hexadecimal (HEX), or binary (BIN) form.

Mode	Description
Unit System	Lets you select from three systems of measurement to specify the default units for displayed results: **SI** (metric or MKS); **Eng/US** (feet, pounds, etc.); or **Custom**.
Custom Units	Lets you select custom defaults. The mode is dimmed until you select Unit System, **3:CUSTOM**.
Language	Lets you localize the calculator into one of several languages, depending on which language Flash applications are installed.
Apps Desktop	Turns the Apps desktop ON or OFF.

Using the Clean Up Menu to Start a New Problem

On the Home screen, the **Clean Up** toolbar menu lets you start a new calculation from a cleared state without resetting the memory.

Clean Up Toolbar Menu

To display the **Clean Up** menu from the Home screen, press:
[2nd] [F6]

Menu Item	Description
Clear a–z	Clears (deletes) all single-character variable names in the current folder, unless the variables are locked or archived. You will be prompted to press [ENTER] to confirm the action. Single-character variable names are often used in symbolic calculations such as: **solve(a•x^2+b•x+c=0,x)** If any of the variables have already been assigned a value, your calculation may produce misleading results. To prevent this, you can select **1:Clear a–z** before beginning the calculation.
NewProb	Places **NewProb** in the entry line. You must then press [ENTER] to execute the command. **NewProb** performs a variety of operations that let you begin a new problem from a cleared state without resetting the memory: • Clears all single-character variable names in the current folder (same as **1:Clear a–z**), unless the variables are locked or archived. • Turns off all functions and stat plots (**FnOff** and **PlotsOff**) in the current graphing mode. • Performs **ClrDraw**, **ClrErr**, **ClrGraph**, **ClrHome**, **ClrIO**, and **ClrTable**.
Restore custom default	If a custom menu other than the default is in effect, this lets you restore the default. Refer to the *Calculator Home Screen* module for information on the custom menu.

Notes:

- When defining a variable that you want to retain, use more than one character in the name. This prevents it from being deleted inadvertently by **1:Clear a–z**.
- For information about checking and resetting memory or other system defaults, refer to *Memory and Variable Management*.

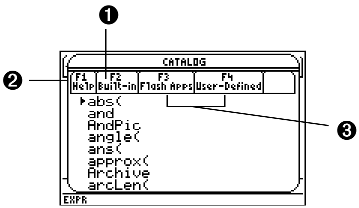
Using the Catalog Dialog Box

The CATALOG provides a way to access any built-in command (functions and instructions) from one convenient list. In addition, the CATALOG dialog box lets you select functions used in Flash applications or user-defined functions (if any have been loaded or defined).

Displaying the CATALOG

To display the CATALOG dialog box, press:
[CATALOG]

The CATALOG defaults to [F2] **Built-in**, which displays an alphabetic list of all pre-installed commands (functions and instructions).



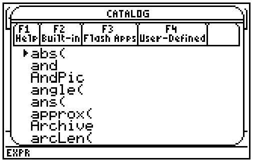
❶ Defaults to [F2] Built-in.
❷ [F1] Help displays a command's parameters in a dialog box.
❸ [F3] and [F4] allow access to Flash application functions and User-Defined functions and programs.

Note: Options that are not currently valid are dimmed. For example, [F3] **Flash Apps** is dimmed if you have not installed a Flash application. [F4] **User-Defined** is dimmed if you have not created a function or a program.

Selecting a Built-in Command from the CATALOG

When you select a command, its name is inserted in the entry line at the cursor location. Therefore, you should position the cursor as necessary before selecting the command.

1. Press:
   [CATALOG]
2. Press [F2] **Built-in**.



- Commands are listed in alphabetical order. Commands that do not start with a letter (+, %, √, Σ, etc.) are at the end of the list.
- To exit the **CATALOG** without selecting a command, press [ESC].

Note: The first time you display the **Built-in** list, it starts at the top of the list. The next time you display the list, it starts at the same place you left it.

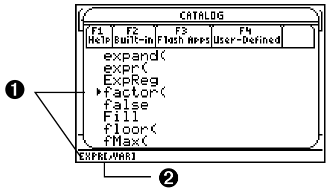
3. Move the ▶ indicator to the command, and press [ENTER].

To move the ▶ indicator:	Press or type:
One function or program at a time	⊖ or ⊕
One page at a time	[2nd] ⊖ or [2nd] ⊕
To the first function that begins with a specified letter	The letter key. (On the TI-89 Titanium, do *not* press [alpha] first. If you do, you need to press [alpha] or [2nd] [a-lock] again before you can type a letter.)

Note: From the top of the list, press ⊕ to move to the bottom. From the bottom, press ⊖ to move to the top.

Information about Parameters

For the command indicated by ▶, the status line shows the required and optional parameters, if any, and their type.



❶ Indicated command and its parameters

❷ Brackets [ ] indicate optional parameters

From the example above, the syntax for **factor** is:

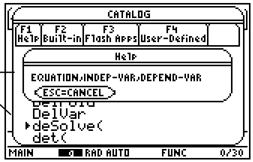
factor(*expression***)**	required
– or –	
factor(*expression,variable***)**	optional

Note: For details about the parameters, refer to that command’s description in the *Technical Reference* module.

Viewing CATALOG Help

You can display a command's parameters in a dialog box by pressing [F1] **Help**. The parameters are the same as those displayed on the status line.

Indicated command and its parameters.

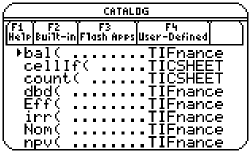


Some commands, such as **ClrDraw**, do not require parameters. If you select one of these commands, parameters will not display on the status line and you will see Unavailable if you press [F1] **Help.**

Press [ESC] to exit the CATALOG Help dialog box.

A Flash application may contain one or more functions. When you select a function, its name is inserted in the entry line at the cursor location. Therefore, you should position the cursor as necessary before selecting the function.

1. Press:
   [CATALOG]
2. Press [F3] **Flash Apps**. (This option is dimmed if no Flash applications are installed.)



- The list is alphabetized by function name. The left column lists functions. The right column lists the Flash application that contains the function.
- Information about a function is displayed in the status line.
- To exit without selecting a function, press [ESC].

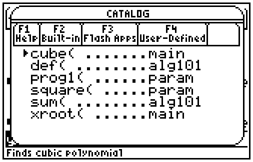
3. Move the ▶ indicator to the function, and press [ENTER].

To move the ▶ indicator:	Press or type:
One function or program at a time	⊖ or ⊕ (down or up arrow)
One page at a time	[2nd] ⊖ or [2nd] ⊕ (down or up arrow)
To the first function that begins with a specified letter	The letter key. (On the TI-89 Titanium, do *not* press [alpha] first. If you do, you need to press [alpha] or [2nd] [a-lock] again before you can type a letter.)

You can create your own functions or programs and then use F4 **User-Defined** to access them. For instructions on how to create functions, see “Creating and Evaluating User-Defined Functions” in *Calculator Home Screen*, and “Overview of Entering a Function” in the *Programming* module. See *Programming* for instructions on how to create and run a program.

When you select a function or program, its name is inserted in the entry line at the cursor location. Therefore, you should position the cursor as necessary before selecting the function or program.

1. Press:
   CATALOG
2. Press F4 **User-Defined.** (This option is dimmed if you have not defined a function or created a program.)



- The list is alphabetized by function / program name. The left column lists functions and programs. The right column lists the folder that contains the function or program.
- If the function or program’s first line is a comment, the comment text is displayed in the status line.
- To exit without selecting a function or program, press ESC.

Note: Use the **VAR-LINK** screen to manage variables, folders, and Flash applications. See the *Memory and Variable Management* module.

3. Move the ▶ indicator to the function or program, and press [ENTER].

To move the ▶ indicator:	Press or type:
One function or program at a time	⊖ or ⊕
One page at a time	[2nd] ⊖ or [2nd] ⊕
To the first function or program that begins with a specified letter	The letter key. (On the TI-89 Titanium, do *not* press [alpha] first. If you do, you need to press [alpha] or [2nd] [a-lock] again before you can type a letter.)

Storing and Recalling Variable Values

When you store a value, you store it as a named variable. You can then use the name instead of the value in expressions. When the calculator encounters the name in an expression, it substitutes the variable's stored value.

Rules for Variable Names

A variable name:

- Can use 1 to 8 characters consisting of letters and digits. This includes Greek letters (but not π), accented letters, and international letters.
  - Do not include spaces.
  - The first character cannot be a digit.
- Can use uppercase or lowercase letters. The names **AB22**, **Ab22**, **aB22**, and **ab22** all refer to the same variable.

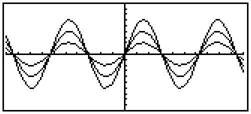
- Cannot be the same as a name that is preassigned by the calculator. Preassigned names include:
  - Built-in functions (such as **abs**) and instructions (such as **LineVert**). Refer to the *Technical Reference* module.
  - System variables (such as **xmin** and **xmax**, which are used to store graph-related values). Refer to the *Technical Reference* module for a list.

Examples

Variable	Description
myvar	OK
a	OK
Log	Not OK, name is preassigned to the **log** function.
Log1	OK
3rdTotal	Not OK, starts with a digit.
circumfer	Not OK, more than 8 characters.

Data Types

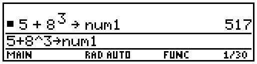
DataTypes	Examples
Expressions	2.54, 1.25E6, 2π, xmin/10, 2+3*i*, $(x–2)^2$, $\sqrt{2}/2$
Lists	{2 4 6 8}, {1 1 2}

DataTypes	Examples
Matrices	$\begin{bmatrix}1 & 0 & 0\end{bmatrix}$, $\begin{bmatrix}1 & 0 & 0\\3 & 4 & 6\end{bmatrix}$
Character strings	“Hello”, “The answer is:”, “xmin/10”
Pictures	
Functions	myfunc(arg), ellipse(x,y,r1,r2)

Storing a Value in a Variable

1. Enter the value you want to store, which can be an expression.
2. Press [STO▸]. The store symbol (→) is displayed.
3. Type the variable name.

   Note: TI-89 Titanium users should use [alpha] as necessary when typing variable names.

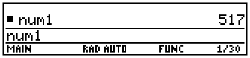


4. Press [ENTER].

To store to a variable temporarily, you can use the “*with*” operator. Refer to “Substituting Values and Setting Constraints” in *Symbolic Manipulation*.

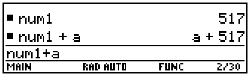
Displaying a Variable

1. Type the variable name.
2. Press [ENTER].



If the variable is undefined, the variable name is shown in the result.

In this example, the variable a is undefined. Therefore, it is used as a symbolic variable.

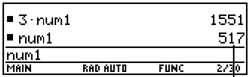


Note: Refer to *Symbolic Manipulation* for information about symbolic manipulation.

Using a Variable in an Expression

1. Type the variable name into the expression.
2. Press [ENTER] to evaluate the expression.

   Note: To view a list of existing variable names, use [2nd] [VAR-LINK] as described in *Memory and Variable Management*.

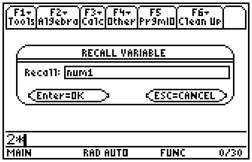


The variable's value did not change.

If you want the result to replace the variable's previous value, you must store the result.

In some cases, you may want to use a variable's actual value in an expression instead of the variable name.

1. Press [2nd] [RCL] to display a dialog box.
2. Type the variable name.
3. Press [ENTER] twice.

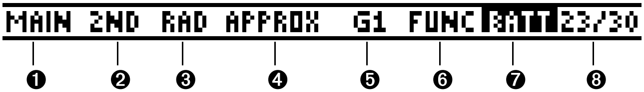


In this example, the value stored in **num1** will be inserted at the cursor position in the entry line.

Status Line Indicators in the Display

The status line is displayed at the bottom of all application screens. It shows information about the current state of the calculator, including several important mode settings.

Status Line Indicators



❶ Current Folder
❷ Modifier Key
❸ Angle Mode

❹ Exact/Approx Mode
❺ Graph Number
❻ Graph Mode
❼ Replace Batteries
❽ History Pairs, Busy/Pause, Locked Variable

Indicator	Meaning
Current Folder	Shows the name of the current folder. Refer to “Using Folders to Store Independent Sets of Variables” in *Calcul*ator Home Screen. MAIN is the default folder.
Modifier Key	Shows which modifier key is in effect, as described below.
2nd	[2nd] — will use the second function of the next key you press.
◆	[◆] — will use the diamond feature of the next key you press.
⬆	[⬆] — will type the uppercase letter for the next key you press. On the TI-89 Titanium, you can use [⬆] to type a letter without having to use [alpha].
a	[alpha] — will type the lowercase letter for the next key you press.
a	[2nd] [a-lock] — lowercase alpha-lock is on. Until you turn this off, will type the lowercase letter for each key you press. To cancel alpha-lock, press [alpha].
A	[⬆] [alpha] — uppercase ALPHA-lock is on. Until you turn this off, will type the uppercase letter for each key you press. To cancel ALPHA-lock, press [alpha].

Indicator	Meaning
Angle Mode	Shows the units in which angle values are interpreted and displayed. To change the Angle mode, use the [MODE] key.
RAD	Radians
DEG	Degrees
GRAD	Gradian
Exact/Approx Mode	Shows how answers are calculated and displayed. To change the Exact/Approx mode, use the [MODE] key.
AUTO	Auto
EXACT	Exact
APPROX	Approximate
Graph Number	If the screen is split to show two independent graphs, this indicates which graph is active — **G1** or **G2**.
Graph Mode	Indicates the type of graphs that can be plotted. To change the Graph mode, use the [MODE] key.
FUNC	**y(x)** functions
PAR	**x(t)** and **y(t)** parametric equations
POL	**r(θ)** polar equations
SEQ	**u(n)** sequences
3D	**z(x,y)** 3D equations
DE	**y'(t)** differential equations

Indicator	Meaning
Battery	Displayed only when the batteries are getting low. If BATT is shown with a black background, change the batteries as soon as possible.
History Pairs, Busy/Pause, Archived	The information shown in this part of the status line depends on the application you are using.
23/30	Displayed on the Home screen to show the number of entry/answer pairs in the history area. Refer to History Information on the Status Line in the *Calculator Home Screen* module.
BUSY	A calculation or graph is in progress.
PAUSE	You paused a graph or program.
	The variable opened in the current editor (Data/Matrix Editor, Program Editor, or Text Editor) is locked or archived and cannot be modified.

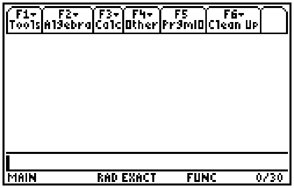
Notes:

- To cancel [2nd], [♦], [alpha], or [↑], press the same key again or press a different modifier key.
- If the next key you press does not have a diamond feature or an associated letter, the key performs its normal operation.

Calculator Home Screen

Calculator Home Screen

The calculator Home screen is the starting point for math operations, including executing instructions, evaluating expressions, and viewing results.



A blank calculator Home screen

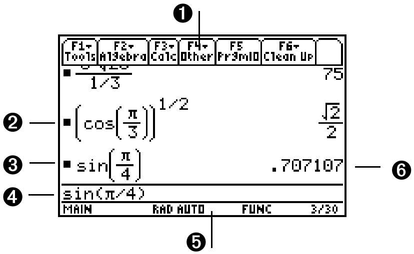
This module describes the parts of the calculator Home screen, how to scroll through or modify the history area; how to use cut, copy, and paste, and more.

Note: The term “calculator Home screen” is used in this module. Other modules use the term “Home screen.” Both terms are interchangeable and refer to the same screen.

Displaying the Calculator Home Screen

When you first turn on your TI-89 Titanium, the Apps desktop is displayed. To display the calculator Home screen, highlight the Home icon and press [ENTER]. You can also display the calculator Home screen by pressing [HOME] (TI-89 Titanium). If you turn off the Apps Desktop mode, the calculator Home screen appears automatically.

The following example contains previously entered data and describes the main parts of the calculator Home screen. Entry/answer pairs in the history area are displayed in “pretty print.” Pretty print displays expressions in the same form in which they are written on the board or in textbooks.



❶ Toolbar
Lets you display menus for selecting operations applicable to the calculator Home screen. To display a toolbar menu, press [F1], [F2], etc.

❷ Pretty Print Display
Shows exponents, roots, fractions, etc., in traditional form.

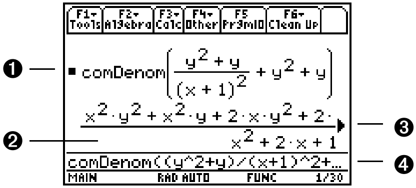
❸ Last Entry
Your last entry.

❹ Entry Line
Where you enter expressions or instructions.

❺ Status Line
Shows the current state of the calculator, including several important mode settings.

❻ Last Answer
Result of your last entry. Note that results are not displayed on the entry line. Note: [♦] [ENTER] (Approx) was used in this example.

The following example shows an answer that is not on the same line as the expression. Note that the answer is longer than the screen width. An arrow (▶) indicates the answer is continued. The entry line contains ellipsis (…). Ellipsis indicates the entry is longer than the screen width.



❶ Last Entry
"Pretty print" is ON. Exponents, roots, fractions, etc., are displayed in the same form in which they are traditionally written.

❷ History Area
Lists entry/answer pairs you have entered. Pairs scroll up the screen as you make new entries.

❸ Answer Continues
Highlight the answer and press ▶ to scroll right and view the rest of it. Note that the answer is not on the same line as the expression.

❹ Expression Continues (…)
Press ▶ to scroll right and view the rest of the entry. Press [2nd] ◀ or [2nd] ▶ to go to the beginning or end of the entry line.

History Area

The history area shows up to eight previous entry/answer pairs (depending on the complexity and height of the displayed expressions). When the display is filled, information scrolls off the top of the screen. You can use the history area to:

- Review previous entries and answers. You can use the cursor to view entries and answers that have scrolled off the screen.
- Recall or auto-paste a previous entry or answer onto the entry line so that you can re-use or edit it.

Scrolling through the History Area

Normally, the cursor is in the entry line. However, you can move the cursor into the history area.

To:	Do this:
View entries or answers that have scrolled off the screen	• From the entry line, press ▲ to highlight the last answer. • Continue using ▲ to move the cursor from answer to entry, up through the history area.
Go to the oldest or newest history pair	If the cursor is in the history area, press [♦] ▲ or [♦] ▼, respectively.
View an entry or answer that is too long for one line (▶ is at end of line)	Move the cursor to the entry or answer. Use ◀ and ▶ to scroll left and right (or [2nd] ◀ and [2nd] ▶ to go to the beginning or end), respectively.

To:	Do this:
Return the cursor to the entry line	Press [ESC], or press ▼ until the cursor is back on the entry line.

Note: An example of viewing a long answer is available.

History Information on the Status Line

Use the history indicator on the status line for information about the entry/answer pairs. For example:

If the cursor is on the entry line:	Total number of pairs that are currently saved.	**8/30**	Maximum number of pairs that can be saved.
If the cursor is in the history area:	Pair number of the highlighted entry or answer.		Total number of pairs that are currently saved.

By default, the last 30 entry/answer pairs are saved. If the history area is full when you make a new entry (indicated by 30/30), the new entry/answer pair is saved and the oldest pair is deleted. The history indicator does not change.

To:	Do this:
Change the number of pairs that can be saved	Press [F1] and select **9:Format**, or press [♦] [|] Then press ▶, use ▲ or ▼ to highlight the new number, and press [ENTER] twice.
Clear the history area and delete all saved pairs	Press [F1] and select **8:Clear Home**, or enter **ClrHome** on the entry line.
Delete a particular entry/answer pair	Move the cursor to either the entry or answer. Press [←] or [CLEAR].

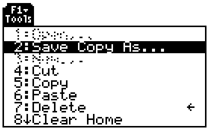
Saving the Calculator Home Screen Entries as a Text Editor Script

To save all the entries in the history area, you can save the calculator Home screen to a text variable. When you want to reexecute those entries, use the Text Editor to open the variable as a command script.

Saving the Entries in the History Area

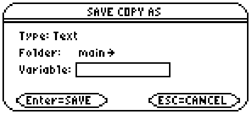
From the calculator Home screen:

1. Press [F1] and select **2:Save Copy As.**



2. Specify a folder and text variable that you want to use to store the entries.

   Note: Only the entries are saved, not the answers.



Item	Description
Type	Automatically set as Text and cannot be changed.
Folder	Shows the folder in which the text variable will be stored. To use a different folder, press ▶ to display a menu of existing folders. Then select a folder.
Variable	Type a valid, unused variable name.

Note: For information about folders, see the *Memory and Variable Management module*.

3. Press [ENTER] (after typing in an input box such as Variable, press [ENTER] twice).

Restoring the Saved Entries

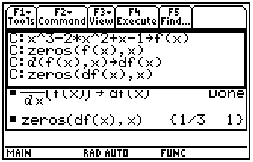
Because the entries are stored in a script format, you cannot restore them from the calculator Home screen. (On the calculator Home screen's [F1] toolbar menu, **1:Open** is not available.) Instead:

1. Use the Text Editor to open the variable containing the saved calculator Home screen entries.

   The saved entries are listed as a series of command lines that you can execute individually, in any order.

2. Starting with the cursor on the first line of the script, press [F4] repeatedly to execute the commands line by line.

3. Display the restored calculator Home screen.



   This split screen shows the Text Editor (with the command line script) and the restored calculator Home screen.

Note: For complete information on using the Text Editor and executing a command script, refer to the *Text Editor* module.

Cutting, Copying, and Pasting Information

Cut, copy, and paste operations let you move or copy information within the same application or between different applications. These operations use the clipboard, which is an area in memory that serves as a temporary storage location.

Auto-paste vs. Cut/Copy/Paste

Auto-paste is a quick way to copy an entry or answer in the history area and paste it to the entry line.

1. Use ⊕ and ⊖ to highlight the item in the history area.
2. Press [ENTER] to auto-paste that item to the entry line.

To copy or move information in the entry line, you must use a cut, copy, or paste operation. (You can perform a copy operation in the history area, but not a cut or paste.)

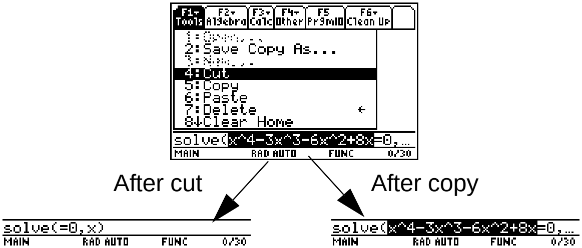
Cutting or Copying Information to the Clipboard

When you cut or copy information, that information is placed in the clipboard. However, cutting deletes the information from its current location (used to move information) and copying leaves the information.

1. Highlight the characters that you want to cut or copy.

   In the entry line, move the cursor to either side of the characters. Hold [↑] and press ◀ or ▶ to highlight characters to the left or right of the cursor, respectively.
2. Press [F1] and select **4:Cut** or **5:Copy**.

Clipboard = (empty or the previous contents)



Clipboard = x^4–3x^3–6x^2+8x

Clipboard = x^4–3x^3–6x^2+8x

Note: You can cut, copy or paste without having to use the [F1] toolbar menu. Press:
[♦] [CUT], [♦] [COPY], or [♦] [PASTE]

Cutting is not the same as deleting. When you delete information, it is not placed in the clipboard and cannot be retrieved.

Note: When you cut or copy information, it replaces the clipboard's previous contents, if any.

Pasting Information from the Clipboard

A paste operation inserts the contents of the clipboard at the current cursor location on the entry line. This does not change the contents of the clipboard.

1. Position the cursor where you want to paste the information.

2. Press F1 and select **6:Paste**, or use the key shortcut:
◆ [PASTE]

Example: Copying and Pasting

Suppose you want to reuse an expression without retyping it each time.

1. Copy the applicable information.

   a) Use ↑ ▶ or ↑ ◀ to highlight the expression.

   b) Press:
   ◆ [COPY]

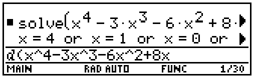
   c) For this example, press ENTER to evaluate the entry.

2. Paste the copied information into a new entry.

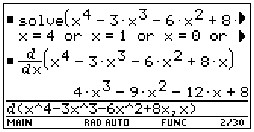
   a) Begin a new entry and place the cursor where you want to paste the copied information.

b) Press [F3] **1** to select the *d* (differentiate) function.

c) Press:
[♦] [PASTE]
to paste the copied expression.



d) Complete the new entry, and press [ENTER].



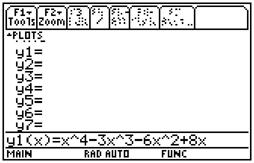
Note: You can also reuse an expression by creating a user-defined function.

3. Paste the copied information into a different application.

a) Press [♦] [Y=] to display the Y= Editor.

b) Press [ENTER] to define **y1(x)**.

c) Press:
[♦] [PASTE]
to paste.



d) Press [ENTER] to save the new definition.

Note: By copying and pasting, you can easily transfer information from one application to another.

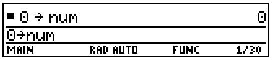
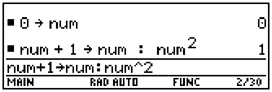
Reusing a Previous Entry or the Last Answer

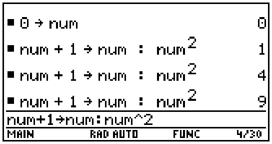
You can reuse a previous entry by reexecuting the entry “as is” or by editing the entry and then reexecuting it. You can also reuse the last calculated answer by inserting it into a new expression.

Reusing the Expression on the Entry Line

When you press [ENTER] to evaluate an expression, the TI-89 Titanium leaves that expression on the entry line and highlights it. You can type over the entry, or you can reuse it as necessary.

For example, using a variable, find the square of 1, 2, 3, etc. As shown below, set the initial variable value and then enter the variable expression. Next, reenter to increment the variable and calculate the square.

TI-89 Titanium	Display
0 [STO▸] [2nd] [a-lock] NUM [ENTER]	
NUM [alpha] [+] 1 [STO▸] [2nd] [a-lock] NUM [2nd] [:] NUM [^] 2 [ENTER]	

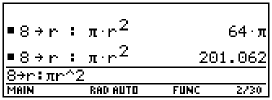
TI-89 Titanium	Display
[ENTER] [ENTER]	

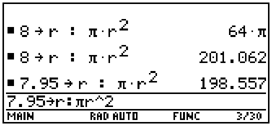
Note: Reexecuting an entry “as is” is useful for iterative calculations that involve variables.

Using the equation **A=πr**^{2}, use trial and error to find the radius of a circle that covers 200 square centimeters.

Note: Editing an entry lets you make minor changes without retyping the entire entry.

The example below uses 8 as the first guess and then displays the answer in its approximate floating-point form. You can edit and reexecute using `7.95` and continue until the answer is as accurate as you want.

TI-89 Titanium	Display
8 [STO▶] [alpha] R [2nd] [:] [2nd] [π] [alpha] R [^] 2 [ENTER]	
[♦] [ENTER]	

TI-89 Titanium	Display
◀ [◆] [DEL] 7.95 [ENTER]	

Note: When the entry contains a decimal point, the result is automatically displayed in floating-point.

Recalling a Previous Entry

You can recall any previous entry that is stored in the history area, even if the entry has scrolled off the top of the screen. The recalled entry *replaces* whatever is currently shown on the entry line. You can then reexecute or edit the recalled entry.

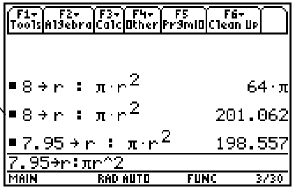
To recall:	Press:	Effect:
The last entry (if you've changed the entry line)	[2nd] [ENTRY] once	If the last entry is still shown on the entry line, this recalls the entry prior to that.
Previous entries	[2nd] [ENTRY] repeatedly	Each press recalls the entry prior to the one shown on the entry line.

Note: You can also use the entry function to recall any previous entry. Refer to **entry( )** in the *Technical Reference* module.

For example:

If the entry line contains the last entry, [2nd] [ENTRY] recalls this entry.

If the entry line is edited or cleared, [2nd] [ENTRY] recalls this entry.



Recalling the Last Answer

Each time you evaluate an expression, the TI-89 Titanium stores the answer to the variable **ans(1)**. To insert this variable in the entry line, press [2nd] [ANS].

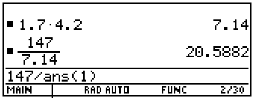
For example, calculate the area of a garden plot that is 1.7 meters by 4.2 meters. Then calculate the yield per square meter if the plot produces a total of 147 tomatoes.

1. Find the area.

   1.7 [×] 4.2 [ENTER]

2. Find the yield.

   147 [÷] [2nd] [ANS] [ENTER]



Variable **ans(1)** is inserted, and its value is used in the calculation.

Just as **ans(1)** always contains the last answer, **ans(2)**, **ans(3)**, etc., also contain previous answers. For example, **ans(2)** contains the next-to-last answer.

Note: Refer to **ans( )** in the *Technical Reference* module.

Auto-Pasting an Entry or Answer from the History Area

You can select any entry or answer from the history area and “auto-paste” a duplicate of it on the entry line. This lets you insert a previous entry or answer into a new expression without having to retype the previous information.

Why Use Auto-Paste

The effect of using auto-paste is similar to [2nd] [ENTRY] and [2nd] [ANS] as described in the previous section, but there are differences.

For entries:

Pasting lets you:	[2nd] [ENTRY] lets you:
Insert any previous entry into the entry line.	Replace the contents of the entry line with any previous entry.

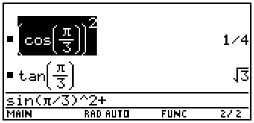
For answers:

Pasting lets you:	[2nd] [ANS] lets you:
Insert the displayed value of any previous answer into the entry line.	Insert the variable **ans(1)**, which contains the last answer only. Each time you enter a calculation, **ans(1)** is updated to the latest answer.

Note: You can also paste information by using the [F1] toolbar menu.

1. On the entry line, place the cursor where you want to insert the entry or answer.
2. Press ↑ to move the cursor up into the history area. This highlights the last answer.
3. Use ↑ and ↓ to highlight the entry or answer to auto-paste.

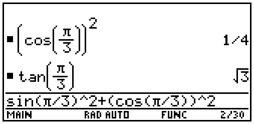
   - ↑ moves from answer to entry up through the history area.
   - You can use ↑ to highlight items that have scrolled off the screen



   Note: To cancel auto-paste and return to the entry line, press [ESC]. To view an entry or answer too long for one line (indicated by ▶ at the end of the line), use → and ← or [2nd] → and [2nd] ←.

4. Press [ENTER].

   The highlighted item is inserted in the entry line.



This pastes the entire entry or answer. If you need only a part of the entry or answer, edit the entry line to delete the unwanted parts.

Creating and Evaluating User-Defined Functions

User-defined functions can be a great time-saver when you need to repeat the same expression (but with different values) multiple times. User-defined functions can also extend your TI-89 Titanium's capabilities beyond the built-in functions.

Format of a Function

The following examples show user-defined functions with one argument and two arguments. You can use as many arguments as necessary. In these examples, the definition consists of a single expression (or statement).

cube(x) = x^3 xroot(x,y) = $y^{1/x}$
➊ ➋ ➌   ➊ ➋ ➌

➊ Function name
➋ Argument list
➌ Definition

When defining functions and programs, use unique names for arguments that will not be used in the arguments for a subsequent function or program call.

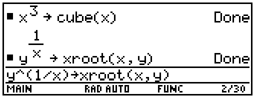
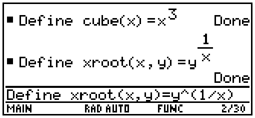
Note: Function names follow the same rules as variable names. Refer to "Storing and Recalling Variable Values" in *Operating the Calculator*.

In the argument list, be sure to use the same arguments that are used in the definition. For example, **cube(n) =** $\mathbf{x^3}$ gives unexpected results when you evaluate the function.

Arguments (x and y in these examples) are placeholders that represent whatever values you pass to the function. They do not represent the variables x and y unless you specifically pass x and y as the arguments when you evaluate the function.

Creating a User-Defined Function

Use one of the following methods.

Method	Description
STO▶	Store an expression to a function name (including the argument list). 
Define command	Define a function name (including the argument list) as an expression. 
Program Editor	Refer to *Programming* or information on creating a user-defined function.

You can also create a user-defined function whose definition consists of multiple statements. The definition can include many of the control and decision-making structures (**If**, **ElseIf**, **Return**, etc.) used in programming.

Note: For information about similarities and differences between functions and programs, refer to *Programming*.

For example, suppose you want to create a function that sums a series of reciprocals based on an entered integer (**n**):

$$\frac{1}{n} + \frac{1}{n-1} + \ldots + \frac{1}{1}$$

When creating the definition of a multi-statement function, it may be helpful to visualize it first in a block form.

```
❶ Func
❷ Local temp,i
  If fPart(nn)≠0 or nn≤0
❸   Return "bad argument"
  0→temp
❹ For i,nn,1,-1
    approx(temp+1/i)→temp
  EndFor
❺ Return temp
❶ EndFunc
```

❶ **Func** and **EndFunc** must begin and end the function.

❷ Variables not in the argument list must be declared as local.

❸ Returns a message if nn is not an integer or if nn≤0.
❹ Sums the reciprocals.
❺ Returns the sum.

When entering a multi-statement function on the calculator Home screen, you must enter the entire function on a single line. Use the **Define** command just as you would for a single-statement function.

Use a colon to separate each statement.

```
Define sumrecip(nn)=Func:Local temp,i: ... :EndFunc
```

Use argument names that will never be used when calling the function or program.

On the calculator Home screen:

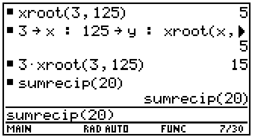
Multi-statement functions show as

Enter a multi-statement function on one line. Be sure to include colons.

Note: It’s easier to create a complicated multi-statement function in the Program Editor than on the calculator Home screen. Refer to *Programming*.

Evaluating a Function

You can use a user-defined function just as you would any other function. Evaluate it by itself or include it in another expression.



Displaying and Editing a Function Definition

To:	Do this:
Display a list of all user-defined functions	Press [2nd] [VAR-LINK] to display the VAR-LINK screen. You may need to use the [F2] **View** toolbar menu to specify the **Function** variable type. (Refer to *Memory and Variable Management*.) – or – Press: [CATALOG] [F4]
Display a list of Flash application functions	Press: [CATALOG] [F3]

To:	Do this:
Display the definition of a user-defined function	From the VAR-LINK screen, highlight the function and display the Contents menu. [2nd] [F6] – or – From the calculator Home screen, press [2nd] [RCL]. Type the function name but not the argument list (such as **xroot**), and press [ENTER] twice. – or – From the Program Editor, open the function. (Refer to *Programming*.)
Edit the definition	From the calculator Home screen, use [2nd] [RCL] to display the definition. Edit the definition as necessary. Then use [STO▸] or **Define** to save the new definition. – or – From the Program Editor, open the function, edit it, and save your changes. (Refer to *Programming*.)

Note: You can view a user-defined function in the CATALOG dialog box, but you cannot use the CATALOG to view or edit its definition.

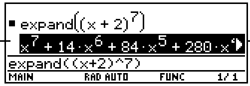
If an Entry or Answer Is “Too Big”

In some cases, an entry or answer may be “too long” and/or “too tall” to be displayed completely in the history area. In other cases, the TI-89 Titanium may not be able to display an answer because there is not enough free memory.

Move the cursor into the history area, and highlight the entry or answer. Then use the cursor pad to scroll. For example:

- The following shows an answer that is too long for one line.

Press ◀ or [2nd] ◀ to scroll left.



Press ▶ or [2nd] ▶ to scroll right.

- The following shows an answer that is both too long and too tall to be displayed on the screen.

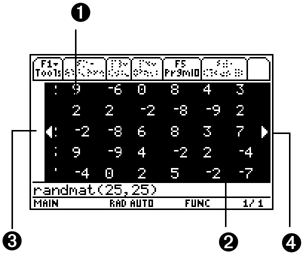
  Note: This example uses the **randMat** function to generate a 25 x 25 matrix.

❶ Press ▲ or [↑] ▲ to scroll up

❷ Press [↑] ▼ to scroll down

❸ Press ◀ or [2nd] ◀ to scroll left

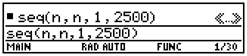
❹ Press ▶ or [2nd] ▶ to scroll right



If There Is not Enough Memory

A << ...>> symbol is displayed when the TI-89 Titanium does not have enough free memory to display the answer.

For example:



Note: This example uses the seq function to generate a sequential list of integers from 1 to 2500.

When you see the << ...>> symbol, the answer cannot be displayed even if you highlight it and try to scroll.

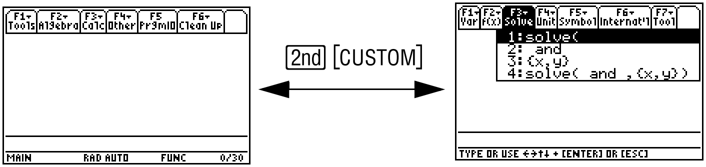
In general, you can try to:

- Free up additional memory by deleting unneeded variables and/or Flash applications. Use [2nd] [VAR-LINK] as described in *Memory and Variable Management*.
- If possible, break the problem into smaller parts that can be calculated and displayed with less memory.

Using the Custom Menu

The TI-89 Titanium has a custom menu that you can turn on and off at any time. You can use the default custom menu or create your own as described in the *Programming* module.

When you turn on the custom menu, it replaces the normal toolbar menu. When you turn it off, the normal menu returns. For example, from the calculator Home screen’s normal toolbar menu, press [2nd] [CATALOG] to toggle the custom menu on and off.



Calculator Home screen normal toolbar menu

Custom menu

Note: You can also turn the custom menu on and off by entering **CustmOn** or **CustmOff** in the entry line and pressing [ENTER].

Unless the menu has been modified, the default custom menu appears.

Menu	Function
[F1] **Var**	Common variable names.
[F2] **f(x)**	Function names such as f(x), g(x), and f(x,y).
[F3] **Solve**	Items related to solving equations.
[F4] **Unit**	Common units such as _m, _ft, and _l.
[F5] **Symbol**	Symbols such as #, ?, and ~.

Menu	Function
International [2nd] [F6]	Commonly accented characters such as è, é,and ê.
Tool [2nd] [F7]	**ClrHome**, **NewProb**, and **CustmOff**.

Note: A custom menu can give you quick access to commonly used items. The *Programming* module shows you how to create custom menus for the items you use most often.

Restoring the Default Custom Menu

If a custom menu other than the default is displayed and you want to restore the default:

1. From the calculator Home screen, use [2nd] [CATALOG] to turn off the custom menu and display the calculator Home screen’s normal toolbar menu.

2. Display the **Clean Up** toolbar menu, and select **3:Restore custom default**.
   [2nd] [F6]

   This pastes the commands used to create the default menu into the entry line.

   Note: The previous custom menu is erased. If that menu was created with a program, it can be recreated later by running the program again.

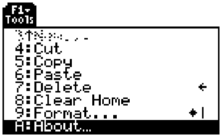
3. Press [ENTER] to execute the commands and restore the default.

Finding the Software Version and ID Number

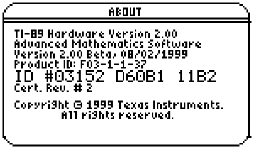
In some situations, you may need to find out information about your TI-89 Titanium, particularly the software version and the unit's ID number.

Displaying the "About" Screen

1. From either the calculator Home screen or the Apps desktop, press [F1] and then select **A:About**.



   Your screen will be different from the one shown to the right.



2. Press [ENTER] or [ESC] to close the screen.

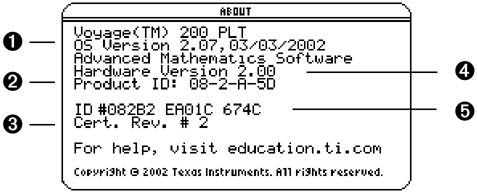
When Do You Need this Information?

The information on the About screen is intended for situations such as:

- If you obtain new or upgraded software or Flash applications for your TI-89 Titanium, you may need to provide your current software version and/or the ID number of your unit.
- If you have difficulties with your TI-89 Titanium and need to contact technical support, knowing the software version may make it easier to diagnose the problem.

The About screen displays the following information about your calculator:

- Hardware version
- OS (Advanced Mathematics Software) version
- Product identifier (Product ID)
- Unit ID
- Apps certificate revision number (Cert. Rev.)



❶ OS version
❷ Product identifier
❸ Apps certificate revision number
❹ Hardware version
❺ Unit ID (required to obtain certificates for installing purchased Apps)

Your screen will be different from the one shown above.

Symbolic Manipulation

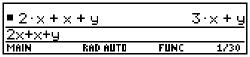
Using Undefined or Defined Variables

When performing algebraic or calculus operations, it is important that you understand the effect of using undefined and defined variables. Otherwise, you may get a number for a result instead of the algebraic expression that you anticipated.

How Undefined and Defined Variables Are Treated

When you enter an expression that contains a variable, the TI-89 Titanium treats the variable in one of two ways.

- If the variable is undefined, it is treated as an algebraic symbol.



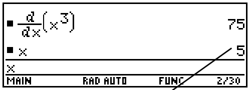
- If the variable is defined (even if defined as 0), its value replaces the variable.

To see why this is important, suppose you want to find the first derivative of x^3 with respect to x.

- If x is undefined, the result is in the form you probably expected.

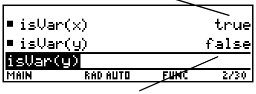
- If *x* is defined, the result may be in a form you did not expect.

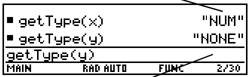
  Note: When defining a variable, it's a good practice to use more than one character in the name. Leave one-character names undefined for symbolic calculations.



Unless you knew that 5 had been stored to x previously, the answer 75 could be misleading.

Determining If a Variable Is Exists

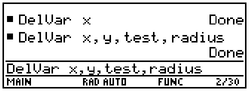
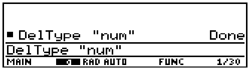
Method:	Example:
Enter the variable name.	If defined, the variable's value is displayed.  If undefined, the variable name is displayed.
Use the **isVar()** function.	If defined, "true" is displayed.  If undefined, "false" is displayed.

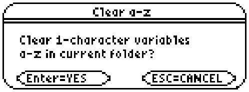
Method:	Example:
Use the **getType** function.	If defined, the variable's type is displayed.  If undefined, "NONE" is displayed.

Note: Use [2nd] [VAR-LINK] to view a list of defined variables, as described in *Memory and Variable Management*.

Deleting a Defined Variable

You can “undefine” a defined variable by deleting it.

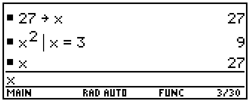
To delete:	Do this:
One or more specified variables	Use the **DelVar** function.  You can also delete variables by using the VAR-LINK screen ([2nd] [VAR-LINK]) as described in *Memory and Variable Management*.
All variables of a specific type **Note:** The Deltype function deletes all variables of the specified type in all folders.	Use the **Deltype** function. 

To delete:	Do this:
All one-letter variables (a – z) in the current folder. **Note:** For information about folders, refer to the *Calculator Home Screen* module.	From the Home screen Clean Up menu, select **1:Clear a-z**. You will be prompted to press [ENTER] to confirm the deletion. 

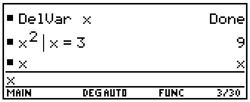
Temporarily Overriding a Variable

By using the **“***with***”** operator (**|**), you can:

- Temporarily override a variable’s defined value.



- Temporarily define a value for an undefined variable.



Note: For more information about the **|** operator, refer to Typing the “*With*” Operator.

To type the **“***with***”** operator (**|**), press:

[|]

Using Exact, Approximate, and Auto Modes

The Exact/Approx mode settings, which are described briefly in *Operating the Handheld*, directly affect the precision and accuracy with which the TI-89 Titanium calculates a result. This section describes these mode settings as they relate to symbolic manipulation.

EXACT Setting

When Exact/Approx = EXACT, the handheld uses exact rational arithmetic with up to 614 digits in the numerator and 614 digits in the denominator. The EXACT setting:

- Transforms irrational numbers to standard forms as much as possible without approximating them. For example, $\sqrt{12}$ transforms to $2\sqrt{3}$ and **ln(1000)** transforms to **3 ln(10)**.
- Converts floating-point numbers to rational numbers. For example, 0.25 transforms to 1/4.

The functions **solve**, **cSolve**, **zeros**, **cZeros**, **factor**, ∫, **fMin**, and **fMax** use only exact symbolic algorithms. These functions do not compute approximate solutions in the EXACT setting.

- Some equations, such as $2^{-x} = x$, have solutions that cannot all be finitely represented in terms of the functions and operators on the handheld.

- With this kind of equation, EXACT will not compute approximate solutions. For example, $\mathbf{2^{-x} = x}$ has an approximate solution $\mathbf{x \approx 0.641186}$, but it is not displayed in the EXACT setting.

Advantages	Disadvantages
Results are exact.	As you use more complicated rational numbers and irrational constants, calculations can: • Use more memory, which may exhaust the memory before a solution is completed. • Take more computing time. • Produce bulky results that are harder to comprehend than a floating-point number.

APPROXIMATE Setting

When Exact/Approx = APPROXIMATE, the handheld converts rational numbers and irrational constants to floating-point. However, there are exceptions:

- Certain built-in functions that expect one of their arguments to be an integer will convert that number to an integer if possible. For example: *d*(**y(x), x, 2.0)** transforms to *d*(**y(x), x, 2).**
- Whole-number floating-point exponents are converted to integers. For example: $x^{2.0}$ transforms to x^2 even in the APPROXIMATE setting.

Functions such as **solve** and ∫ (integrate) can use both exact symbolic and approximate numeric techniques. These functions skip all or some of their exact symbolic techniques in the APPROXIMATE setting.

Advantages	**Disadvantages**
If exact results are not needed, this might save time and/or use less memory than the EXACT setting. Approximate results are sometimes more compact and comprehensible than exact results.	Results with undefined variables or functions often exhibit incomplete cancellation. For example, a coefficient that should be **0** might be displayed as a small magnitude such as **1.23457E−11**.
If you do not plan to use symbolic computations, approximate results are similar to familiar, traditional numeric calculators.	Symbolic operations such as limits and integration are less likely to give satisfying results in the APPROXIMATE setting. Approximate results are sometimes less compact and comprehensible than exact results. For example, you may prefer to see **1/7** instead of **.142857**.

AUTO Setting

When Exact/Approx = AUTO, the handheld uses exact rational arithmetic wherever all of the operands are rational numbers. Otherwise, floating-point arithmetic is used after converting any rational operands to floating-point. In other words, floating-point is "infectious." For example:

1/2 - 1/3 transforms to **1/6**
but
0.5 - 1/3 transforms to **.16666666666667**

This floating-point infection does not leap over barriers such as undefined variables or between elements of lists or matrices. For example:

(1/2 - 1/3) x + (0.5 - 1/3) y transforms to **x/6 + .16666666666667 y**
and
{1/2 - 1/3, 0.5 - 1/3} transforms to **{1/6, .16666666666667}**

In the AUTO setting, functions such as **solve** determine as many solutions as possible exactly, and then use approximate numerical methods if necessary to determine additional solutions. Similarly,] (integrate) uses approximate numerical methods if appropriate where exact symbolic methods fail.

Advantages	Disadvantages
You see exact results when practical, and approximate numeric results when exact results are impractical.	If you are interested only in exact results, some time may be wasted seeking approximate results.
You can often control the format of a result by choosing to enter some coefficients as either rational or floating-point numbers.	If you are interested only in approximate results, some time may be wasted seeking exact results. Moreover, you might exhaust the memory seeking those exact results.

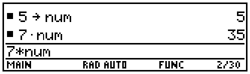
Automatic Simplification

When you type an expression on the entry line and press [ENTER], the TI-89 Titanium automatically simplifies the expression according to its default simplification rules.

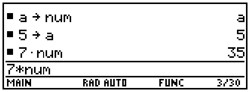
Default Simplification Rules

All of the following rules are applied automatically. You do not see intermediate results.

- If a variable has a defined value, that value replaces the variable.



  If the variable is defined in terms of another variable, the variable is replaced with its “lowest level” value (called infinite lookup).

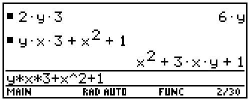


  Default simplification does not modify variables that use path names to indicate a folder. For example, *x+class\x* does not simplify to 2x.

Note: For information about folders, refer to the *Calculator Home Screen* module.

- For functions:
  - The arguments are simplified. (Some built-in functions delay simplification of some of their arguments.)
  - If the function is a built-in or user-defined function, the function definition is applied to the simplified arguments. Then the functional form is replaced with this result.

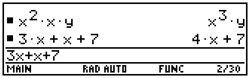
- Numeric subexpressions are combined.
- Products and sums are sorted into order.



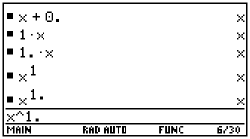
Products and sums involving undefined variables are sorted according to the first letter of the variable name.

- Undefined variables *r* through *z* are assumed to be true variables, and are placed in alphabetical order at the beginning of a sum.
- Undefined variables *a* through *q* are assumed to represent constants, and are placed in alphabetical order at the end of a sum (but before numbers).

- Similar factors and similar terms are collected.

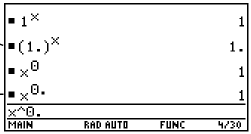


- Identities involving zeros and ones are exploited.

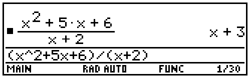


This floating-point number causes numeric results to be shown as floating-point.

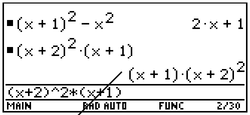
If a floating-point whole number is entered as an exponent, it is treated as an integer (and does not produce a floating-point result).



- Polynomial greatest common divisors are canceled.

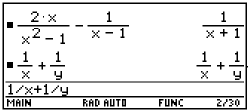


- Polynomials are expanded unless no key cancellation can occur.



No key cancellation

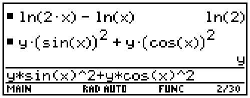
- Common denominators are formed unless no key cancellation can occur.



No key cancellation

- Functional identities are exploited. For example:

  ln(2x) = ln(2) + ln(x)
  and
  sin(x)^2 + cos(x)^2 = 1



How Long Is the Simplification Process?

Depending on the complexity of an entry, result, or intermediate expression, it can take a long time to expand an expression and cancel common divisors as necessary for simplification.

To interrupt a simplification process that is taking too long, press ON. You can then try simplifying only a portion of the expression. (Auto-paste the entire expression on the entry line, and then delete the unwanted parts.)

Delayed Simplification for Certain Built-In Functions

Usually, variables are automatically simplified to their lowest possible level before they are passed to a function. For certain functions, however, complete simplification is delayed until after the function is performed.

Functions that Use Delayed Simplification

Functions that use delayed simplification have a required var argument that performs the function with respect to a variable. These functions have at least two arguments with the general form:

function(*expression, var* [, …]**)**

Note: Not all functions that use a var argument use delayed simplification.

For example: **solve(x^2−x−2=0,x)**
*d***(x^2−x−2,x)**
∫(x^2−x−2,x)
limit(x^2**−x−2,x,5)**

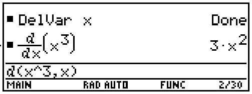
For a function that uses delayed simplification:

1. The *var* variable is simplified to the lowest level at which it remains a variable (even if it could be further simplified to a non-variable value).
2. The function is performed using the variable.
3. If *var* can be further simplified, that value is then substituted into the result.

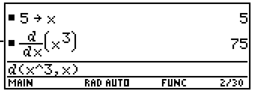
Note: You may or may not want to define a numeric value for *var*, depending on the situation.

For example:

x cannot be simplified.

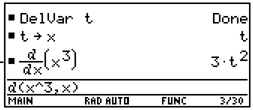


x is not simplified. The function uses x^3, and then substitutes 5 for x.

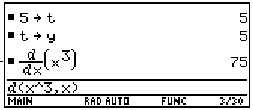


Note: The example to the right finds the derivative of x^3 at x=5. If x^3 was initially simplified to 75, you would find the derivative of 75, which is not what you want.

x is simplified to t. The function uses t^3.



x is simplified to t. The function uses t^3, and then substitutes 5 for t.



Substituting Values and Setting Constraints

The **“***with***”** operator (**|**) lets you temporarily substitute values into an expression or specify domain constraints.

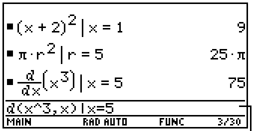
Typing the “With” Operator

To type the **“***with***”** operator (**|**), press:

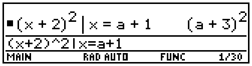
|

Substituting for a Variable

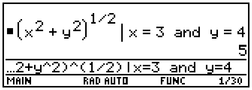
For every occurrence of a specified variable, you can substitute a numeric value or an expression.



First derivative of x^3 at x = 5

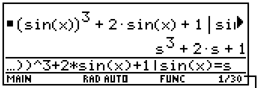


To substitute for multiple variables at the same time, use the Boolean *and* operator.



Substituting for a Simple Expression

For every occurrence of a simple expression, you can substitute a variable, numeric value, or another expression.



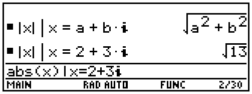
Substituting s for sin(x) shows that the expression is a polynomial in terms of sin(x).

By replacing a commonly used (or long) term, you can display results in a more compact form.

Note: acos(x) is different from **a*cos(x)**.

Substituting Complex Values

You can substitute complex values just as you would for other values.



All undefined variables are treated as real numbers in symbolic calculations. To perform complex symbolic analysis, you must define a complex variable. For example:

x+y*i*→z

Then you can use *z* as a complex variable. You can also use *z_*. For more information see the _ (underscore) topic in the *Technical Reference* module.

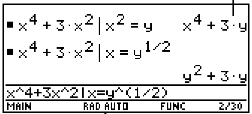
Note:

- For an overview of complex numbers, refer to the *Technical Reference* module.
- To get the complex i, press [2nd] [i]. Do not simply type the letter **i** on the keyboard.

Be Aware of the Limitations of Substitutions

- Substitution occurs only where there is an exact match for the substitution.

Only x^2 was replaced, not x^4.



Define the substitution in simpler terms for a more complete substitution.

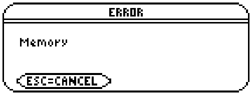
- Infinite recursions can occur when you define a substitution variable in terms of itself.

  Substitutes sin(x+1), sin(x+1+1), sin(x+1+1+1), etc

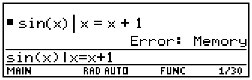
  sin(x)|x=x+1

  When you enter a substitution that causes an infinite recursion:

  - An error message is displayed.

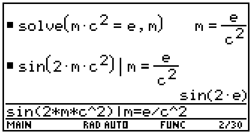


  - When you press [ESC], an error is shown in the history area.



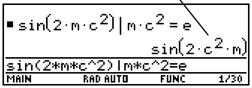
- Internally, an expression is sorted according to the automatic simplification rules. Therefore, products and sums may not match the order in which you entered them.

  - As a general rule, you should substitute for a single variable.



- Substituting for more general expressions (either **m·c²=e** or **c²·m=e**) may not work as you anticipate.

No match for substitution



Note: Use the **solve** function to help determine the single-variable substitution.

Specifying Domain Constraints

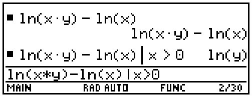
Many identities and transformations are valid for only a particular domain. For example:

ln(x∗y) = ln(x) + ln(y)	only if **x** and/or **y** is not negative
Sin⁻¹(sin(θ)) = θ	only if θ ≥ ⁻π**/2** and θ ≤ π**/2** radians

Use the **“***with***”** operator to specify the domain constraint.

Because ln(x∗y) = ln(x) + ln(y) is not always valid, the logarithms are not combined.

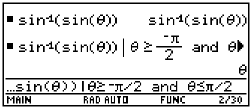
With a constraint, the identity is valid and the expression is simplified.



Note: Enter **ln(x∗y)** instead of **ln(xy)**; otherwise, xy is interpreted as a single variable named xy.

Because $\sin^{-1}(\sin(\theta)) = \theta$ is not always valid, the expression is not simplified.

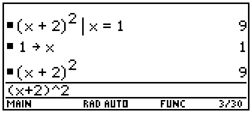
With a constraint, the expression can be simplified.



Note: For ≥ or ≤, press ◆ [>] or ◆ [<]. You can also use [2nd] [MATH] **8** or [2nd] [CHAR] **2** to select them from a menu.

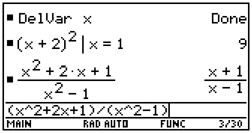
Using Substitutions vs. Defining a Variable

In many cases, you can achieve the same effect as a substitution by defining the variable.

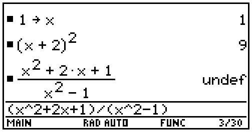


However, substitution is preferable for most cases because the variable is defined only for the current calculation and does not accidentally affect later calculations.

Substituting x=1 does not affect the next calculation.



Storing 1→x affects the subsequent calculations.calculation.



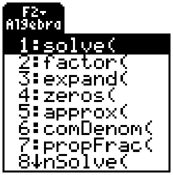
Caution: After x is defined, it can affect all calculations that involve x (until you delete x).

Overview of the Algebra Menu

You can use the F2 **Algebra** toolbar menu to select the most commonly used algebraic functions.

The Algebra Menu

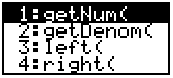
From the Home screen, press F2 to display:



This menu is also available from the **MATH** menu. Press 2nd [MATH] and then select **9:Algebra**.

Note: For a complete description of each function and its syntax, refer to the *Technical Reference* module.

Menu Item	Description
solve	Solves an equation for a specified variable. This returns real solutions only, regardless of the **Complex Format** mode setting. Displays answers with "*and*" and "*or*" connecting solutions. (For complex solutions, select **A:Complex** from the **Algebra** menu.)
factor	Factors an expression with respect to all its variables or with respect to only a specified variable.
expand	Expands an expression with respect to all its variables or with respect to only a specified variable.
zeros	Determines the values of a specified variable that make an expression equal to zero. Displays in a list.
approx	Evaluates an expression using floating-point arithmetic, where possible. This is equivalent to using [MODE] to set Exact/Approx = APPROXIMATE (or using [♦] [ENTER] to evaluate an expression).
comDenom	Calculates a common denominator for all terms in an expression and transforms the expression into a reduced ratio of a numerator and denominator.
propFrac	Returns an expression as a proper fraction expression.
nSolve	Calculates a single solution for an equation as a floating-point number (as opposed to solve, which may display several solutions in a rational or symbolic form).

Menu Item	Description
Trig	Displays the submenu: 
	tExpand — Expands trig expressions with angle sums and multiple angles.
	TCollect — Collects the products of integer powers of trig functions into angle sums and multiple angles. **tCollect** is the opposite of **tExpand**.
Complex	Displays the submenu:  These are the same as **solve**, **factor**, and **zeros**; but they also compute complex results.
Extract	Displays the submenu: 
	getNum — Applies **comDenom** and then returns the resulting numerator.
	getDenom — Applies **comDenom** and then returns the resulting denominator.
	left — Returns the left-hand side of an equation or inequality.

Menu Item	Description
	right — Returns the right-hand side of an equation or inequality.

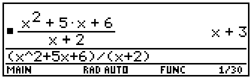
Note: The **left** and **right** functions are also used to return a specified number of elements or characters from the left or right side of a list or character string.

Common Algebraic Operations

This section gives examples for some of the functions available from the [F2] **Algebra** toolbar menu. For complete information about any function, refer to the *Technical Reference* module. Some algebraic operations do not require a special function.

Adding or Dividing Polynomials

You can add or divide polynomials directly, without using a special function.



Factoring and Expanding Polynomials

Use the **factor** ([F2] **2**) and **expand** ([F2] **3**) functions.

factor(*expression* [,*var*]**)**

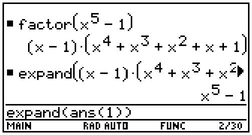
└─for factoring with respect to a variable

expand(*expression* [,*var*]**)**

└─for partial expansion with respect to a variable

Factor $\mathbf{x^5 - 1}$. Then expand the result.

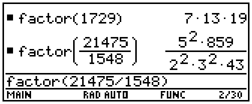
Notice that **factor** and **expand** perform opposite operations.



Finding Prime Factors of a Number

The **factor** ([F2] **2**) function lets you do more than simply factor an algebraic polynomial.

You can find prime factors of a rational number (either an integer or a ratio of integers).

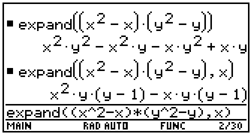


Finding Partial Expansions

With the **expand** (F2 **3**) function's optional var value, you can do a partial expansion that collects similar powers of a variable.

Do a full expansion of **(x^2−x) (y^2−y)** with respect to all variables.

Then do a partial expansion with respect to x.



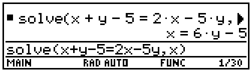
Solving an Equation

Use the **solve** (F2 **1**) function to solve an equation for a specified variable.

solve(*equation*, *var***)**

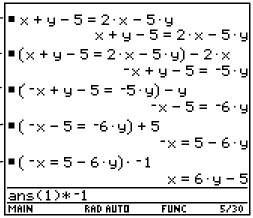
Solve **x + y − 5 = 2x − 5y** for x.

Notice that solve displays only the final result.



To see intermediate results, you can manually solve the equation step-by-step.

x [+] **y** [−] **5** [=] **2x** [−] **5y**
[−] **2 x**
[−] **y**
[+] **5**
[×] [(-)] **1**



Note: An operation such as [−] **2** [×] subtracts 2x from both sides.

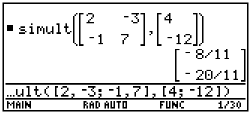
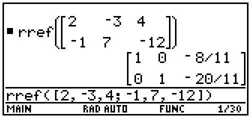
Solving a System of Linear Equations

Consider a set of two equations with two unknowns:

2x − 3y = 4
⁻x + 7y = ⁻12

To solve this system of equations, use any of the following methods.

Method	Example
Use the **solve** function for a one-step solution.	**solve(**2x−3y=4 and ⁻x+7y=⁻12,{x,y}**)**
Use the **solve** function with substitution (|) for step-by-step manipulation.	Substitutions are in the form of an equality, such as x=3 or y=sin(x). To be most effective, the left side should be a simple variable. See ”Symbolic Manipulation” in the *Previews* chapter, which solved for **x = ⁻8/11** and **y = ⁻20/11**.

Method	Example
Use the **simult** function with a matrix.	Enter the coefficients as a matrix and the results as a constant column matrix. 
Use the **rref** function with a matrix.	Enter the coefficients as an augmented matrix. 

Note: The simult and rref matrix functions are not on the [F2] **Algebra** menu. Use [2nd] [MATH] **4** or the **Catalog.**

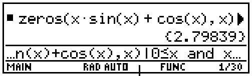
Use the **zeros** ([F2] **4**) function.

zeros(*expression*, *var***)**

Use the expression **x**

sin(x) + cos(x).

Find the zeros with respect to x in the interval **0** ≤ **x** and **x** ≤ **3**.



Use the "*with*" operator to specify the interval.

Note: For ≥ or ≤, type [♦] [>] or [♦] [<].You can also use [2nd] [MATH] **8** or [2nd] [CHAR] **2** to select them from a menu.

Use the **propFrac** ([F2] **7**) and **comDenom** ([F2] **6**) functions.

propFrac(*rational expression* [,*var*]**)**

for proper fractions with respect to a variable

comDenom(*expression* [,*var*]**)**

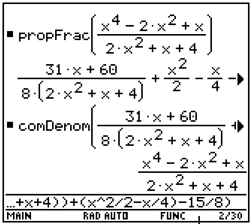
for common denominators that collect similar powers of this variable

Find a proper fraction for the expression **(x^4 − 2x^2 + x) / (2x^2 + x + 4).**

Then transform the answer into a ratio of a fully expanded numerator and a fully expanded denominator.

Notice that **propFrac** and **comDenom** perform opposite operations.

Note: You can use **comDenom** with an expression, list, or matrix.



If you do this example on your handheld, the **propFrac** function scrolls off the top of the screen.

In this example:

- $\frac{31x + 60}{8}$ is the remainder of **x^4−2x^2+x** divided by **2x^2+x+4.**

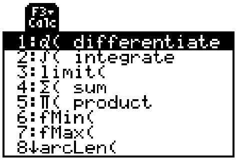
- $\frac{x^2}{2} - \frac{x}{4} - 15/8$ is the quotient.

Overview of the Calc Menu

You can use the [F3] **Calc** toolbar menu to select commonly used calculus functions.

The Calc Menu

From the Home screen, press [F3] to display:



This menu is also available from the MATH menu. Press [2nd] [MATH] and then select **A:Calculus**.

Note: For a complete description of each function and its syntax, refer to the *Technical Reference* module.

Menu Item	Description
d **differentiate**	Differentiates an expression with respect to a specified variable.

∫ **integrate**	Integrates an expression with respect to a specified variable.
limit	Calculates the limit of an expression with respect to a specified variable.
Σ **sum**	Evaluates an expression at discrete variable values within a range and then calculates the sum.
Π **product**	Evaluates an expression at discrete variable values within a range and then calculates the product.
fMin	Finds candidate values of a specified variable that minimize an expression.
fMax	Finds candidate values of a specified variable that maximize an expression.
arcLen	Returns the arc length of an expression with respect to a specified variable.
taylor	Calculates a Taylor polynomial approximation to an expression with respect to a specified variable.
nDeriv	Calculates the numerical derivative of an expression with respect to a specified variable.
nInt	Calculates an integral as a floating-point number using quadrature (an approximation using weighted sums of integrand values).
deSolve	Symbolically solves many 1st and 2nd order differential equations, with or without initial conditions.
impDif	Computes implicit derivatives for equations in two variables in which one variable is defined implicitly in terms of another.

Note: The d symbol for differentiate is a special symbol. It is not the same as typing the letter **D** on the keyboard. Use [F3] **1** or [2nd] [d].

Common Calculus Operations

This section gives examples for some of the functions available from the [F3] **Calc** toolbar menu. For complete information about any calculus function, refer to the *Technical Reference* module.

Integrating and Differentiating

Use the ∫ **integrate** ([F3] **2**) and d **differentiate** ([F3] **1**) functions.

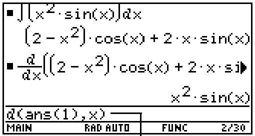
∫ **(***expression*, *var* [,*low*] [,*up*]**)**

[,*low*] [,*up*] — lets you specify limits or a constant of integration

d **(***expression*, *var* [,*order*]**)**

Integrate **x**^{2}∗**sin(x)** with respect to x.

Differentiate the answer with respect to x.



To get d, use [F3] **1** or [2nd] [d]. Do not simply type the letter D on the keyboard.

Note: You can integrate an expression only; you can differentiate an expression, list, or matrix.

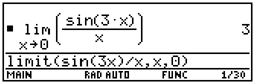
Finding a Limit

Use the **limit** ([F3] **3**) function.

limit(*expression*, *var*, *point* [,*direction*]**)**

negative number = from left
positive number= from right
omitted number or 0 = both

Find the limit of **sin(3x) / x** as x approaches 0.



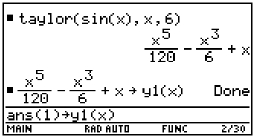
Note: You can find a limit for an expression, list, or matrix.

Use the **taylor** ([F3] **9**) function.

taylor(*expression*, *var*, *order* [,*point*]**)**

if omitted, expansion point is 0

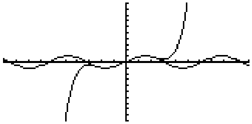
Find a 6th order Taylor polynomial for **sin(x)** with respect to x**.**



Store the answer as a user-defined function named **y1(x)**.

Then graph **sin(x)** and the Taylor polynomial.

```
Graph sin(x):Graph
y1(x)
```



Important: Degree-mode scaling by π/180 may cause calculus application results to appear in a different form.

User-Defined Functions and Symbolic Manipulation

You can use a user-defined function as an argument for the TI-89 Titanium ’s built-in algebra and calculus functions.

For Information about Creating a User-Defined Function

Refer to:

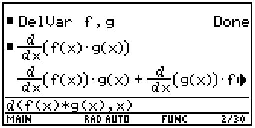
- “Creating and Evaluating User-Defined Functions” in the *Calculator Home Screen* module.
- “Graphing a Function Defined on the Home Screen” and “Graphing a Piecewise Defined Function” in the *Calculator Home Screen* module.
- “Overview of Entering a Function” in the *Programming* module.

Undefined Functions

You can use functions such as **f(x)**, **g(t)**, **r(θ)**, etc., that have not been assigned a definition. These “undefined” functions yield symbolic results. For example:

Use **DelVar** to ensure that **f(x)** and **g(x)** are not defined.

Then find the derivative of **f(x) g(x)** with respect to x.



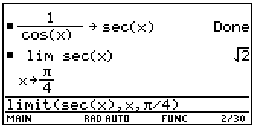
Note: To select **d** from the Calc toolbar menu, press [F3] **1** (or press [2nd] [*d*] on the keyboard).

Single-Statement Functions

You can use user-defined functions consisting of a single expression. For example:

- Use [STO▶] to create a user-defined secant function, where:

$$\sec x = \frac{1}{\cos x}$$



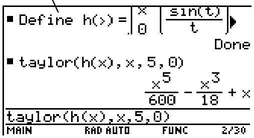
Then find the limit of **sec(x)** as *x* approaches π/4.

Note: To select **limit** from the Calc toolbar menu, press [F3] **3**.

- Use **Define** to create a user-defined function **h(x)**, where:

$$h(x) = \int_0^x \frac{\sin t}{t}$$

Define h(x)=
∫(sin(t)/t,t,0,x).



Then find a 5th order Taylor polynomial for **h(x)** with respect to *x*.

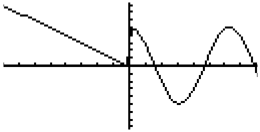
Note: To select ∫ from the **Calc** toolbar menu, press [F3] **2** (or press [2nd] [∫] on the keyboard). To select taylor, press [F3] **9**.

Multi-Statement vs. Single-Statement Functions

Multi-statement user-defined functions should be used as an argument for numeric functions (such as **nDeriv** and **nInt**) only.

In some cases, you may be able to create an equivalent single-statement function. For example, consider a piecewise function with two pieces.

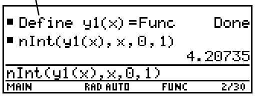
When:	Use expression:
x < 0	⁻x
x ≥ 0	5 cos(x)



- If you were to create a multi-statement user-defined function with the form:

```
Func
  If x<0 Then
  Return ëx
Else
  Return 5cos(x)
EndIf
EndFunc
```

Define
y1(x)=Func:If x<0
Then: ... :EndFunc



Then numerically integrate **y1(x)** with respect to *x*.

Note: To select **nInt** from the Calc toolbar menu, press [F3] **B:nInt**.

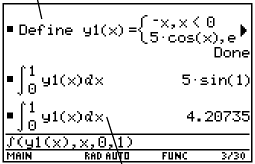
- Create an equivalent single-statement user-defined function.

Use the TI-89 Titanium’s built-in **when** function.

Then integrate **y1(x)** with respect to *x*.

Note: To select ∫ from the Calc toolbar menu, press [F3] **2** (or press [2nd] [∫] on the keyboard).

Define y1(x)=when(x<0, ⁻x, 5cos(x))



Press [♦] [ENTER] for a floating-point result.

If You Get an Out-of-Memory Error

The TI-89 Titanium stores intermediate results in memory and then deletes them when the calculation is complete. Depending on the complexity of the calculation, the handheld may run out of memory before a result can be calculated.

Freeing Up Memory

- Delete unneeded variables and/or Flash applications, particularly large-sized ones.
  - Use [2nd] [VAR-LINK] as described in *Memory and Variable Management* to view and delete variables and/or Flash applications.
- On the Home screen:

- Clear the history area ([F1] **8**) or delete unneeded history pairs.
- You can also use [F1] **9** to reduce the number of history pairs that will be saved.

- Use [MODE] to set Exact/Approx = APPROXIMATE. (For results that have a large number of digits, this uses less memory than **AUTO** or EXACT. For results that have only a few digits, this uses more memory.)

Simplifying Problems

- Split the problem into parts.
  - Split **solve(**a∗b=0,*var***)** into **solve(**a=0,*var***)** and **solve(**b=0,var**)**. Solve each part and combine the results.
- If several undefined variables occur only in a certain combination, replace that combination with a single variable.
  - If m and c occur only as **m∗c**^{2}, substitute e for **m∗c**^{2}.
  - In the expression $\frac{(a+b)^2+\sqrt{(a+b)^2}}{1-(a+b)^2}$, substitute c for **(a+b)** and use $\frac{c^2+\sqrt{c^2}}{1-c^2}$. In the solution, replace c with **(a+b)**.
- For expressions combined over a common denominator, replace sums in denominators with unique new undefined variables.
  - In the expression $\frac{x}{\sqrt{a^2+b^2}+c}+\frac{y}{\sqrt{a^2+b^2}+c}$, substitute d for $\sqrt{a^2+b^2}+c$ and are $\frac{x}{d}+\frac{y}{d}$. In the solution, replace d with $\sqrt{a^2+b^2}+c$.
- Substitute known numeric values for undefined variables at an earlier stage, particularly if they are simple integers or fractions.

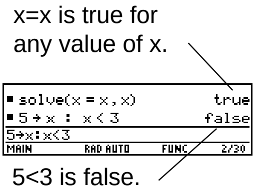
- Reformulate a problem to avoid fractional powers.
- Omit relatively small terms to find an approximation.

Special Constants Used in Symbolic Manipulation

The result of a calculation may include one of the special constants described in this section. In some cases, you may also need to enter a constant as part of your entry.

true, false

These indicate the result of an identity or a Boolean expression.



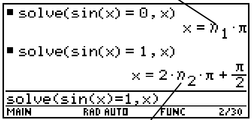
@n1 ... @n255

This notation indicates an “arbitrary integer” that represents any integer.

When an arbitrary integer occurs multiple times in the same session, each occurrence is numbered consecutively. After it reaches 255, arbitrary integer consecutive numbering restarts at @n0. Use Clean Up **2:NewProb** to reset to @n1.

Note: For @, press: [♦] [STO▸]

A solution is at every integer multiple of π.



Both @n and @n2 represent any arbitrary integer, but this notation identifies separate arbitrary integers.

@1 ... @255

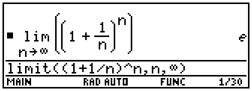
This notation indicates an “arbitrary constant” that represents any integer.

When an arbitrary constant occurs multiple times in the same session, each occurrence is numbered consecutively. After it reaches 255, arbitrary integer consecutive numbering restarts at @0. Use Clean Up **2:NewProb** to reset to @1.

Note: For @, press: [♦] [STO▸]

∞, *e*

∞ represents infinity, and e represents the constant **2.71828...** (base of the natural logarithms).



These constants are often used in entries as well as results.

Notes:
For ∞, press:
[♦] [∞]
For *e*, press:
[♦] [e^x]

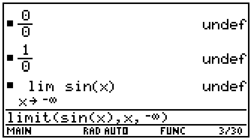
undef

This indicates that the result is undefined.

Mathematically undefined

±∞ (undetermined sign)

Non-unique limit



Constants and Measurement Units

Entering Constants or Units

You can use a menu to select from a list of available constants and units, or you can type them directly from the keyboard.

From a Menu

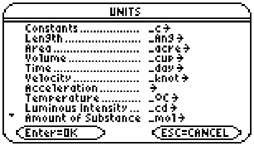
The following shows how to select a unit, but you can use the same general procedure to select a constant.

From the Home screen:

1. Type the value or expression.

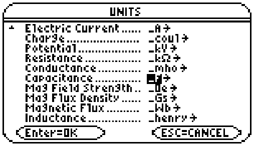
   6.3

2. Display the **UNITS** dialog box. Press:
   [2nd] [UNITS]



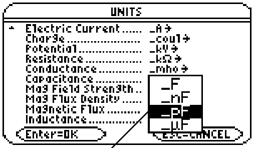
3. Use ⊖ and ⊕ to move the cursor to the applicable category.

   Note: Use [2nd] ⊖ and [2nd] ⊕ to scroll one page at a time through the categories.



4. To select the highlighted (default) unit, press [ENTER].
– or –
To select a different unit from the category, press ⊙. Then highlight the applicable unit, and press [ENTER].

Note: If you created a user-defined unit for an existing category, it is listed in the menu.



You can also move the cursor by typing the first letter of a unit.

The selected unit is placed in the entry line. Constant and unit names always begin with an underscore (_).

6.3_pF

From the Keyboard

If you know the abbreviation that the TI-89 Titanium uses for a particular constant or unit, you can type it directly from the keyboard. For example:

256_m

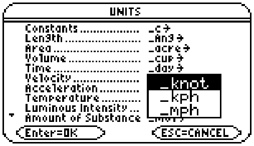
- The first character must be an underscore (_). For _, press:
  [♦] [_]

- A space or a multiplication symbol (∗) before the underscore is optional. For example, **256_m, 256 _m**, and **256∗_m** are equivalent.
- However, if you are adding units to a variable, you must put a space or ∗ before the underscore. For example, *x_m* is treated as a variable, not as x with a unit.

Note: You can type units in either uppercase or lowercase characters.

You may need to combine two or more units from different categories.

For example, suppose you want to enter a velocity in meters per second. In the UNITS dialog box, however, the **Velocity** category does not contain this unit.



You can enter meters per second by combining **_m** and **_s** from the **Length** and **Time** categories, respectively.

3∗9.8_m/_s

Combine the units _m and _s. There is no pre-defined m/_s unit.

Note: Create a user-defined unit for frequently used combinations.

In a calculation, you may need to use parentheses ( ) to group a value and its units so that they are evaluated properly. This is particularly true for division problems. For example:

To calculate:	Enter:
$\frac{100\_m}{2\_s}$	100_m/(2_s) 50 • $\frac{\_m}{\_s}$ You must use parentheses for (2_s). This is important for division. If you omit the parentheses, you will get unexpected units. For example: 100_m/2_s 50. ∗_m ∗_s

Note: If you have any doubt about how a value and its units will be evaluated, group them within parentheses ( ).

Here's why you get unexpected units if you do not use parentheses. In a calculation, a unit is treated similar to a variable. For example: **100_m** is treated as **100∗_m** and **2_s** is treated as **2∗_s**. Without parentheses, the entry is calculated as:

$$100*\_m / 2*\_s = \frac{100*\_m}{2} *\_s = 50. *\_m *\_s.$$

Converting from One Unit to Another

You can convert from one unit to another in the same category, including any user-defined units.

For All Units Except Temperature

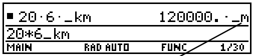
If you use a unit in a calculation, it is converted and displayed automatically in the current default unit for that category, unless you use the ▶ conversion operator as described later. The following examples assume that your default units are set to the SI system of metric units.

Notes:

- Refer to the list of pre-defined units.
- From the UNITS dialog box, you can select available units from a menu.

To multiply 20 times 6 kilometers.

20∗6_km



Shown in the default unit for Length, (_m in SI system).

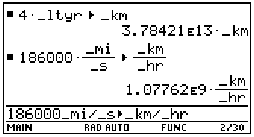
If you want to convert to a unit other than the default, use the ▸ conversion operator.

expression_unit1 ▸ *_unit2*

└── For ▸, press 2nd [▸].

To convert 4 light years to kilometers:

4_ltyr ▸ _km



To convert 186000 miles/second to kilometers/hour:

186000_mi/_s ▸ _km/_hr

If an expression uses a combination of units, you can specify a conversion for some of the units only. Any units for which you do not specify a conversion will be displayed according to your defaults.

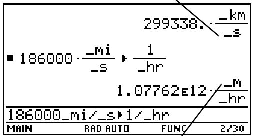
To convert 186000 miles/second from miles to kilometers:

186000_mi/_s ▸ _km

To convert 186000 miles/second from seconds to hours:

186000_mi/_s ▸ 1/_hr

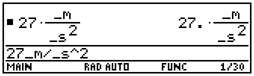
Because a Time conversion is not specified, it is shown in its default unit (_s in this example).



Because a Length conversion is not specified, it is shown in its default unit (_m in this example).

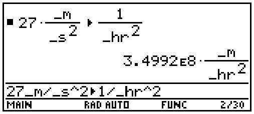
To enter meters per second squared:

27_m/_s^2



To convert meters per second squared from seconds to hours:

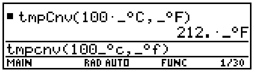
27_m/_s^2 ▸ 1/_hr^2



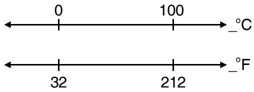
To convert a temperature value, you must use **tmpCnv( )** instead of the ▸ operator.

tmpCnv(*expression_°tempUnit1, _°tempUnit2***)**
└─For °, press [2nd] [°]

For example, to convert 100_°C to _°F:



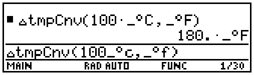
tmpCnv(*100_°c, _°f***)**



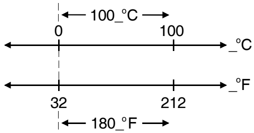
To convert a temperature range (the difference between two temperature values), use **∆tmpCnv( ).**

∆tmpCnv(*expression_°tempUnit1, _°tempUnit2***)**

For example, to convert a 100_°C range to its equivalent range in _°F:



∆tmpCnv(*100_°c***,** *_°f***)**



Note: For ∆, press:
[♦] [(] [↑] [D]

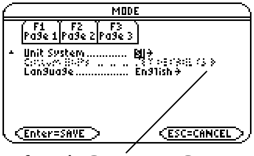
Setting the Default Units for Displayed Results

All results involving units are displayed in the default unit for that category. For example, if the default unit for **Length** is *_m*, any length result is displayed in meters (even if you entered *_km* or *_ft* in the calculation).

If You're Using the SI or ENG/US System

The SI and ENG/US systems of measurement (set from **Page 3** of the MODE screen) use built-in default units, which you cannot change.

The default units for these systems are available.

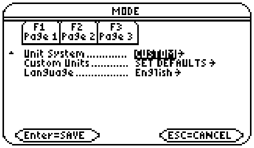


If Unit System=SI or ENG/US, the Custom Units item is dimmed. You cannot set a default for individual categories.

Setting Custom Defaults

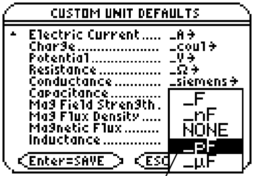
To set custom defaults:

1. Press MODE F3 ▶ **3** to set **Unit System = CUSTOM**.
2. Press ▼ to highlight **SET DEFAULTS**.
3. Press ▶ to display the **CUSTOM UNIT DEFAULTS** dialog box.



4. For each category, you can highlight its default, press ⓡ, and select a unit from the list.

5. Press [ENTER] twice to save your changes and exit the **MODE** screen.



You can also move the cursor by typing the first letter of a unit.

Notes:

- You can also use **setUnits( )** or **getUnits( )** to set or return information about default units. Refer to the *Technical Reference* module.

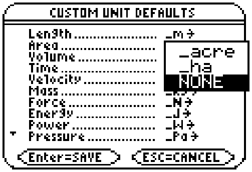
- When the **CUSTOM UNIT DEFAULTS** dialog box first appears, it shows the current default units.

What is the NONE Default?

Many categories let you select NONE as the default unit.

This means that results in that category are displayed in the default units of its components.

For example, **Area = Length**^{2}, so **Length** is the component of **Area**.



- If the defaults are **Area = _acre** and **Length = _m** (meters), area results are shown with **_acre** units.
- If you set **Area = NONE**, area results are shown with **_m**^{2} units.

Note: NONE is not available for base categories such as **Length** and **Mass** that have no components.

Creating Your Own User-Defined Units

In any category, you can expand the list of available units by defining a new unit in terms of one or more pre-defined units. You can also use “standalone” units.

Why Use Your Own Units?

Some example reasons to create a unit are:

- You want to enter length values in dekameters. Define *10_m* as a new unit named *_dm*.
- Instead of entering *_m/_s*^{2} as an acceleration unit, you define that combination of units as a single unit named *_ms2*.
- You want to calculate how many times someone blinks. You can use *_blinks* as a valid unit without defining it. This “standalone” unit is treated similar to a variable that is not defined. For instance, *3_blinks* is treated the same as *3a*.

Note: If you create a user-defined unit for an existing category, you can select it from the UNITS dialog box menu. But you cannot use [MODE] to select the unit as a default for displayed results.

Rules for User-Defined Unit Names

The naming rules for units are similar to variables.

- Can have up to 8 characters.
- First character must be an underscore. For _, press:
  [♦] [_]
- Second character can be any valid variable name character except _ or a digit. For example, *_9f* is not valid.
- Remaining characters (up to 6) can be any valid variable name character except an underscore.

Defining a Unit

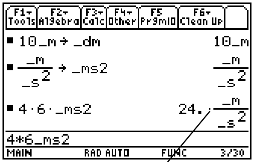
Define a unit the same way you store to a variable.

definition → *_newUnit*

For →, press [STO▸]

For example, to define a dekameter unit:

10_m →_dm



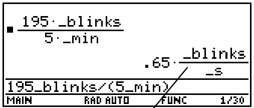
Assuming unit defaults for Length and Time are set to _m and _s.

To define an acceleration unit:

_m/_s^2 →_ms2

To calculate 195 blinks in 5 minutes as _blinks/_min:

195_blinks/(5_min)



Assuming unit default for Time is set to _s.

Notes:

- User-defined units are displayed in lowercase characters, regardless of the case you use to define them.
- User-defined units such as *_dm* are stored as variables. You can delete them the same as you would any variable.

List of Pre-Defined Constants and Units

This section lists the pre-defined constants and units by category. You can select any of these from the UNITS dialog box. If you use [MODE] to set default units, note that categories with only one defined unit are not listed.

Defaults for SI and ENG/US

The SI and ENG/US systems of measurement use built-in default units. In this section, the built-in defaults are indicated by (SI) and (ENG/US). In some categories, both systems use the same default.

Some categories do not have default units.

Constants

	Description	**Value**
_c	speed of light	2.99792458E8_m/_s
_Cc	coulomb constant	8.9875517873682E9_N•_m^2 /_coul^2
_g	acceleration of gravity	9.80665_m/_s^2
_Gc	gravitational constant	6.6742E⁻11_m^3/_kg/_s^2
_h	Planck’s constant	6.6260693E⁻34_J•_s
_k	Boltzmann’s constant	1.3806505E⁻23_J/_°K
_Me	electron rest mass	9.1093826E⁻31_kg

	Description	Value
_Mn	neutron rest mass	1.67492728E-27_kg
_Mp	proton rest mass	1.67262171E-27_kg
_Na	Avogadro's number	6.0221415E23 /_mol
_q	electron charge	1.60217653E-19_coul
_Rb	Bohr radius	5.291772108E-11_m
_Rc	molar gas constant	8.314472_J/_mol/_°K
_Rdb	Rydberg constant	10973731.568525 /_m
_Vm	molar volume	2.2413996E-2_m^3/_mol
_ε0	permittivity of a vacuum	8.8541878176204E-12_F/_m
_σ	Stefan-Boltzmann constant	5.670400E-8_W/_m^2/_°K^4
_φ0	magnetic flux quantum	2.06783372E-15_Wb
_μ0	permeability of a vacuum	1.2566370614359E-6_N/_A^2
_μb	Bohr magneton	9.27400949E-24_J • _m^2/_Wb

Notes:

- The calculator simplifies unit expressions and displays results according to your default units. Therefore, constant values displayed on your screen may appear different from the values in this table.
- For Greek characters, refer to *Quick Reference Key Table.*

- These values represent the most up-to-date constants available at time of printing from the CODATA Internationally recommended values of the Fundamental Physical Constants available on the National Institute of Standards and Technology (NIST) web site. (http://physics.nist.gov/cuu/Constants/index.html).

Length

_Ang	angstrom	_mi	mile
_au	astronomical unit	_mil	1/1000 inch
_cm	centimeter	_mm	millimeter
_fath	fathom	_Nmi	nautical mile
_fm	fermi	_pc	parsec
_ft	foot (ENG/US)	_rod	rod
_in	inch	_yd	yard
_km	kilometer	_µ	micron
_ltyr	light year	_Å	angstrom
_m	meter (SI)		

Area

_acre	acre	NONE (SI) (ENG/US)	
_ha	hectare		

Volume

_cup	cup	_ml	milliliter
_floz	fluid ounce	_pt	pint
_flozUK	British fluid ounce	_qt	quart
_gal	gallon	_tbsp	tablespoon
_galUK	British gallon	_tsp	teaspoon
_l	liter	NONE (SI) (ENG/US)	

Time

_day	day	_s	second (SI) (ENG/US)
_hr	hour	_week	week
_min	minute	_yr	year
_ms	millisecond	_μs	microsecond
_ns	nanosecond		

Velocity

_knot	knot	_mph	miles per hour
_kph	kilometers per hour	NONE (SI) (ENG/US)	

Acceleration

no pre-defined units			

Temperature

_°C	°Celsius (For °, press 2nd [°].)	_°K	°Kelvin
_°F	°Fahrenheit	_°R	°Rankine (no default)

Luminous Intensity

_cd	candela (no default)		

Amount of Substance

_mol	mole (no default)		

Mass

_amu	atomic mass unit	_oz	ounce
_gm	gram	_slug	slug
_kg	kilogram (SI)	_ton	ton
_lb	pound (ENG/US)	_tonne	metric ton

_mg	milligram	_tonUK	long ton
_mton	metric ton		

Force

_dyne	dyne	_N	newton (SI)
_kgf	kilogram force	_tonf	ton force
_lbf	pound force (ENG/US)		

Energy

_Btu	British thermal unit (ENG/US)	_J	joule (SI)
_cal	calorie	_kcal	kilocalorie
_erg	erg	_kWh	kilowatt-hour
_eV	electron volt	_latm	liter-atmosphere
_ftlb	foot-pound		

Power

_hp	horsepower (ENG/US)	_W	watt (SI)
_kW	kilowatt		

Pressure

_atm	atmosphere	_mmHg	millimeters of mercury
_bar	bar	_Pa	pascal (SI)
_inH2O	inches of water	_psi	pounds per square inch (ENG/US)
_inHg	inches of mercury	_torr	millimeters of mercury
_mmH2O	millimeters of water		

Viscosity, Kinematic

_St	stokes		

Viscosity, Dynamic

_P	poise		

Frequency

_GHz	gigahertz	_kHz	kilohertz
_Hz	hertz (SI) (ENG/US	_MHz	megahertz

Electric Current

_A	ampere (SI) (ENG/US)	_mA	milliampere
_kA	kiloampere	_μA	microampere

Charge

_coul	coulomb (SI) (ENG/US)		

Potential

_kV	kilovolt	_V	volt (SI) (ENG/US)
_mV	millivolt	_volt	volt

Resistance

_kΩ	kilo ohm	_ohm	ohm
_MΩ	megaohm	_Ω	ohm (SI) (ENG/US)

Conductance

_mho	mho (ENG/US)	_siemens	siemens (SI)
_mmho	millimho	_μmho	micromho

Capacitance

_F	farad (SI) (ENG/US)	_pF	picofarad
_nF	nanofarad	_µF	microfarad

Mag Field Strength

_Oe	oersted	NONE (SI) (ENG/US)

Mag Flux Density

_Gs	gauss	_T	tesla (SI) (ENG/US)

Magnetic Flux

_Wb	weber (SI) (ENG/US)		

Inductance

_henry	henry (SI) (ENG/US)	_nH	nanohenry
_mH	millihenry	_µH	microhenry

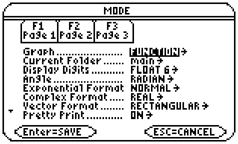
Basic Function Graphing

Overview of Steps in Graphing Functions

To graph one or more **y(x)** functions, use the general steps shown below. For a detailed description of each step, refer to the following pages. You may not need to do all the steps each time you graph a function.

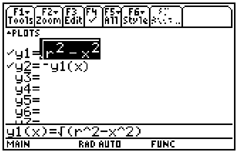
Graphing Functions

1. Set **Graph** mode ([MODE]) to **FUNCTION**. Also set **Angle** mode, if necessary.



2. Define x and y components on Y= Editor ([♦] [Y=]).
3. Select ([F4]) which defined functions to graph.

   Note: To turn off any stat data plots, press [F5] **5** or use [F4] to deselect them.



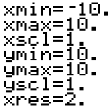
4. Set the display style for a function.

   [2nd] [F6]

   This is optional. For multiple equations, this helps visually distinguish one from another.

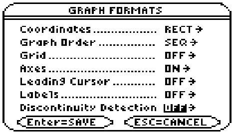
5. Define the viewing window [♦] [WINDOW]).

   [F2] **Zoom** also changes the viewing window.

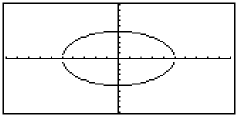


6. Change the graph format if necessary.

   [F1] **9**
   – or –
   [♦] [I]



7. Graph the selected functions ([♦] [GRAPH]).



Exploring the Graph

From the Graph screen, you can:

- Display the coordinates of any pixel by using the free-moving cursor, or of a plotted point by tracing a function.
- Use the [F2] **Zoom** toolbar menu to zoom in or out on a portion of the graph.

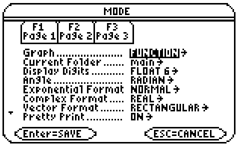
- Use the F5 **Math** toolbar menu to find a zero, minimum, maximum, etc.

Setting the Graph Mode

Before graphing **y(x)** functions, you must select FUNCTION graphing. You may also need to set the Angle mode, which affects how the TI-89 Titanium graphs trigonometric functions.

Graph Mode

1. Press MODE to display the **MODE** dialog box, which shows the current mode settings.



2. Set the Graph mode to **FUNCTION.** Refer to “Setting Modes” in *Operating the Calculator*.

   For graphs that do not use complex numbers, set **Complex Format = REAL.** Otherwise, it may affect graphs that use powers, such as $\mathbf{x}^{1/3}$.

While this module specifically describes **y(x)** function graphs, the calculator lets you select from six Graph mode settings.

Graph Mode Setting	Description
FUNCTION	**y(x)** functions
PARAMETRIC	**x(t)** and **y(t)** parametric equations

Graph Mode Setting	Description
POLAR	**r(θ)** polar equations
SEQUENCE	**u(n)** sequences
3D	**z(x,y)** 3D equations
DIFFERENTIAL EQUATION	**y'(t)** differential equations

Angle Mode

When using trigonometric functions, set the Angle mode for the units (RADIAN, DEGREE or GRADIAN) in which you want to enter and display angle values.

Checking the Status Line

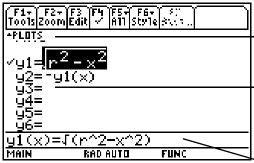
To see the current Graph mode and Angle mode, check the status line at the bottom of the screen.

Defining Functions for Graphing

In FUNCTION graphing mode, you can graph functions named **y1(x)** through **y99(x)**. To define and edit these functions, use the Y= Editor. (The Y= Editor lists function names for the current graphing mode. For example, in POLAR graphing mode, function names are **r1(θ)**, **r2(θ)**, etc.)

Defining a New Function

1. Press [♦] [Y=] to display the Y= Editor.



Plots — You can scroll above y1= to see a list of stat plots.

Function List — You can scroll through the list of functions and definitions.

Entry Line — Where you define or edit the function highlighted in the list.

Note: The function list shows abbreviated function names such as **y1**, but the entry line shows the full name **y1(x)**.

2. Press ▼ and ▲ to move the cursor to any undefined function. (Use [2nd] ▼ and [2nd] ▲ to scroll one page at a time.)
3. Press [ENTER] or [F3] to move the cursor to the entry line.
4. Type the expression to define the function.
   - The independent variable in function graphing is x.
   - The expression can refer to other variables, including matrices, lists, and other functions. Only floats and lists of floats will produce a plot.

   Note: For an undefined function, you do not need to press [ENTER] or [F3]. When you begin typing, the cursor moves to the entry line.
5. When you complete the expression, press [ENTER].

The function list now shows the new function, which is automatically selected for graphing.

Note: If you accidentally move the cursor to the entry line, press [ESC] to move it back to the function list.

Editing a Function

From the Y= Editor:

1. Press ⓓ and ⓤ to highlight the function.
2. Press [ENTER] or [F3] to move the cursor to the entry line.
3. Do any of the following:
   - Use ⓡ and ⓛ to move the cursor within the expression and edit it. Refer to “Editing an Expression in the Entry Line” in *Operating the Calculator*.

     – or –
   - Press [CLEAR] once or twice to clear the old expression, and then type the new one.
4. Press [ENTER].

   The function list now shows the edited function, which is automatically selected for graphing.

   Note: To cancel any editing changes, press [ESC] instead of [ENTER].

Clearing a Function

From the Y= Editor:

To erase:	Do this:
A function from the function list	Highlight the function and press [←] or [CLEAR].
A function from the entry line	Press [CLEAR] once or twice (depending on the cursor's location) and then press [ENTER].
All functions	Press [F1] and then select **8:Clear Functions**. When prompted for confirmation, press [ENTER].

Note: [F1] **8** does not erase any stat plots.

You don't have to clear a function to prevent it from being graphed. You can select the functions you want to graph.

Shortcuts to Move the Cursor

From the Y= Editor:

Press:	To:
[♦] ⊖ or [♦] ⊖	Go to function 1 or to the last defined function, respectively. If the cursor is on or past the last defined function, [♦] ⊖ goes to function 99.

From the Home Screen or a Program

You can also define and evaluate a function from the Home screen or a program.

- Use the **Define** and **Graph** commands. Refer to:
  - “Graphing a Function Defined on the Home Screen” and “Graphing a Piecewise Defined Function” in *Additional Graphing Topics*.
  - “Overview of Entering a Function” in *Programming*.
- Store an expression directly to a function variable. Refer to:
  - “Storing and Recalling Variable Values” in *Operating the Calculator*.
  - “Creating and Evaluating User-Defined Functions” in *Calculator Home Screen*.

Note: User-defined functions can have almost any name. However, if you want them to appear in the Y= Editor, use function names **y1(x)**, **y2(x)**, etc.

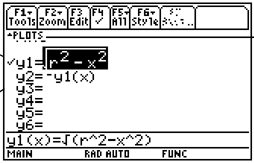
Selecting Functions to Graph

Regardless of how many functions are defined in the Y= Editor, you can select the ones you want to graph.

Selecting or Deselecting Functions

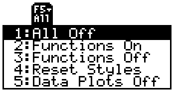
Press ♦ [Y=] to display the Y= Editor.

A “✓” indicates which functions will be graphed the next time you display the Graph screen.



If PLOT numbers are displayed, those stat plots are selected.

In this example, Plots 1 and 2 are selected. To view them, scroll above y1=.

To select or deselect:	Do this:
A specified function	• Move the cursor to highlight the function. • Press [F4]. This procedure selects a deselected function or deselects a selected function.
All functions	• Press [F5] to display the **All** toolbar menu. • Select the applicable item. 

You don’t have to select a function when you enter or edit it; it is selected automatically. To turn off any stat plots, press [F5] **5** or use [F4] to deselect them.

From the Home Screen or a Program

You can also select or deselect functions from the Home screen or a program.

- Use the **FnOn** and **FnOff** commands (available from the Home screen's [F4] **Other** toolbar menu) for functions. Refer to the *Technical Reference* module.
- Use the **PlotsOn** and **PlotsOff** commands for stat plots. Refer to the *Technical Reference* module.

Setting the Display Style for a Function

For each defined function, you can set a style that specifies how that function will be graphed. This is useful when graphing multiple functions. For example, set one as a solid line, another as a dotted line, etc.

Displaying or Changing a Function's Style

From the Y= Editor:

1. Move the cursor to highlight the applicable function.
2. Select the **Style** menu and press: [2nd] [F6]

- Although the Line item is initially highlighted, the function's current style is indicated by a ✓ mark.
- To exit the menu without making a change, press [ESC].

3. To make a change, select the applicable style.

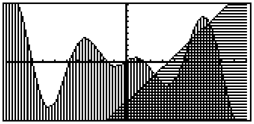
Style	Description
Line	Connects plotted points with a line. This is the default.
Dot	Displays a dot at each plotted point.
Square	Displays a solid box at each plotted point.
Thick	Connects plotted points with a thick line.
Animate	A round cursor moves along the leading edge of the graph but does not leave a path.
Path	A round cursor moves along the leading edge of the graph and does leave a path.
Above	Shades the area above the graph.
Below	Shades the area below the graph.

To set Line as the style for all functions, press F5 and select **4:Reset Styles**.

If You Use Above or Below Shading

The TI-89 Titanium has four shading patterns, used on a rotating basis. If you set one function as shaded, it uses the first pattern. The next shaded function uses the second pattern, etc. The fifth shaded function reuses the first pattern.

When shaded areas intersect, their patterns overlap.



From the Home Screen or a Program

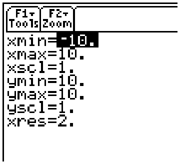
You can also set a function’s style from the Home screen or a program. Refer to the **Style** command in the *Technical Reference* module.

Defining the Viewing Window

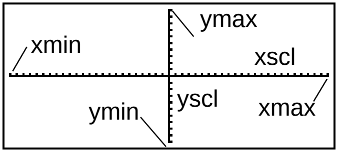
The viewing window represents the portion of the coordinate plane displayed on the Graph screen. By setting Window variables, you can define the viewing window’s boundaries and other attributes. Function graphs, parametric graphs, etc., have their own independent set of Window variables.

Displaying Window Variables in the Window Editor

Press [♦] [WINDOW] to display the Window Editor.



Window Variables
(shown in Window Editor)



Corresponding Viewing Window
(shown on Graph screen)

Variable	Description
xmin, xmax, ymin, ymax	Boundaries of the viewing window.

Variable	Description
xscl, yscl	Distance between tick marks on the x and y axes.
xres	Sets pixel resolution (1 through 10) for function graphs. The default is 2. • At 1, functions are evaluated and graphed at each pixel along the x axis. • At 10, functions are evaluated and graphed at every 10th pixel along the x axis.

To turn off tick marks, set **xscl=0** and/or **yscl=0**. Small values of **xres** improve the graph's resolution but may reduce the graphing speed.

Changing the Values

From the Window Editor:

1. Move the cursor to highlight the value you want to change.
2. Do any of the following:
   - Type a value or an expression. The old value is erased when you begin typing.

     – or –
   - Press [CLEAR] to clear the old value; then type the new one.

     – or –
   - Press ◀ or ▶ to remove the highlighting; then edit the value.

Values are stored as you type them; you do not need to press [ENTER]. [ENTER] simply moves the cursor to the next Window variable. If you type an expression, it is evaluated when you move the cursor to a different Window variable or leave the Window Editor.

From the Home Screen or a Program

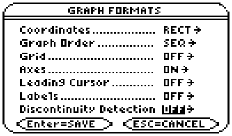
You can also store values directly to the Window variables from the Home screen or a program. Refer to “Storing and Recalling Variable Values” in *Operating the Calculator*.

Changing the Graph Format

You can set the graph format to show or hide reference elements such as the axes, a grid, and the cursor’s coordinates. Function graphs, parametric graphs, etc., have their own independent set of graph formats.

Displaying Graph Format Settings

From the Y= Editor, Window Editor, or Graph screen, press [F1] and select **9:Format.**



- The GRAPH FORMATS dialog box shows the current settings.
- To exit without making a change, press [ESC].

You also can display the GRAPH FORMATS dialog box from the Y= Editor, Window Editor, or Graph screen. Press: [♦] [|]

Format	Description
Coordinates	Shows cursor coordinates in rectangular (RECT) or polar (POLAR) form, or hides (OFF) the coordinates.

Format	Description
Graph Order	Graphs functions one at a time (SEQ) or all at the same time (SIMUL). Not available when Discontinuity Detection is set to ON.
Grid	Shows (ON) or hides (OFF) grid points that correspond to the tick marks on the axes.
Axes	Shows (ON) or hides (OFF) the x and y axes.
Leading Cursor	Shows (ON) or hides (OFF) a reference cursor that tracks the functions as they are graphed.
Labels	Shows (ON) or hides (OFF) labels for the x and y axes.
Discontinuity Detection	Eliminates (ON) or allows (OFF) faux asymptotes and connections in a jump discontinuity.

To turn off tick marks, define the viewing window so that **xscl** and/or **yscl = 0**.

Changing Settings

From the GRAPH FORMATS dialog box:

1. Move the cursor to highlight the format setting.
2. Press ▶ to display a menu of valid settings for that format.
3. Select a setting. Either:
   - Move the cursor to highlight the setting, and then press [ENTER].

     – or –
   - Press the number for that setting.

4. After changing all applicable format settings, press [ENTER] to save your changes and close the **GRAPH FORMATS** dialog box.

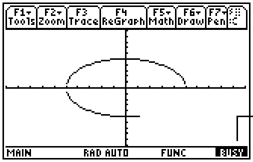
   Note: To cancel a menu or exit the dialog box without saving any changes, use [ESC] instead of [ENTER].

Graphing the Selected Functions

When you are ready to graph the selected functions, display the Graph screen. This screen uses the display style and viewing window that you previously defined.

Displaying the Graph Screen

Press [♦] [GRAPH]. The TI-89 Titanium automatically graphs the selected functions.



BUSY indicator shows while graphing is in progress.

If you select an [F2] **Zoom** operation from the Y= Editor or Window Editor, the TI-89 Titanium automatically displays the Graph screen.

Interrupting Graphing

While graphing is in progress:

- To pause graphing temporarily, press [ENTER]. (The PAUSE indicator replaces BUSY.) To resume, press [ENTER] again.
- To cancel graphing, press [ON]. To start graphing again from the beginning, press [F4] (**ReGraph**).

If You Need to Change the Viewing Window

Depending on various settings, a function may be graphed such that it is too small, too large, or offset too far to one side of the screen. To correct this:

- Redefine the viewing window with different boundaries.
- Use a Zoom operation.

Smart Graph

When you display the Graph screen, the Smart Graph feature displays the previous window contents immediately, provided nothing has changed that requires regraphing.

Smart Graph updates the window and regraphs only if you have:

- Changed a mode setting that affects graphing, a function’s graphing attribute, a Window variable, or a graph format.
- Selected or deselected a function or stat plot. (If you only select a new function, Smart Graph adds that function to the Graph screen.)
- Changed the definition of a selected function or the value of a variable in a selected function.
- Cleared a drawn object.

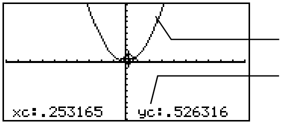
- Changed a stat plot definition.

Displaying Coordinates with the Free-Moving Cursor

To display the coordinates of any location on the Graph screen, use the free-moving cursor. You can move the cursor to any pixel on the screen; the cursor is not confined to a graphed function.

Free-Moving Cursor

When you first display the Graph screen, no cursor is visible. To display the cursor, press a cursor pad arrow. The cursor moves from the center of the screen, and its coordinates are displayed.



y1(x)=x^2

The “**c**” indicates these are cursor coordinates. The values are stored in the **xc** and **yc** system variables. Rectangular coordinates use **xc** and **yc**. Polar coordinates use **rc** and θ**c**.

If your screen does not show coordinates, set the graph format so that **Coordinates = RECT** or **POLAR**. Press:

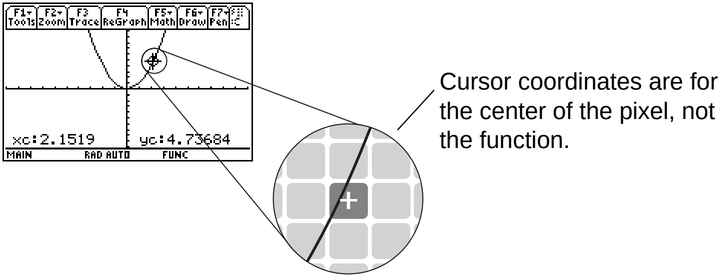
[◆] [I]

To move the free-moving cursor:	**Press:**
To an adjoining pixel	A cursor pad arrow for any direction.

To move the free-moving cursor:	Press:
In increments of 10 pixels	[2nd] and then a cursor pad arrow.

Note: To hide the cursor and its coordinates temporarily, press [CLEAR], [ESC], or [ENTER]. The next time you move the cursor, it moves from its last position.

When you move the cursor to a pixel that appears to be “on” the function, it may be near the function but not on it.



To increase the accuracy:

- Use the **Trace** tool described on the next page to display coordinates that are on the function.
- Use a **Zoom** operation to zoom in on a portion of the graph.

Tracing a Function

To display the exact coordinates of any plotted point on a graphed function, use the [F3] **Trace** tool. Unlike the free-moving cursor, the trace cursor moves only along a function's plotted points.

Beginning a Trace

From the Graph screen, press [F3].

The trace cursor appears on the function, at the middle x value on the screen. The cursor's coordinates are displayed at the bottom of the screen.

If multiple functions are graphed, the trace cursor appears on the lowest-numbered function selected in the Y= Editor. The function number is shown in the upper right part of the screen.

If any stat plots are graphed, the trace cursor appears on the lowest-numbered stat plot.

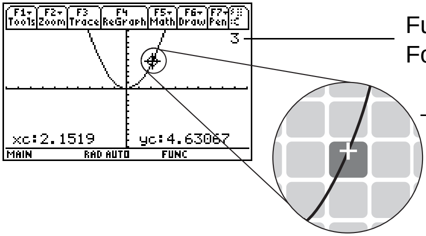
Moving along a Function

To move the trace cursor:	Do this:
To the previous or next plotted point	Press ◀ or ▶.
Approximately 5 plotted points (it may be more or less than 5, depending on the **xres** Window variable)	Press [2nd] ◀ or [2nd] ▶.

To move the trace cursor:	Do this:
To a specified x value on the function	Type the x value and press [ENTER].

Note: If you enter an x value, it must be between **xmin** and **xmax**.

The trace cursor moves only from plotted point to plotted point along the function, not from pixel to pixel.



If your screen does not show coordinates, set the graph format so that **Coordinates = RECT** or **POLAR**. Press:
[♦] [I]

Each displayed y value is calculated from the x value; that is, **y=yn(x)**. If the function is undefined at an x value, the y value is blank.

You can continue to trace a function that goes above or below the viewing window. You cannot see the cursor as it moves in that “off the screen” area, but the displayed coordinate values show its correct coordinates.

Note: Use QuickCenter to trace a function that goes above or below the window.

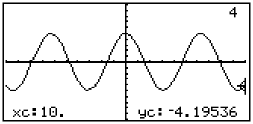
Moving from Function to Function

Press ⊖ or ⊖ to move to the previous or next selected function at the same x value. The new function number is shown on the screen.

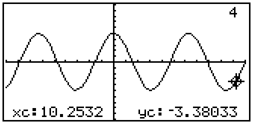
The “previous or next” function is based on the order of the selected functions in the Y= Editor, not the appearance of the functions as graphed on the screen.

Automatic Panning

If you trace a function off the left or right edge of the screen, the viewing window automatically pans to the left or right. There is a slight pause while the new portion of the graph is drawn.



Before automatic pan



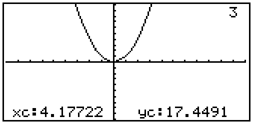
After automatic pan

After an automatic pan, the cursor continues tracing.

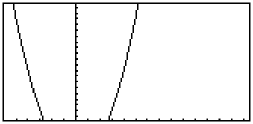
Note: Automatic panning does not work if stat plots are displayed or if a function uses a shaded display style.

Using QuickCenter

If you trace a function off the top or bottom of the viewing window, you can press [ENTER] to center the viewing window on the cursor location.



Before using QuickCenter



After using QuickCenter

After QuickCenter, the cursor stops tracing. If you want to continue tracing, press [F3].

You can use QuickCenter at any time during a trace, even when the cursor is still on the screen.

Canceling Trace

To cancel a trace at any time, press [ESC].

A trace is also canceled when you display another application screen such as the Y= Editor. When you return to the Graph screen and press [F3] to begin tracing:

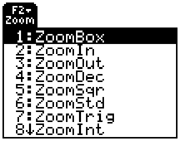
- If Smart Graph regraphed the screen, the cursor appears at the middle x value.
- If Smart Graph does not regraph the screen, the cursor appears at its previous location (before you displayed the other application).

Using Zooms to Explore a Graph

The [F2] **Zoom** toolbar menu has several tools that let you adjust the viewing window. You can also save a viewing window for later use.

Overview of the Zoom Menu

Press [F2] from the Y= Editor, Window Editor, or Graph screen.



Procedures for using **ZoomBox**, **ZoomIn**, **ZoomOut**, **ZoomStd**, **Memory**, and **SetFactors** are given later in this section.

For more information about the other items, refer to the *Technical Reference* module.

Note: If you select a **Zoom** tool from the Y=Editor or Window Editor, the TI-89 Titanium automatically displays the Graph screen.

Zoom Tool	Description
ZoomBox	Lets you draw a box and zoom in on that box.
ZoomIn, **ZoomOut**	Lets you select a point and zoom in or out by an amount defined by **SetFactors**.
ZoomDec	Sets Δx and Δy to .1, and centers the origin.
ZoomSqr	Adjusts Window variables so that a square or circle is shown in correct proportion (instead of a rectangle or ellipse).

Zoom Tool	Description
ZoomStd	Sets Window variables to their default values. xmin = ⁻10 ymin = ⁻10 xres = 2 xmax = 10 ymax = 10 xscl = 1 yscl = 1
ZoomTrig	Sets Window variables to preset values that are often appropriate for graphing trig functions. Centers the origin and sets: $\Delta x = \pi/24$ (.130899... radians or 7.5 degrees) ymin = −4 ymax = 4 xscl = $\pi/2$ (1.570796... radians or 90 degrees) yscl = 0.5
ZoomInt	Lets you select a new center point, and then sets Δx and Δy to 1 and sets **xscl** and **yscl** to 10.
ZoomData	Adjusts Window variables so that all selected stat plots are in view.
ZoomFit	Adjusts the viewing window to display the full range of dependent variable values for the selected functions. In function graphing, this maintains the current **xmin** and **xmax** and adjusts **ymin** and **ymax**.
Memory	Lets you store and recall Window variable settings so that you can recreate a custom viewing window.
SetFactors	Lets you set **Zoom** factors for **ZoomIn** and **ZoomOut**.

Δx and Δy are the distances from the center of one pixel to the center of an adjoining pixel.

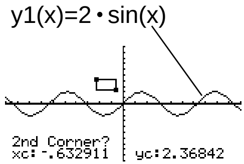
1. From the F2 **Zoom** menu, select **1:ZoomBox.**

   The screen prompts for **1st Corner?**

2. Move the cursor to any corner of the box you want to define, and then press ENTER.

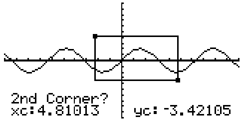
   The cursor changes to a small square, and the screen prompts for **2nd Corner?**

   Note: To move the cursor in larger increments, use 2nd ▶, 2nd ▼, etc.



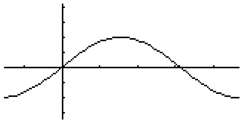
3. Move the cursor to the opposite corner of the zoom box.

   As you move the cursor, the box stretches.



4. When you have outlined the area you want to zoom in on, press ENTER.

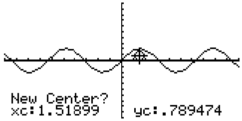
   The Graph screen shows the zoomed area. You can cancel **ZoomBox** by pressing ESC before you press ENTER.



Zooming In and Out on a Point

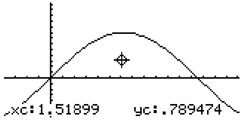
1. From the [F2] **Zoom** menu, select **2:ZoomIn** or **3:ZoomOut.**

   A cursor appears, and the screen prompts for **New Center?**



2. Move the cursor to the point where you want to zoom in or out, and then press [ENTER].

   The TI-89 Titanium adjusts the Window variables by the **Zoom** factors defined in **SetFactors.**



- For a **ZoomIn**, the x variables are divided by **xFact**, and the y variables are divided by **yFact.**

  new **xmin = xmin/xFact** , etc.

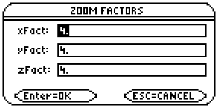
- For a **ZoomOut**, the x variables are multiplied by **xFact**, and the y variables are multiplied by **yFact.**

  new **xmin = xmin * xFact** , etc.

Changing Zoom Factors

The Zoom factors define the magnification and reduction used by **ZoomIn** and **ZoomOut.**

1. From the [F2] **Zoom** menu, select **C:SetFactors** to display the **ZOOM FACTORS** dialog box.



Zoom factors must be ≥ 1, but they do not have to be integers. The default setting is 4.

Note: To exit without saving any changes, press [ESC].

2. Use ▼ and ▲ to highlight the value you want to change. Then:
   - Type the new value. The old value is cleared automatically when you begin typing.

     – or –
   - Press ◀ or ▶ to remove the highlighting, and then edit the old value.
3. Press [ENTER] (after typing in an input box, you must press [ENTER] twice) to save any changes and exit the dialog box.

Saving or Recalling a Viewing Window

After using various **Zoom** tools, you may want to return to a previous viewing window or save the current one.

1. From the [F2] **Zoom** menu, select **B:Memory** to display its submenu.

2. Select the applicable item.

Select:	To:
1:ZoomPrev	Return to the viewing window displayed before the previous zoom.
2:ZoomSto	Save the current viewing window. (The current Window variable values are stored to the system variables **zxmin**, **zxmax**, etc.)
3:ZoomRcl	Recall the viewing window last stored with **ZoomSto**.

Note: You can store only one set of Window variable values at a time. Storing a new set overwrites the old set.

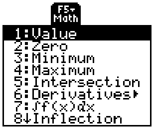
Restoring the Standard Viewing Window

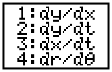
You can restore the Window variables to their default values at any time. From the F2 **Zoom** menu, select **6:ZoomStd**.

Using Math Tools to Analyze Functions

On the Graph screen, the F5 **Math** toolbar menu has several tools that help you analyze graphed functions.

Press [F5] from the Graph screen.





On the Derivatives submenu, only dy/dx is available for function graphing. The other derivatives are available for other graphing modes (parametric, polar, etc.).

Math Tool	Description
Value	Evaluates a selected **y(x)** function at a specified x value.
Zero, Minimum, Maximum	Finds a zero (x-intercept), minimum, or maximum point within an interval.
Intersection	Finds the intersection of two functions.
Derivatives	Finds the derivative (slope) at a point.
∫f(x)dx	Finds the approximate numerical integral over an interval.
Inflection	Finds the inflection point of a curve, where its second derivative changes sign (where the curve changes concavity).
Distance	Draws and measures a line between two points on the same function or on two different functions.
Tangent	Draws a tangent line at a point and displays its equation.
Arc	Finds the arc length between two points along a curve.

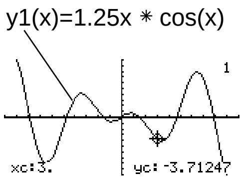
Math Tool	Description
Shade	Depends on the number of functions graphed. • If only one function is graphed, this shades the function’s area above or below the x axis. • If two or more functions are graphed, this shades the area between any two functions within an interval.

Note: For Math results, cursor coordinates are stored in system variables xc and yc (rc and θc if you use polar coordinates). Derivatives, integrals, distances, etc., are stored in the system variable *sysMath*.

Finding y(x) at a Specified Point

1. From the **Graph** screen, press [F5] and select **1:Value**.
2. Type the x value, which must be a real value between **xmin** and **xmax.** The value can be an expression.
3. Press [ENTER].

   The cursor moves to that x value on the first function selected in the Y= Editor, and its coordinates are displayed.



4. Press ⊖ or ⊕ to move the cursor between functions at the entered x value. The corresponding y value is displayed.

   If you press ◀ or ▶, the free-moving cursor appears. You may not be able to move it back to the entered x value.

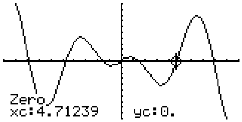
You can also display function coordinates by tracing the function (F3), typing an x value, and pressing ENTER.

Finding a Zero, Minimum, or Maximum within an Interval

1. From the **Graph** screen, press F5 and select **2:Zero**, **3:Minimum**, or **4:Maximum.**
2. As necessary, use ⊙ and ⊙ to select the applicable function.

   Note: Typing x values is a quick way to set bounds.
3. Set the lower bound for x. Either use ⊙ and ⊙ to move the cursor to the lower bound or type its x value.
4. Press ENTER. A ▸ at the top of the screen marks the lower bound.
5. Set the upper bound, and press ENTER.

   The cursor moves to the solution, and its coordinates are displayed.



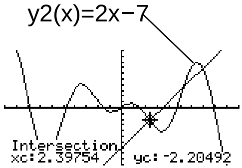
Finding the Intersection of Two Functions within an Interval

1. From the **Graph** screen, press F5 and select **5:Intersection.**
2. Select the first function, using ⊙ or ⊙ as necessary, and press ENTER. The cursor moves to the next graphed function.
3. Select the second function, and press ENTER.
4. Set the lower bound for x. Either use ⊙ and ⊙ to move the cursor to the lower bound or type its x value.

5. Press [ENTER]. A ▶ at the top of the screen marks the lower bound.

6. Set the upper bound, and press [ENTER].

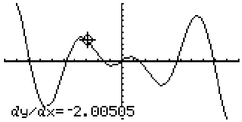
   The cursor moves to the intersection, and its coordinates are displayed.



Finding the Derivative (Slope) at a Point

1. From the **Graph** screen, press [F5] and select **6:Derivatives.** Then select **1:dy/dx** from the submenu.
2. As necessary, use ⊖ and ⊕ to select the applicable function.
3. Set the derivative point. Either move the cursor to the point or type its x value.
4. Press [ENTER].

   The derivative at that point is displayed.



Finding the Numerical Integral over an Interval

1. From the **Graph** screen, press [F5] and select **7:∫f(x)dx.**
2. As necessary, use ⊖ and ⊕ to select the applicable function.

   Note: Typing x values is a quick way to set the limits.

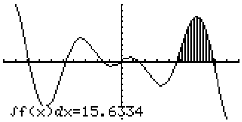
3. Set the lower limit for x. Either use ◀ and ▶ to move the cursor to the lower limit or type its x value.

4. Press [ENTER]. A ▶ at the top of the screen marks the lower limit.

   Note: To erase the shaded area, press [F4] (**ReGraph**).

5. Set the upper limit, and press [ENTER].

   The interval is shaded, and its approximate numerical integral is displayed.



Finding an Inflection Point within an Interval

1. From the **Graph** screen, press [F5] and select **8:Inflection**.

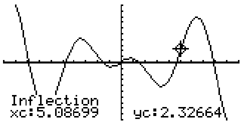
2. As necessary, use ▼ and ▲ to select the applicable function.

3. Set the lower bound for x. Either use ◀ and ▶ to move the cursor to the lower bound or type its x value.

4. Press [ENTER]. A ▶ at the top of the screen marks the lower bound.

5. Set the upper bound, and press [ENTER].

   The cursor moves to the inflection point (if any) within the interval, and its coordinates are displayed.

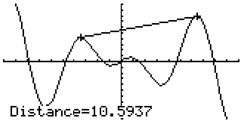


Finding the Distance between Two Points

1. From the **Graph** screen, press [F5] and select **9:Distance**.

2. As necessary, use ⊽ and ⊼ to select the function for the first point.
3. Set the first point. Either use ◁ or ▷ to move the cursor to the point or type its x value.
4. Press [ENTER]. A **+** marks the point.
5. If the second point is on a different function, use ⊽ and ⊼ to select the function.
6. Set the second point. (If you use the cursor to set the point, a line is drawn as you move the cursor.)
7. Press [ENTER].

   The distance between the two points is displayed, along with the connecting line.



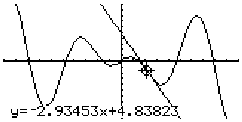
Drawing a Tangent Line

1. From the **Graph** screen, press [F5] and select **A:Tangent.**
2. As necessary, use ⊽ and ⊼ to select the applicable function.

   Note: To erase a drawn tangent line, press [F4] (**ReGraph**).

3. Set the tangent point. Either move the cursor to the point or type its x value.
4. Press [ENTER].

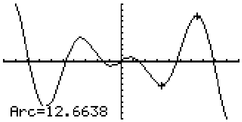
   The tangent line is drawn, and its equation is displayed.



Finding an Arc Length

1. From the **Graph** screen, press [F5] and select **B:Arc**.
2. As necessary, use ⊖ and ⊕ to select the applicable function.
3. Set the first point of the arc. Either use ◀ or ▶ to move the cursor or type the x value.
4. Press [ENTER]. A **+** marks the first point.
5. Set the second point, and press [ENTER].

   A **+** marks the second point, and the arc length is displayed.



Shading the Area between a Function and the x Axis

You must have only one function graphed. If you graph two or more functions, the Shade tool shades the area between two functions.

1. From the **Graph** screen, press [F5] and select **C:Shade.** The screen prompts for **Above X axis?**
2. Select one of the following. To shade the function’s area:
   - Above the x axis, press [ENTER].
   - Below the x axis, press:
     [alpha] **N**

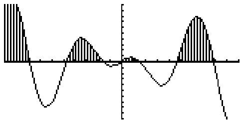
3. Set the lower bound for x. Either use ◀ and ▶ to move the cursor to the lower bound or type its x value.

   Note: If you do not press ◀ or ▶, or type an x value when setting the lower and upper bound, **xmin** and **xmax** will be used as the lower and upper bound, respectively.

4. Press [ENTER]. A ▸ at the top of the screen marks the lower bound.

5. Set the upper bound, and press [ENTER].

   The bounded area is shaded. To erase the shaded area, press [F4] (**ReGraph**).



Shading the Area between Two Functions within an Interval

You must have at least two functions graphed. If you graph only one function, the Shade tool shades the area between the function and the x axis.

1. From the **Graph** screen, press [F5] and select **C:Shade.** The screen prompts for **Above?**
2. As necessary, use ▼ or ▲ to select a function. (Shading will be above this function.)
3. Press [ENTER]. The cursor moves to the next graphed function, and the screen prompts for **Below?**
4. As necessary, use ▼ or ▲ to select another function. (Shading will be below this function.)
5. Press [ENTER].

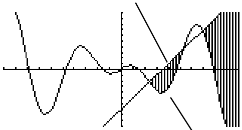
6. Set the lower bound for x. Either use ◀ and ▶ to move the cursor to the lower bound or type its x value.

   Note: If you do not press ◀ or ▶, or type an x value when setting the lower and upper bound, **xmin** and **xmax** will be used as the lower and upper bound, respectively.

7. Press [ENTER]. A ▸ at the top of the screen marks the lower bound.

8. Set the upper bound, and press [ENTER].

   The bounded area is shaded. To erase the shaded area, press [F4] (**ReGraph**).



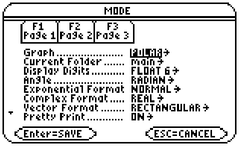
Polar Graphing

Overview of Steps in Graphing Polar Equations

To graph polar equations, use the same general steps used for **y(x)** functions as described in *Basic Function Graphing*. Any differences that apply to polar equations are described on the following pages.

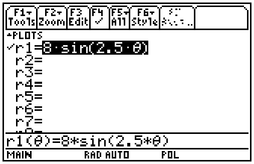
Graphing Polar Equations

1. Set **Graph** mode ([MODE]) to **POLAR**. Also set **Angle** mode, if necessary.



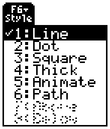
2. Define *x* and y components on Y= Editor ([♦] [Y=]).
3. Select ([F4]) which defined equations to graph. Select the x or y component, or both.

   Note: To turn off any stat data plots press [F5] **5** or use [F4] to deselect them.



4. Set the display style for an equation. You can set either the x or y component.

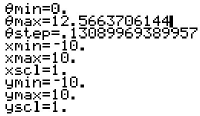
   [2nd] [F6]

   This is optional. For multiple equations, this helps visually distinguish one from another.

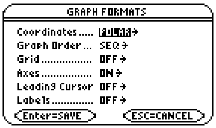
5. Define the viewing window ([♦] [WINDOW]).

   [F2] **Zoom** also changes the viewing window.

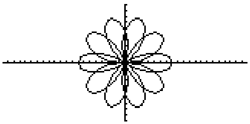
   

6. Change the graph format if necessary.

   [F1] **9**
   – or –
   [♦] [I]

7. Graph the selected equations ([♦] [GRAPH]).

Exploring the Graph

From the Graph screen, you can:

- Display the coordinates of any pixel by using the free-moving cursor, or of a plotted point by tracing a polar equation.

- Use the [F2] **Zoom** toolbar menu to zoom in or out on a portion of the graph.
- Use the [F5] **Math** toolbar menu to find derivatives, tangents, etc. Some menu items are not available for polar graphs.

Differences in Polar and Function Graphing

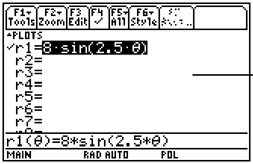
This module assumes that you already know how to graph **y(x)** functions as described in *Basic Function Graphing*. This section describes the differences that apply to polar equations.

Setting the Graph Mode

Use [MODE] to set **Graph = POLAR** before you define equations or set Window variables. The Y= Editor and the Window Editor let you enter information for the current **Graph** mode setting only.

You should also set the **Angle** mode to the units (RADIAN or DEGREE) you want to use for θ.

Defining Polar Equations on the Y= Editor



You can define polar equations for r1(θ) through r99(θ).

You can use the **Define** command from the Home screen (see the *Technical Reference* module) to define functions and equations for any graphing mode, regardless of the current mode.

The Y= Editor maintains an independent function list for each **Graph** mode setting. For example, suppose:

- In FUNCTION graphing mode, you define a set of **y(x)** functions. You change to POLAR graphing mode and define a set of **r(θ)** equations.
- When you return to FUNCTION graphing mode, your **y(x)** functions are still defined in the Y= Editor. When you return to POLAR graphing mode, your **r(θ)** equations are still defined.

Selecting the Display Style

The **Above** and **Below** styles are not available for polar equations and are dimmed on the Y= Editor's **Style** toolbar menu.

Window Variables

The Window Editor maintains an independent set of Window variables for each **Graph** mode setting (just as the Y= Editor maintains independent function lists). Polar graphs use the following Window variables.

Variable	Description
θmin, θmax	Smallest and largest θ values to evaluate.

Variable	Description
θstep	Increment for the θ value. Polar equations are evaluated at: r(θmin) r(θmin+θstep) r(θmin+2(θstep)) ... not to exceed ... r(θmax)
xmin, **xmax**, **ymin**, **ymax**	Boundaries of the viewing window.
xscl, **yscl**	Distance between tick marks on the x and y axes.

Note: You can use a negative θ**step.** If so, θ**min** must be greater than θ**max.**

Standard values (set when you select **6:ZoomStd** from the [F2] **Zoom** toolbar menu) are:

θmin = 0.		xmin = ⁻10.	ymin = ⁻10.
θmax = 2π	(6.2831853... radians or 360 degrees)	xmax = 10.	ymax = 10.
θstep = π/24	(.1308996... radians or 7.5 degrees)	xscl = 1.	yscl = 1.

You may need to change the standard values for the θ variables (θ**min**, θ**max**, θ**step**) to ensure that enough points are plotted.

Setting the Graph Format

To display coordinates as **r** and θ values, use:

[F1] **9**
– or –
[♦] [I]

to set **Coordinates = POLAR**. If **Coordinates = RECT**, the polar equations will be graphed properly, but coordinates will be displayed as x and y.

When you trace a polar equation, the θ coordinate is shown even if **Coordinates = RECT**.

Exploring a Graph

As in function graphing, you can explore a graph by using the following tools. Any displayed coordinates are shown in polar or rectangular form as set in the graph format.

Tool	For Polar Graphs:
Free-Moving Cursor	Works just as it does for function graphs.
[F2] **Zoom**	Works just as it does for function graphs. • Only **x** (**xmin**, **xmax**, **xscl**) and **y** (**ymin**, **ymax**, **yscl**) Window variables are affected. • The θ Window variables (θ**min**, θ**max**, θ**step**) are not affected unless you select **6:ZoomStd** (which sets θ**min = 0**, θ**max = 2**π, and θ**step =** π**/24**).

Tool	For Polar Graphs:
[F3] **Trace**	Lets you move the cursor along a graph one θ**step** at a time. • When you begin a trace, the cursor is on the first selected equation at θ**min**. • QuickCenter applies to all directions. If you move the cursor off the screen (top or bottom, left or right), press [ENTER] to center the viewing window on the cursor location. • Automatic panning is not available. If you move the cursor off the left or right side of the screen, the TI-89 Titanium / Voyage™ 200 graphing calculator will not automatically pan the viewing window. However, you can use QuickCenter.
[F5] **Math**	Only **1:Value**, **6:Derivatives**, **9:Distance**, **A:Tangent**, and **B:Arc** are available for polar graphs. These tools are based on θ values. For example: • **1:Value** displays an **r** value (or x and y, depending on the graph format) for a specified θ value. • **6:Derivatives** finds **dy/dx** or **dr/d**θ at a point defined for a specified θ value.

During a trace, you can also evaluate **r(**θ**)** by typing the θ value and pressing [ENTER].

Note: You can use QuickCenter at any time during a trace, even if the cursor is still on the screen.

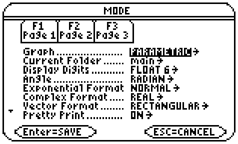
Parametric Graphing

Overview of Steps in Graphing Parametric Equations

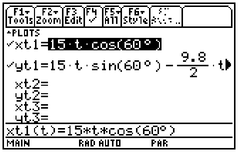
To graph parametric equations, use the same general steps used for **y(x)** functions as described in *Basic Function Graphing*. Any differences that apply to parametric equations are described on the following pages.

Graping Parametric Equations

1. Set **Graph** mode ([MODE]) to **PARAMETRIC**. Also set **Angle** mode, if necessary.



2. Define x and y components on Y= Editor ([♦] [Y=]).



3. Select ([F4]), which defined equations to graph. Select the x or y component, or both.

   Note: To turn off any stat data plots, press [F5] **5** or use [F4] to deselect them.

4. Set the display style for an equation. You can set either the x or y component.

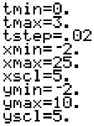
   [2nd] [F6]

   This is optional. For multiple equations, this helps visually distinguish one from another.

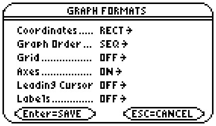
5. Define the viewing window ([♦] [WINDOW]).

   [F2] **Zoom** also changes the viewing window.

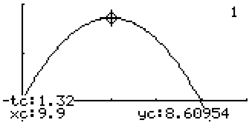
   

6. Change the graph format if necessary.

   [F1] **9**
   – or –
   [♦] [I]

7. Graph the selected equations ([♦] [GRAPH]).

Exploring the Graph

From the Graph screen, you can:

- Display the coordinates of any pixel by using the free-moving cursor, or of a plotted point by tracing a parametric equation.

- Use the [F2] **Zoom** toolbar menu to zoom in or out on a portion of the graph.
- Use the [F5] **Math** toolbar menu to find derivatives, tangents, etc. Some menu items are not available for parametric graphs.

Differences in Parametric and Function Graphing

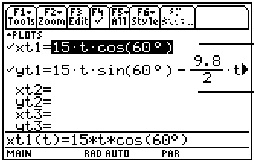
This module assumes that you already know how to graph **y(x)** functions as described in Basic Function Graphing. This section describes the differences that apply to parametric equations.

Setting the Graph Mode

Use [MODE] to set **Graph = PARAMETRIC** before you define equations or set Window variables. The Y= Editor and the Window Editor let you enter information for the *current* **Graph** mode setting only.

Defining Parametric Equations on the Y= Editor

To graph a parametric equation, you must define both its x and y components. If you define only one component, the equation cannot be graphed. (However, you can use single components to generate an automatic table as described in *Tables.*)



Enter x and y components on separate lines.

You can define xt1(t) through xt99(t) and yt1(t) through yt99(t).

Be careful when using implied multiplication with **t.** For example:

Enter:	Instead of:	Because:
t∗cos(60)	tcos(60)	**tcos** is interpreted as a user-defined function called **tcos**, not as implied multiplication. In most cases, this refers to a nonexistent function. So the TI-89 Titanium simply returns the function name, not a number.

Note: When using **t**, be sure implied multiplication is valid for your situation. You can use the **Define** command from the Home screen (see the *Technical Reference* module) to define functions and equations for any graphing mode, regardless of the current mode.

The Y= Editor maintains an independent function list for each **Graph** mode setting. For example, suppose:

- In FUNCTION graphing mode, you define a set of **y(x)** functions. You change to PARAMETRIC graphing mode and define a set of x and *y* components.
- When you return to FUNCTION graphing mode, your **y(x)** functions are still defined in the Y= Editor. When you return to PARAMETRIC graphing mode, your x and y components are still defined.

Selecting Parametric Equations

To graph a parametric equation, select either its x or y component or both. When you enter or edit a component, it is selected automatically.

Selecting x and y components separately can be useful for tables as described in *Tables*. With multiple parametric equations, you can select and compare all the x components or all the y components.

Selecting the Display Style

You can set the style for either the x or y component. For example, if you set the x component to **Dot**, the TI-89 Titanium automatically sets the y component to **Dot.**

Note: Use the **Animate** and **Path** styles for interesting projectile-motion effects.

The **Above** and **Below** styles are not available for parametric equations and are dimmed on the Y= Editor's **Style** toolbar menu.

Window Variables

The Window Editor maintains an independent set of Window variables for each **Graph** mode setting (just as the Y= Editor maintains independent function lists). Parametric graphs use the following Window variables.

Note: You can use a negative **tstep.** If so, **tmin** must be greater than **tmax.**

Variable	Description
tmin, **tmax**	Smallest and largest **t** values to evaluate.
tstep	Increment for the **t** value. Parametric equations are evaluated at: x(tmin) y(tmin) x(tmin+tstep) y(tmin+tstep) x(tmin+2(tstep)) y(tmin+2(tstep)) ... not to exceed ... ... not to exceed ... x(tmax) y(tmax)
xmin, **xmax**, **ymin**, **ymax**	Boundaries of the viewing window.

Variable	Description
xscl, **yscl**	Distance between tick marks on the x and y axes.

Standard values (set when you select **6:ZoomStd** from the [F2] **Zoom** toolbar menu) are:

tmin = 0		xmin = ⁻10.	ymin = ⁻10.
tmax = 2π	(6.2831853... radians or 360 degrees)	xmax = 10.	ymax = 10.
tstep =π/24	(.1308996... radians or 7.5 degrees)	xscl = 1.	yscl = 1.

You may need to change the standard values for the **t** variables (**tmin**, **tmax**, **tstep**) to ensure that enough points are plotted.

Exploring a Graph

As in function graphing, you can explore a graph by using the following tools.

Note: During a trace, you can also evaluate **x(t)** and **y(t)** by typing the **t** value and pressing [ENTER]. You can use QuickCenter at any time during a trace, even if the cursor is still on the screen.

Tool	For Parametric Graphs:
Free-Moving Cursor	Works just as it does for function graphs.

Tool	For Parametric Graphs:
[F2] **Zoom**	Works just as it does for function graphs, with the following exceptions: • Only **x** (**xmin**, **xmax**, **xscl**) and **y** (**ymin**, **ymax**, **yscl**) Window variables are affected. • The **t** Window variables (**tmin**, **tmax**, **tstep**) are not affected unless you select **6:ZoomStd** (which sets **tmin = 0**, **tmax = 2**π, and **tstep =** π**/24**).
[F3] **Trace**	Lets you move the cursor along a graph one **tstep** at a time. • When you begin a trace, the cursor is on the first selected parametric equation at **tmin**. • QuickCenter applies to all directions. If you move the cursor off the screen (top or bottom, left or right), press [ENTER] to center the viewing window on the cursor location. • Automatic panning is not available. If you move the cursor off the left or right side of the screen, the TI-89 Titanium will not automatically pan the viewing window. However, you can use QuickCenter.
[F5] **Math**	Only **1:Value**, **6:Derivatives**, **9:Distance**, **A:Tangent**, and **B:Arc** are available for parametric graphs. These tools are based on **t** values. For example: • **1:Value** displays x and y values for a specified **t** value. • **6:Derivatives** finds **dy/dx**, **dy/dt**, or **dx/dt** at a point defined for a specified **t** value.

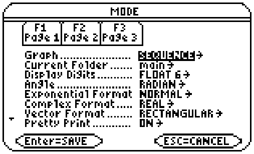
Sequence Graphing

Overview of Steps in Graphing Sequences

To graph sequences, use the same general steps used for **y(x)** functions as described in *Basic Function Graphing*. Any differences are described on the following pages.

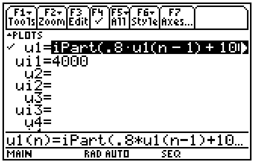
Graphing Sequences

1. Set **Graph** mode ([MODE]) to **SEQUENCE**. Also set **Angle** mode, if necessary.



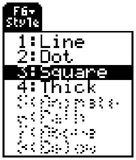
2. Define sequences and, if needed, initial values on Y= Editor ([♦] [Y=]).
3. Select ([F4]) which defined sequences to graph. Do not select initial values.

   Note: To turn off any stat data plots, press [F5] **5** or use [F4] to deselect them.



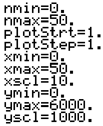
4. Set the display style for a sequence.

   [2nd] [F6]
   For sequences, the default style is **Square**.



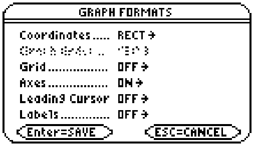
5. Define the viewing window (◆ [WINDOW]).

   F2 **Zoom** also changes the viewing window.

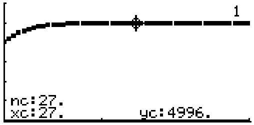


6. Change the graph format if necessary.

   F1 **9**
   — or —
   ◆ F



7. Graph the selected equations (◆ [GRAPH]).



Exploring the Graph

From the Graph screen, you can:

- Display the coordinates of any pixel by using the free-moving cursor, or of a plotted point by tracing a sequence.
- Use the F2 **Zoom** toolbar menu to zoom in or out on a portion of the graph.
- Use the F5 **Math** toolbar menu to evaluate a sequence. Only **1:Value** is available for sequences.
- Plot sequences on **Time** (the default), **Web**, or **Custom** axes.

Note: You can also evaluate a sequence while tracing. Simply enter the n value directly from the keyboard.

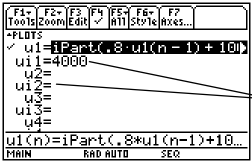
Differences in Sequence and Function Graphing

This module assumes that you already know how to graph **y(x)** functions as described in *Basic Function Graphing*. This section describes the differences that apply to sequences.

Setting the Graph Mode

Use MODE to set **Graph = SEQUENCE** before you define sequences or set Window variables. The Y= Editor and the Window Editor let you enter information for the current Graph mode setting only.

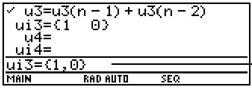
Defining Sequences on the Y= Editor



You can define sequences u1(n) through u99(n).

Use ui only for recursive sequences, which require one or more initial values.

If a sequence requires more than one initial value, enter them as a list enclosed in braces { } and separated by commas. You must use a list to enter two or more initial values.



Enter {1,0} even though {1 0} is shown in the sequence list.

If a sequence requires an initial value but you do not enter one, you will get an error when graphing.

On the Y= Editor, Axes lets you select the axes that are used to graph the sequences. Optionally, for sequences only, you can select different axes for the graph. TIME is the default.

Axes	Description
TIME	Plots n on the x axis and **u(n)** on the y axis.
WEB	Plots **u(n−1)** on the x axis and **u(n)** on the y axis.
CUSTOM	Lets you select the x and y axes.

The Y= Editor maintains an independent function list for each Graph mode setting. For example, suppose:

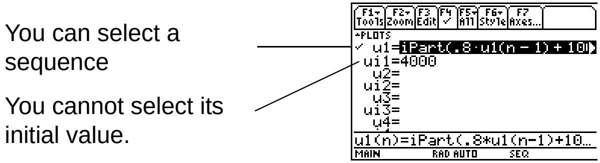
- In FUNCTION graphing mode, you define a set of **y(x)** functions. You change to SEQUENCE graphing mode and define a set of **u(n)** sequences.
- When you return to FUNCTION graphing mode, your **y(x)** functions are still defined in the Y= Editor. When you return to SEQUENCE graphing mode, your **u(n)** sequences are still defined.

Note: You can use the **Define** command from the Home screen (see *Technical Reference*) to define functions and equations for any graphing mode, regardless of the current mode.

Selecting Sequences

With TIME and WEB axes, the TI-89 Titanium graphs only the selected sequences. If you entered any sequences that require an initial value, you must enter the corresponding ui value.

Note: With TIME and CUSTOM axes, all defined sequences are evaluated even if they are not plotted.



With CUSTOM axes, when you specify a sequence in the custom settings, it is graphed regardless of whether it is selected.

Selecting the Display Style

Only the **Line**, **Dot**, **Square**, and **Thick** styles are available for sequence graphs. **Dot** and **Square** mark only those discrete integer values (in plotstep increments) at which a sequence is plotted.

The Window Editor maintains an independent set of Window variables for each Graph mode setting (just as the Y= Editor maintains independent function lists). Sequence graphs use the following Window variables.

Variable	Description
nmin, nmax	Smallest and largest n values to evaluate. Sequences are evaluated at: u(nmin) u(nmin+1) u(nmin+2) ... not to exceed ... u(nmax)
plotStrt	The term number that will be the first one plotted (depending on plotstep). For example, to begin plotting with the 2nd term in the sequence, set **plotstrt = 2**. The first term will be evaluated at **nmin** but not plotted.
plotStep	Incremental n value *for graphing only*. This does not affect how the sequence is evaluated, only which points are plotted. For example, suppose **plotstep = 2**. The sequence is evaluated at each consecutive integer but is plotted at only every other integer.
xmin, xmax, ymin, ymax	Boundaries of the viewing window.
xscl, yscl	Distance between tick marks on the x and y axes.

Note: Both **nmin** and **nmax** must be positive integers, although **nmin** can be zero; **nmin**, **nmax**, **plotstrt** and **plotstep** must be integers ≥ 1. If you do not enter integers, they will be rounded to integers.

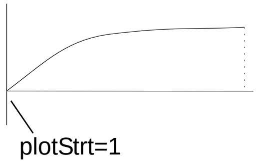
Standard values (set when you select **6:ZoomStd** from the [F2] **Zoom** toolbar menu) are:

nmin = 1.	xmin = -10.	ymin = -10.
nmax = 10.	xmax = 10.	ymax = 10.
plotstrt = 1.	xscl = 1.	yscl = 1.
plotstep = 1.		

You may need to change the standard values for the n and plot variables to ensure that sufficient points are plotted.

To see how **plotstrt** affects graph, look at the following examples of a recursive sequence.

This graph is plotted beginning with the 1st term.



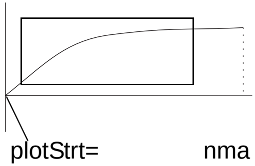
This graph is plotted beginning with the 9th term.

Note: Both of these graphs use the same Window variables, except for **plotstrt.**

With TIME axes (from Axes on the Y= Editor), you can set **plotstrt = 1** and still graph only a selected part of the sequence. Simply define a viewing window that shows only the area of the coordinate plane you want to view.

You could set:

- **xmin** = first n value to graph
- **xmax** = **nmax** (although you can use other values)
- **ymin** and **ymax** = expected values for the sequence



Changing the Graph Format

The Graph Order format is not available.

- With TIME or CUSTOM axes, multiple sequences are always plotted simultaneously.
- With WEB axes, multiple sequences are always plotted sequentially.

Exploring a Graph

As in function graphing, you can explore a graph by using the following tools. Any displayed coordinates are shown in rectangular or polar form as set in the graph format.

Tool	For Sequence Graphs:
Free-Moving Cursor	Works just as it does for function graphs.

Tool	For Sequence Graphs:
[F2] **Zoom**	Works just as it does for function graphs. • Only **x** (**xmin**, **xmax**, **xscl**) and **y** (**ymin**, **ymax**, **yscl**) Window variables are affected. • The n and plot Window variables (**nmin**, **nmax**, **plotStrt**, **plotStep**) are not affected unless you select **6:ZoomStd** (which sets all Window variables to their standard values).
[F3] **Trace**	Depending on whether you use TIME, CUSTOM, or WEB axes, Trace operates very differently. • With TIME or CUSTOM axes, you move the cursor along the sequence one plotstep at a time. To move approximately ten plotted points at a time, press [2nd] ▶ or [2nd] ◀. - When you begin a trace, the cursor is on the first selected sequence at the term number specified by plotstrt, even if it is outside the viewing window. - QuickCenter applies to all directions. If you move the cursor off the screen (top or bottom, left or right), press [ENTER] to center the viewing window on the cursor location. • With WEB axes, the trace cursor follows the web, not the sequence.
[F5] **Math**	Only **1:Value** is available for sequence graphs. • With TIME and WEB axes, the **u(n)** value (represented by **yc**) is displayed for a specified n value. • With CUSTOM axes, the values that correspond to x and y depend on the axes you choose.

During a trace, you can evaluate a sequence by typing a value for n and pressing [ENTER]. You can use QuickCenter at any time during a trace, even if the cursor is still on the screen.

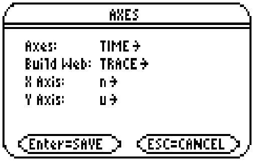
Setting Axes for Time, Web, or Custom Plots

For sequences only, you can select different types of axes for the graph. Examples of the different types are given later in this module.

Displaying the AXES Dialog Box

From the Y= Editor, Axes:

- Depending on the current Axes setting, some items may be dimmed.
- To exit without making any changes, press [ESC].



Item	Description
Axes	TIME — Plots **u(n)** on the y axis and n on the x axis. WEB — Plots **u(n)** on the y axis and **u(n−1)** on the x axis. CUSTOM — Lets you select the x and y axes.
Build Web	Active only when Axes = WEB, this specifies whether a web is drawn manually (TRACE) or automatically (AUTO).

Item	Description
X Axis and **Y Axis**	Active only when Axes = CUSTOM, these let you select the value or sequence to plot on the x and y axes.

To change any of these settings, use the same procedure that you use to change other types of dialog boxes, such as the MODE dialog box.

Using Web Plots

A web plot graphs **u(n)** vs. **u(n−1)**, which lets you study the long-term behavior of a recursive sequence. The examples in this section also show how the initial value can affect a sequence's behavior.

Valid Functions for Web Plots

A sequence must meet the following criteria; otherwise, it will not be graphed properly on WEB axes. The sequence:

- Must be recursive with only one recursion level; **u(n−1)** but not **u(n−2)**.
- Cannot reference n directly.
- Cannot reference any other defined sequence except itself.

When You Display the Graph Screen

After you select WEB axes and display the Graph screen, the TI-89 Titanium:

- Draws a **y=x** reference line.
- Plots the selected sequence definitions as functions, with **u(n−1)** as the independent variable. This effectively converts a recursive sequence into a nonrecursive form for graphing.

For example, consider the sequence u1(n) = $\sqrt{5-u1(n-1)}$ and an initial value of **ui1=1**.

The TI-89 Titanium draws the **y=x** reference line and then plots y = $y = \sqrt{5-x}$.

Drawing the Web

After the sequence is plotted, the web may be displayed manually or automatically, depending on how you set **Build Web** on the AXES dialog box.

If Build Web =	The web is:
TRACE	Not drawn until you press [F3]. The web is then drawn step-by-step as you move the trace cursor (you must have an initial value before using Trace). **Note**: With WEB axes, you cannot trace along the sequence itself as you do in other graphing modes.
AUTO	Drawn automatically. You can then press [F3] to trace the web and display its coordinates.

The web:

1. Starts on the x axis at the initial value ui (when **plotstrt = 1**).
2. Moves vertically (either up or down) to the sequence.

3. Moves horizontally to the **y=x** reference line.
4. Repeats this vertical and horizontal movement until **n=nmax**.

Note: The web starts at **plotstrt**. The value of n is incremented by 1 each time the web moves to the sequence (**plotStep** is ignored).

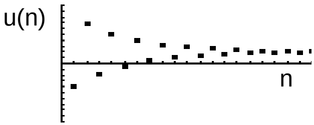
Example: Convergence

1. On the Y= Editor (◆ [Y=]), define u1(n) = ⁻.8u1(n−1) + 3.6. Set initial value ui1 = ⁻4.
2. Set **Axes = TIME**.
3. On the Window Editor (◆ [WINDOW]), set the Window variables.

nmin=1	xmin=0	ymin=⁻10
nmax=25	xmax=25	ymax=10
plotstrt=1	xscl=1	yscl=1
plotstep=1		

4. Graph the sequence (◆ [GRAPH]).

   By default, a sequence uses the **Square** display style.



5. On the Y= Editor, set **Axes = WEB** and **Build Web = AUTO**.

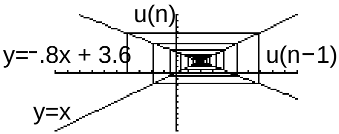
6. On the Window Editor, change the Window variables.

nmin=1	xmin=-10	ymin=-10
nmax=25	xmax=10	ymax=10
plotstrt=1	xscl=1	yscl=1
plotstep=1		

7. Regraph the sequence.

   Web plots are always shown as lines, regardless of the selected display style.

   Note: During a trace, you can move the cursor to a specified n value by typing the value and pressing [ENTER].



8. Press [F3]. As you press ▶, the trace cursor follows the web. The screen displays the cursor coordinates nc, xc, and yc (where xc and yc represent **u(n−1)** and **u(n)**, respectively).

As you trace to larger values of nc, you can see xc and yc approach the convergence point.

Note: When the nc value changes, the cursor is on the sequence. The next time you press ▶, nc stays the same but the cursor is now on the **y=x** reference line.

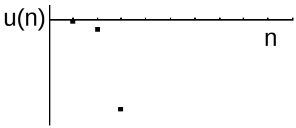
Example: Divergence

1. On the Y= Editor ([♦] [Y=]), define u1(n) = 3.2u1(n−1) − .8(u1(n−1))^2. Set initial value ui1 = 4.45.
2. Set **Axes = TIME**.
3. On the Window Editor ([♦] [WINDOW]), set the Window variables.

nmin=0	xmin=0	ymin=⁻75
nmax=10	xmax=10	ymax=10
plotstrt=1	xscl=1	yscl=1
plotstep=1		

4. Graph the sequence ([♦] [GRAPH]).

   Because the sequence quickly diverges to large negative values, only a few points are plotted.

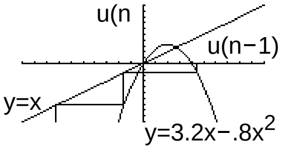


5. On the Y= Editor, set **Axes = WEB** and **Build Web = AUTO**.
6. On the Window Editor ([♦] [WINDOW]), set the Window variables.

nmin=0	xmin=⁻10	ymin=⁻10
nmax=10	xmax=10	ymax=10
plotstrt=1	xscl=1	yscl=1
plotstep=1		

7. Regraph the sequence.

   The web plot shows how quickly the sequence diverges to large negative values.



Example: Oscillation

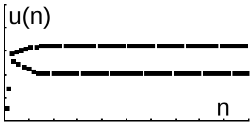
This example shows how the initial value can affect a sequence.

1. On the Y= Editor (◆ [Y=]), use the same sequence defined in the divergence example: u1(n) = 3.2u1(n−1) − .8(u1(n−1))^2. Set initial value ui1 = 0.5.
2. Set **Axes = TIME**.
3. On the Window Editor (◆ [WINDOW]), set the Window variables.

nmin=1	xmin=0	ymin=0
nmax=100	xmax=100	ymax=5
plotstrt=1	xscl=10	yscl=1
plotstep=1		

4. Graph the sequence (◆ [GRAPH]).

   Note: Compare this graph with the divergence example. This is the same sequence with a different initial value.



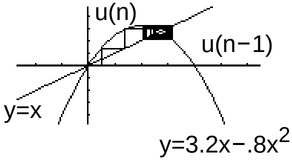
5. On the Y= Editor, set **Axes = WEB** and **Build Web = AUTO**.

6. On the Window Editor ([♦] [WINDOW]), set the Window variables.

nmin=1	xmin=2.68	ymin=4.7
nmax=100	xmax=6.47	ymax=47
plotstrt=1	xscl=1	yscl=1
plotstep=1		

7. Regraph the sequence.

   Note: The web moves to an orbit oscillating between two stable points.

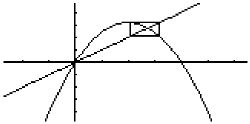


8. Press [F3]. Then use ⊙ to trace the web.

   As you trace to larger values of nc, notice that xc and yc oscillate between 2.05218 and 3.19782.

9. On the Window Editor, set **plotstrt=50**. Then regraph the sequence.

   Note: By starting the web plot at a later term, the stable oscillation orbit is shown more clearly.



Using Custom Plots

CUSTOM axes give you great flexibility in graphing sequences. As shown in the following example, CUSTOM axes are particularly effective for showing relationships between one sequence and another.

Using the predator-prey model in biology, determine the numbers of rabbits and foxes that maintain population equilibrium in a certain region.

R	=	Number of rabbits
M	=	Growth rate of rabbits if there are no foxes (use .05)
K	=	Rate at which foxes can kill rabbits (use .001)
W	=	Number of foxes
G	=	Growth rate of foxes if there are rabbits (use .0002)
D	=	Death rate of foxes if there are no rabbits (use .03)

Rn	=	$R_{n-1} (1 + M - K W_{n-1})$
Wn	=	$W_{n-1} (1 + G R_{n-1} - D)$

1. On the Y= Editor ([♦] [Y=]), define the sequences and initial values for $\mathbf{R_n}$ and $\mathbf{W_n}$.

   u1(n) = u1(n−1) ∗ (1 + .05 − .001 ∗ u2(n−1))
   ui1 = 200
   u2(n) = u2(n−1) ∗ (1 + .0002 ∗ u1(n−1) − .03)
   ui2 = 50

   Note: Assume there are initially 200 rabbits and 50 foxes.

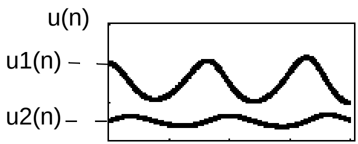
2. Set **Axes = TIME**.

3. On the Window Editor ([♦] [WINDOW]), set the Window variables.

nmin=0	xmin=0	ymin=0
nmax=400	xmax=400	ymax=300
plotstrt=1	xscl=100	yscl=100
plotstep=1		

4. Graph the sequence ([♦] [GRAPH]).

   Note: Use [F3] to individually trace the number of rabbits **u1(n)** and foxes **u2(n)** over time **(n)**.

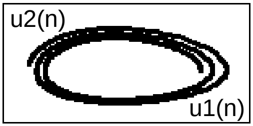


5. On the Y= Editor, set **Axes = CUSTOM**, **X Axis = u1**, and **Y Axis = u2**.
6. On the Window Editor ([♦] [WINDOW]), set the Window variables.

nmin=0	xmin=84	ymin=25
nmax=400	xmax=237	ymax=75
plotstrt=1	xscl=50	yscl=10
plotstep=1		

7. Regraph the sequence.

   Note: Use [F3] to trace both the number of rabbits (xc) and foxes (yc) over the cycle of 400 generations.



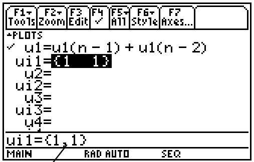
Using a Sequence to Generate a Table

Previous sections described how to graph a sequence. You can also use a sequence to generate a table. Refer to *Tables* for detailed information.

Example: Fibonacci Sequence

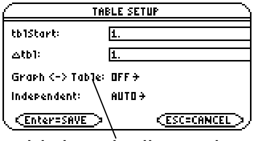
In a Fibonacci sequence, the first two terms are 1 and 1. Each succeeding term is the sum of the two immediately preceding terms.

1. On the Y= Editor (◆ [Y=]), define the sequence and set the initial values as shown.



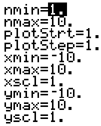
You must enter {1,1}, although {1 1} is shown in the sequence list.

2. Set table parameters (◆ [TBLSET]) to:
tblStart = 1
Δtbl = 1
Independent = AUTO

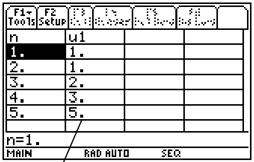


This item is dimmed if you are not using TIME axes.

3. Set Window variables (◆ [WINDOW]) so that **nmin** has the same value as **tblStart**.



4. Display the table (◆ [TABLE]).



Fibonacci sequence is in column 2.

5. Scroll down the table (⊖ or [2nd] ⊖) to see more of the sequence.

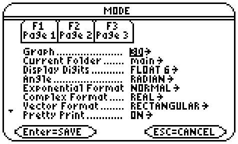
3D Graphing

Overview of Steps in Graphing 3D Equations

To graph 3D equations, use the same general steps used for **y(x)** functions as described in *Basic Function Graphing*. Any differences that apply to 3D equations are described on the following pages.

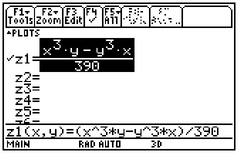
Graphing 3D Equations

1. Set Graph mode ([MODE]) to **3D**. Also set **Angle** mode, if necessary.



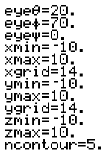
2. Define 3D equations on Y= Editor ([♦] [Y=]).
3. Select ([F4]) which equation to graph. You can select only one 3D equation.

   To turn off any stat data plots, press [F5] **5** or use [F4] to deselect them.



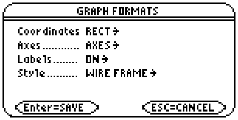
4. Define the viewing cube ([♦] [WINDOW]).

   For 3D graphs, the viewing window is called the viewing cube. [F2] **Zoom** also changes the viewing cube.



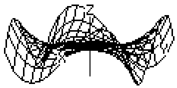
5. Change the graph format if necessary.

   [F1] **9**
   – or -
   [♦] [|]
   Note: To help you see the orientation of 3D graphs, turn on **Axes and Labels**.



6. Graph the selected equations ([♦] [GRAPH]).

   Note: Before displaying the graph, the screen shows the “percent evaluated.”



Exploring the Graph

From the Graph screen, you can:

- Trace the equation.
- Use the [F2] **Zoom** toolbar menu to zoom in or out on a portion of the graph. Some of the menu items are dimmed because they are not available for 3D graphs.
- Use the [F5] **Math** toolbar menu to evaluate the equation at a specified point. Only **1:Value** is available for 3D graphs.

You can also evaluate **z(x,y)** while tracing. Type the x value and press [ENTER]; then type the y value and press [ENTER].

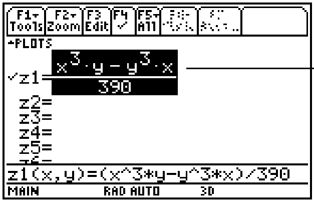
Differences in 3D and Function Graphing

This module assumes that you already know how to graph **y(x)** functions as described in Basic Function Graphing. This section describes the differences that apply to 3D equations.

Setting the Graph Mode

Use [MODE] to set **Graph = 3D** before you define equations or set Window variables. The Y= Editor and the Window Editor let you enter information for the current Graph mode setting only.

Defining 3D Equations on the Y= Editor



You can define 3D equations for **z1(x,y)** through **z99(x,y)**.

The Y= Editor maintains an independent function list for each Graph mode setting. For example, suppose:

- In FUNCTION graphing mode, you define a set of **y(x)** functions. You change to 3D graphing mode and define a set of **z(x,y)** equations.

- When you return to FUNCTION graphing mode, your **y(x)** functions are still defined in the Y= Editor. When you return to 3D graphing mode, your **z(x,y)** equations are still defined.

Note: You can use the **Define** command from the Home screen (see the Technical Reference module) to define functions and equations for any graphing mode, regardless of the current mode.

Selecting the Display Style

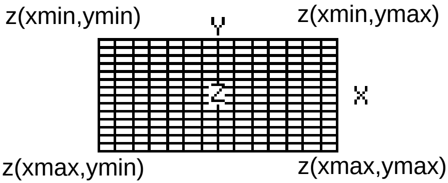
Because you can graph only one 3D equation at a time, display styles are not available. On the Y= Editor, the Style toolbar menu is dimmed.

For 3D equations, however, you can use:
[F1] **9**
– or -
[♦] [I]
to set the Style format to WIRE FRAME or HIDDEN SURFACE.

Window Variables

The Window Editor maintains an independent set of Window variables for each Graph mode setting (just as the Y= Editor maintains independent function lists). 3D graphs use the following Window variables.

Variable	Description
eyeθ, eyeφ, eyeψ	Angles (always in degrees) used to view the graph.

Variable	Description
xmin, **xmax**, **ymin**, **ymax**, **zmin**, **zmax**	Boundaries of the viewing cube.
xgrid, **ygrid**	The distance between **xmin** and **xmax** and between **ymin** and **ymax** is divided into the specified number of grids. The **z(x,y)** equation is evaluated at each grid point where the grid lines (or grid wires) intersect. The incremental value along x and y is calculated as: $\text{x increment} = \frac{\text{xmax} - \text{xmin}}{\text{xgrid}}$ $\text{y increment} = \frac{\text{ymax} - \text{ymin}}{\text{ygrid}}$ The number of grid wires is **xgrid + 1** and **ygrid + 1**. For example, when **xgrid = 14** and **ygrid = 14**, the xy grid consists of 225 (15 × 15) grid points. 
ncontour	The number of contours evenly distributed along the displayed range of z values.

Note: If you enter a fractional number for **xgrid** or **ygrid**, it is rounded to the nearest whole number ≥ 1. The 3D mode does not have **scl** Window variables, so you cannot set tick marks on the axes.

Standard values (set when you select **6:ZoomStd** from the [F2] **Zoom** toolbar menu) are:

eyeθ = 20.	xmin = ⁻10.	ymin = ⁻10.	zmin = ⁻10.
eyeϕ = 70.	xmax = 10.	ymax = 10.	zmax = 10.
eyeψ = 0.	xgrid = 14.	ygrid = 14.	ncontour = 5.

You may need to increase the standard values for the grid variables (**xgrid**, **ygrid**) to ensure that enough points are plotted.

Note: Increasing the grid variables decreases the graphing speed.

Setting the Graph Format

The Axes and Style formats are specific to the 3D graphing mode.

Exploring a Graph

As in function graphing, you can explore a graph by using the following tools. Any displayed coordinates are shown in rectangular or cylindrical form as set in the graph format. In 3D graphing, cylindrical coordinates are shown when you use use:
[F1] **9**
– or –

[♦] [I]
to set **Coordinates = POLAR**.

Tool	For 3D Graphs:
Free-Moving Cursor	The free-moving cursor is not available.
[F2] **Zoom**	Works essentially the same as it does for function graphs, but remember that you are now using three dimensions instead of two. • Only the following zooms are available: **2:ZoomIn**, **3:ZoomOut**, **5:ZoomSqr**, **6:ZoomStd**, **A:ZoomFit**, **B:Memory**, **C:SetFactors** • Only **x** (**xmin**, **xmax**), **y** (**ymin**, **ymax**), and **z** (**zmin**, **zmax**) Window variables are affected. • The **grid** (**xgrid**, **ygrid**) and **eye** (**eye**θ, **eye**φ, **eye**ψ) Window variables are not affected unless you select **6:ZoomStd** (which resets these variables to their standard values).
[F3] **Trace**	Lets you move the cursor along a grid wire from one grid point to the next on the 3D surface. • When you begin a trace, the cursor appears at the midpoint of the xy grid. • QuickCenter is available. At any time during a trace, regardless of the cursor's location, you can press [ENTER] to center the viewing cube on the cursor. • Cursor movement is restricted in the x and y directions. You cannot move the cursor beyond the viewing cube boundaries set by **xmin**, **xmax**, **ymin**, and **ymax**.

Tool	For 3D Graphs:
[F5] **Math**	Only **1:Value** is available for 3D graphs. This tool displays the z value for a specified x and y value. After selecting **1:Value**, type the x value and press [ENTER]. Then type the y value and press [ENTER].

Note: During a trace, you can also evaluate **z(x,y)**. Type the x value and press [ENTER]; then type the y value and press [ENTER].

Moving the Cursor in 3D

When you move the cursor along a 3D surface, it may not be obvious why the cursor moves as it does. 3D graphs have two independent variables (**x,y**) instead of one, and the x and y axes have a different orientation than other graphing modes.

How to Move the Cursor

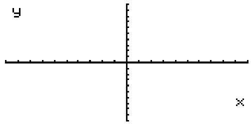
On a 3D surface, the cursor always follows along a grid wire.

Cursor Key	Moves the cursor to the next grid point in the:
▶	Positive x direction
◀	Negative x direction
▲	Positive y direction
▼	Negative y direction

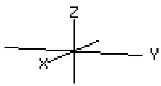
Note: You can move the cursor only within the x and y boundaries set by Window variables **xmin**, **xmax**, **ymin**, and **ymax**.

Although the rules are straightforward, the actual cursor movement can be confusing unless you know the orientation of the axes.

In 2D graphing, the x and y axes always have the same orientation relative to the Graph screen.



In 3D graphing, x and y have a different orientation relative to the Graph screen. Also, you can rotate and/or elevate the viewing angle.



eyeθ=20 eyeϕ=70 eyeψ=0

Note: To show the axes and their labels from the Y= Editor, Window Editor, or Graph screen, use:
[♦] [I]

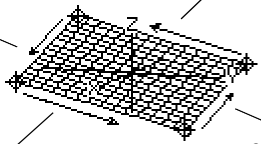
Simple Example of Moving the Cursor

The following graph shows a sloped plane that has the equation $z1(x,y) = -(x + y) / 2$. Suppose you want to trace around the displayed boundary.

When you press [F3], the trace cursor appears at the midpoint of the xy grid. Use the cursor pad to move the cursor to any edge.

moves in a positive x direction, up to xmax.

moves in a negative y direction, back to ymin.



moves in a positive y direction, up to ymax.

moves in a negative x direction, back to xmin.

By displaying and labeling the axes, you can more easily see the pattern in the cursor movement. To move grid points closer together, you can increase Window variables **xgrid** and **ygrid**.

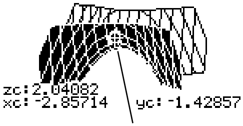
When the trace cursor is on an interior point in the displayed plane, the cursor moves from one grid point to the next along one of the grid wires. You cannot move diagonally across the grid. Notice that the grid wires may not appear parallel to the axes.

Example of the Cursor on a Hidden Surface

On more complex shapes, the cursor may appear as if it is not on a grid point. This is an optical illusion caused when the cursor is on a hidden surface.

For example, consider a saddle shape z1(x,y) = $(x^2-y^2) / 3$. The following graph shows the view looking down the y axis.

Now look at the same shape at 10° from the x axis (eyeθ = 10).



You can move the cursor so that it does not appear to be on a grid point.

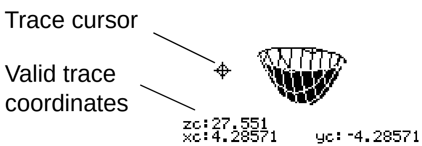
If you cut away the front side, you can see the cursor is actually on a grid point on the hidden back side.

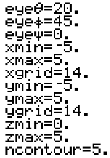
Note: To cut away the front of the saddle in this example, set **xmax=0** to show only negative x values.

Example of an "Off the Curve" Cursor

Although the cursor can move only along a grid wire, you will see many cases where the cursor does not appear to be on the 3D surface at all. This occurs when the z axis is too short to show **z(x,y)** for the corresponding x and y values.

For example, suppose you trace the paraboloid $z(x,y) = x^2 + .5y^2$ graphed with the indicated Window variables. You can easily move the cursor to a position such as:





Although the cursor is actually tracing the paraboloid, it appears off the curve because the trace coordinates:

- **xc** and **yc** are within the viewing cube.

  – but –

- **zc** is outside the viewing cube.

Note: QuickCenter lets you center the viewing cube on the cursor's location. Simply press [ENTER].

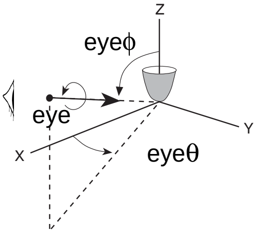
When **zc** is outside the z boundary of the viewing cube, the cursor is physically displayed at **zmin** or **zmax** (although the screen shows the correct trace coordinates).

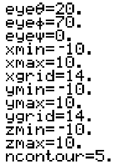
Rotating and/or Elevating the Viewing Angle

In 3D graphing mode, the **eye**θ and **eye**ϕ Window variables let you set viewing angles that determine your line of sight. The **eye**ψ Window variable lets you rotate the graph around that line of sight.

The viewing angle has three components:

- **eye**θ — angle in degrees from the positive x axis.
- **eye**ϕ — angle in degrees from the positive z axis.
- **eye**ψ — angle in degrees by which the graph is rotated counter-clockwise around the line of sight set by **eye**θ and **eye**ϕ.





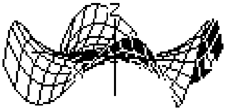
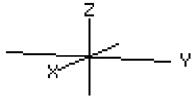
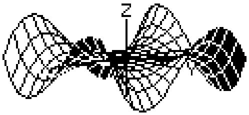
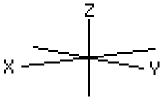
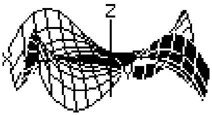
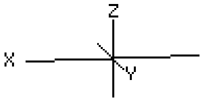
Do not enter a °symbol. For example, type 20, 70, and 0, not 20°, 70° and 0°.

Note: When **eye**ψ**=0**, the z axis is vertical on the screen. When **eye**ψ**=90**, the z axis is rotated 90° counterclockwise and is horizontal.

In the Window Editor (◆ [WINDOW]), always enter **eye**θ, **eye**ϕ, and **eye**ψ in degrees, regardless of the current angle mode.

Effect of Changing eye

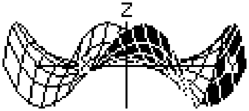
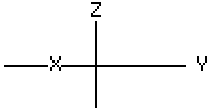
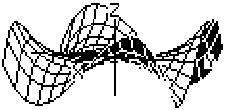
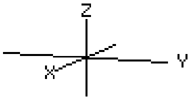
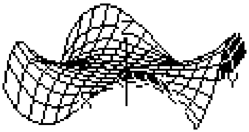
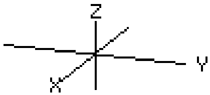
The view on the Graph screen is always oriented along the viewing angle. From this point of view, you can change **eye**θ to rotate the viewing angle around the z axis.

$z1(x,y) = (x^3y - y^3x) / 390$		**In this example eyeφ = 70**
	eyeθ = 20	
	eyeθ = 50	
	eyeθ = 80	

Note: This example increments **eye**θ by 30.

Effect of Changing eye

By changing **eyeφ**, you can elevate your viewing angle above the xy plane. If **90 < eyeφ < 270**, the viewing angle is below the xy plane.

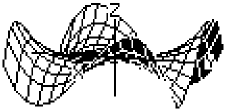
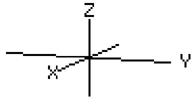
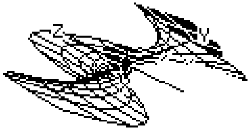
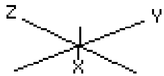
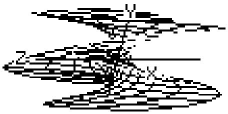
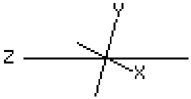
$z1(x,y) = (x^3y - y^3x) / 390$		**In this example eyeθ = 20**
	eyeφ = 90	
	eyeφ = 70	
	eyeφ = 50	

Note: This example starts on the xy plane (**eyeφ = 90**) and decrements **eyeφ** by 20 to elevate the viewing angle.

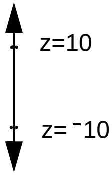
Effect of Changing eye

The view on the Graph screen is always oriented along the viewing angles set by **eyeθ** and **eyeφ**. You can change **eyeψ** to rotate the graph around that line of sight.

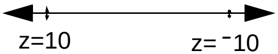
Note: During rotation, the axes expand or contract to fit the screen's width and height. This causes some distortion as shown in the example.

$z1(x,y)=(x^3y-y^3x) / 390$		**In this example, eyeθ=20 and eyeφ=70**
	eyeψ = 0	
	eyeψ = 45	
	eyeψ = 90	

When **eyeψ=0**, the z axis runs the height of the screen.



When **eyeψ=90**, the z axis runs the width of the screen.



As the z axis rotates 90°, its range ($^{-}$10 to 10 in this example) expands to almost twice its original length. Likewise, the x and y axes expand or contract.

From the Home Screen or a Program

The **eye** values are stored in the system variables **eye**θ, **eye**φ, and **eye**ψ. You can access or store to these variables as necessary.

To type φ or ψ, press [♦] [(] [alpha] **[F]** or [♦] [(] [Y], respectively. You can also press [2nd] [CHAR] and use the Greek menu.

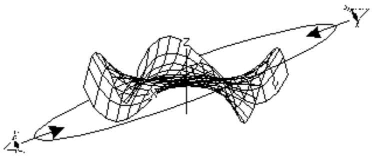
Animating a 3D Graph Interactively

After plotting any 3D graph, you can change the viewing angle interactively by using the cursor.

The Viewing Orbit

When using ◀ and ▶ to animate a graph, think of it as moving the viewing angle along its “viewing orbit” around the graph.

Moving along this orbit can cause the z axis to wobble slightly during the animation.



Note: The viewing orbit affects the eye Window variables in differing amounts.

Animating the Graph

To:	Do this:
Animate the graph incrementally.	Press and release the cursor quickly.
Move along the viewing orbit.	◀ or ▶
Change the viewing orbit's elevation. (primarily increases or decreases eyeϕ)	▲ or ▼
Animate the graph continuously.	Press and hold the cursor for about 1 second, and then release it. To stop, press [ESC], [ENTER], [ON], or [♦] [␣] (space).
Change between 4 animation speeds (increase or decrease the incremental changes in the eye Window variables).	Press [+] or [−].
Change the viewing angle of a non-animated graph to look along the x, y, or z axis.	Press X, Y or Z, respectively.
Return to the initial eye angle values.	Press 0 (zero).

Notes: If the graph is shown in expanded view, it returns to normal view automatically when you press a cursor key.

- After animating the graph, you can stop and then re-start the animation in the same direction by pressing:
  [ENTER] or [alpha] [␣]
- During an animation, you can switch to the next graph format style by pressing:
  [I]
- You can view a graphic that shows the eye angles.

Animating a Series of Graph Pictures

You can also animate a graph by saving a series of graph pictures and then flipping (or cycling) through those pictures. Refer to “Animating a Series of Graph Pictures” Additonal Graphing Topics. This method gives you more control over the Window variable values, particularly **eye**ψ, which rotates the graph.

Changing the Axes and Style Formats

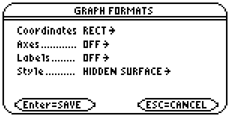
With its default settings, the TI-89 Titanium displays hidden surfaces on a 3D graph but does not display the axes. However, you can change the graph format at any time.

Displaying the GRAPH FORMATS Dialog Box

From the Y= Editor, Window Editor, or Graph screen, press:
[F1] **9**

– or -
[♦] [I]

- The dialog box shows the current graph format settings.
- To exit without making a change, press [ESC].



To change any of these settings, use the same procedure that you use to change other types of dialog boxes, such as the MODE dialog box.

Examples of Axes Settings

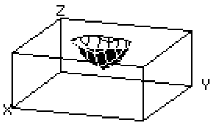
To display the valid **Axes** settings, highlight the current setting and press ▶.

z1(x,y) = x^2+.5y^2

- AXES — Shows standard xyz axes.

- BOX — Shows 3-dimensional box axes.

  The edges of the box are determined by the Window variables **xmin**, **xmax**, etc.



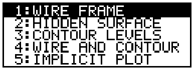
In many cases, the origin (0,0,0) is inside the box, not at a corner. For example, if **xmin = ymin = zmin = ⁻10** and **xmax = ymax = zmax = 10**, the origin is at the center of the box.

Note: Setting Labels = ON is helpful when you display either type of 3D axes.

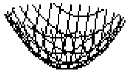
Examples of Style Settings

Note: WIRE FRAME is faster to graph and may be more convenient when you're experimenting with different shapes.

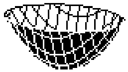
To display the valid Style settings, highlight the current setting and press ▶.



- WIRE FRAME — Shows the 3D shape as a transparent wire frame.



- HIDDEN SURFACES — Uses shading to differentiate the two sides of the 3D shape.

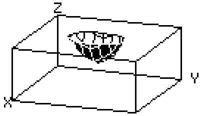
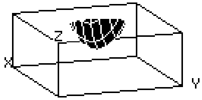
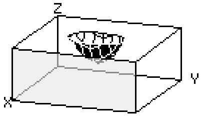
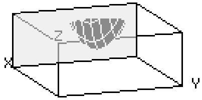


Later sections in this module describe CONTOUR LEVELS, WIRE AND COUNTOUR, and implicit plots.

Be Aware of Possible Optical Illusions

The eye angles used to view a graph (**eye**θ, **eye**ϕ, and **eye**ψ Window variables) can result in optical illusions that cause you to lose perspective on a graph. Typically, most optical illusions occur when the eye angles are in a negative quadrant of the coordinate system.

Optical illusions may be more noticeable with box axes. For example, it may not be immediately obvious which is the “front” of the box.

Looking down from above the xy plane	Looking up from below the xy plane
	
eyeθ = 20, eyeϕ = 55, eyeψ = 0	eyeθ = 20, eyeϕ = 120, eyeψ= 0
	

Note: The first two examples show the graphs as displayed on the screen. The second two examples use artificial shading (which is not displayed on the screen) to show the front of the box.

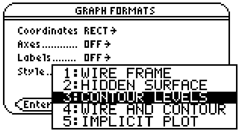
To minimize the effect of optical illusions, use the GRAPH FORMATS dialog box to set Style = HIDDEN SURFACE.

Contour Plots

In a contour plot, a line is drawn to connect adjacent points on the 3D graph that have the same z value. This module discusses the CONTOUR LEVELS and WIRE AND CONTOUR graph format styles.

In 3D graphing mode, define an equation and graph it as you would any 3D equation, with the following exception. Display the GRAPH FORMATS dialog box by pressing [F1] **9** from the Y= Editor, Window editor, or Graph screen. Then set:

Style = CONTOUR LEVELS
– or –
Style = WIRE AND CONTOUR

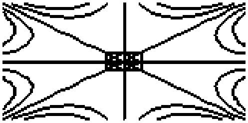
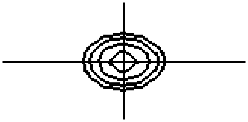
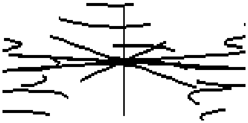
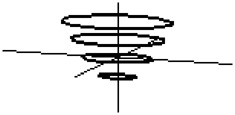
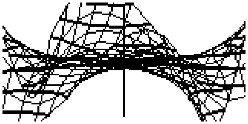
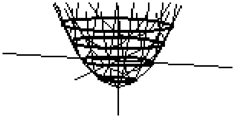


- For CONTOUR LEVELS, only the contours are shown.
  - The viewing angle is set initially so that you are viewing the contours by looking down the z axis. You can change the viewing angle as necessary.
  - The graph is shown in expanded view. To switch between expanded and normal view, press [×].
  - The Labels format is set to OFF automatically.
- For WIRE AND CONTOUR, the contours are drawn on a wire frame view. The viewing angle, view (expanded or normal), and Labels format retain their previous settings.

Notes:

- From the Graph screen, you can switch from one graph format style to the next (skipping IMPLICIT PLOT) by pressing:
  [I]
- Pressing:
  [I]
  to select CONTOUR LEVELS does not affect the viewing angle, view, or Labels

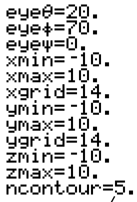
format as it does if you use:
[♦] [I]

Style	z1(x,y)=(x^3y-y^3x) / 390	z1(x,y)=$x^2+.5y^2-5$
	Looking down z axis	
CONTOUR LEVELS		
	Using eyeθ=20, eyeϕ=70, eyeψ=0	
CONTOUR LEVELS		
WIRE AND CONTOUR		

Note: These examples use the same x, y, and z Window variable values as a **ZoomStd** viewing cube. If you use **ZoomStd**, press Z to look down the z axis. Do not confuse the contours with the grid lines. The contours are darker.

You can set the ncontour Window variable (♦ [WINDOW]) to specify the number of contours that will be evenly distributed along the displayed range of z values, where:

$$\text{increment} = \frac{\text{zmax} - \text{zmin}}{\text{ncontour} + 1}$$



The default is 5. You can set this to 0 through 20.

The z values for the contours are:

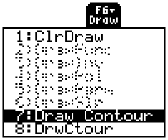
zmin + increment
zmin + 2(increment)
zmin + 3(increment)
⋮
zmin + ncontour(increment)

If **ncontour=5** and you use the standard viewing window (**zmin=-10** and **zmax=10**) , the increment is 3.333. Five contours are drawn for z=-6.666, -3.333, 0, 3.333, and 6.666.

Note, however, that a contour is not drawn for a z value if the 3D graph is not defined at that z value.

If a contour graph is currently displayed, you can specify a point on the graph and draw a contour for the corresponding z value.

1. To display the **Draw** menu, press:
   [2nd] [F6]



2. Select **7:Draw Contour.**
3. Either:
   - Type the point's x value and press [ENTER], and then type the y value and press [ENTER].

     – or –

   - Move the cursor to the applicable point. (The cursor moves along the grid lines.) Then press [ENTER].

For example, suppose the current graph is z1(x,y)=x^2+.5y^2–5. If you specify x=2 and y=3, a contour is drawn for z=3.5.

Note: Any existing contours remain on the graph. To remove the default contours, display the Window editor ([♦] [WINDOW]) and set **ncontour=0**.

Drawing Contours for Specified Z Values

From the Graph screen, display the Draw menu and then select **8:DrwCtour.** The Home screen is displayed automatically with **DrwCtour** in the entry line. You can then specify one or more z values individually or generate a sequence of z values.

Some examples are:

DrwCtour 5	Draws a contour for z=5.
DrwCtour {1,2,3}	Draws contours for z=1, 2, and 3.
DrwCtour seq(n,n,⁻10,10,2)	Draws contours for a sequence of z values from ⁻10 through 10 in steps of 2 (⁻10, ⁻8, ⁻6, etc.).

Note: To remove the default contours, use ◆ [WINDOW] and set **ncontour=0.**

The specified contours are drawn on the current 3D graph. (A contour is not drawn if the specified z value is outside the viewing cube or if the 3D graph is not defined at that z value.)

Notes about Contour Plots

For a contour plot:

- You can use the cursor keys to animate the contour plot.
- You cannot trace (F3) the contours themselves. However, you can trace the wire frame as seen when Style=WIRE AND CONTOUR.
- It may take awhile to evaluate the equation initially.

- Because of possible long evaluation times, you first may want to experiment with your 3D equation by using Style=WIRE FRAME. The evaluation time is much shorter. Then, after you’re sure you have the correct Window variable values, display the Graph Formats dialog box and set Style=CONTOUR LEVELS or WIRE AND CONTOUR.
  [♦] [I]

Example: Contours of a Complex Modulus Surface

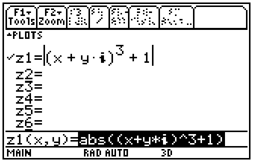
The complex modulus surface given by **z(a,b) = abs(f(a+b*i*))** shows all the complex zeros of any polynomial **y=f(x)**.

Example

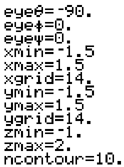
In this example, let f(x)=x^3+1. By substituting the general complex form x+y*i* for x, you can express the complex surface equation as z(x,y)=abs($(x+y\ i)^3$+1).

1. Use [MODE] to set **Graph=3D**.
2. Press [♦] [Y=], and define the equation:

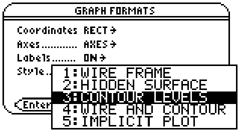
   z1(x,y)=abs((x+y∗*i*)^3+1)



3. Press [♦] [WINDOW], and set the Window variables as shown.



4. Display the Graph Formats dialog box: [◆] [I] Turn on the axes, set **Style = CONTOUR LEVELS**, and return to the Window editor.



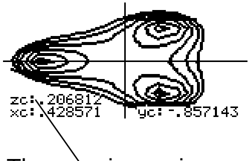
5. Press [◆] [GRAPH] to graph the equation.

   It will take awhile to evaluate the graph; so be patient. When the graph is displayed, the complex modulus surface touches the xy plane at exactly the complex zeros of the polynomial:

   $^-1$, $\frac{1}{2}+\frac{\sqrt{3}}{2}i$, and $\frac{1}{2}-\frac{\sqrt{3}}{2}i$

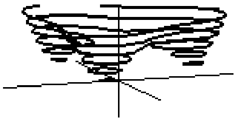
6. Press [F3], and move the trace cursor to the zero in the fourth quadrant.

   The coordinates let you estimate .428–.857i as the zero.



The zero is precise when z=0.

7. Press [ESC]. Then use the cursor keys to animate the graph and view it from different eye angles.



This example shows eyeθ=70, eyeϕ=70, and eyeψ=0.

Notes:

- For more accurate estimates, increase the **xgrid** and **ygrid** Window variables. However, this increases the graph evaluation time.
- When you animate the graph, the screen changes to normal view. Use ⊠ to toggle between normal and expanded views.

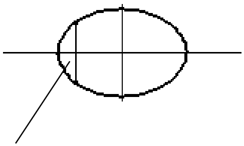
Implicit Plots

An implicit plot is used primarily as a way to graph 2D implicit forms that cannot be graphed in function graphing mode. Technically, an implicit plot is a 3D contour plot with a single contour drawn for **z=0** only.

Explicit and Implicit Forms

In 2D function graphing mode, equations have an explicit form **y=f(x)**, where y is unique for each value of x.

Many equations, however, have an implicit form **f(x,y)=g(x,y)**, where you cannot explicitly solve for y in terms of x or for x in terms of y.



y is not unique for each x, so you cannot graph this in function graphing mode.

By using implicit plots in 3D graphing mode, you can graph these implicit forms without solving for y or x.

Rearrange the implicit form as an equation set to zero.	f(x,y)–g(x,y)=0
In the Y= Editor, enter the non-zero side of the equation. This is valid because an implicit plot automatically sets the equation equal to zero.	z1(x,y)=f(x,y)–g(x,y)
For example, given the ellipse equation shown to the right, enter the implicit form in the Y= Editor.	If x^2+.5y^2=30, then z1(x,y)=x^2+.5y^2–30.

Notes: You can also graph many implicit forms if you either:

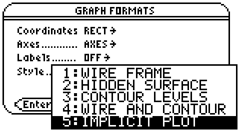
- Express them as parametric equations.
- Break them into separate, explicit functions.

Selecting the Graph Format Style

In 3D graphing mode, define an appropriate equation and graph it as you would any 3D equation, with the following exception. Display the GRAPH FORMATS dialog box from the Y= Editor, Window editor, or Graph screen:
[♦] [I]

Note: From the Graph screen, you can switch to the other graph format styles by

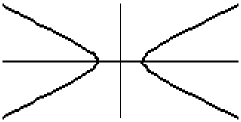
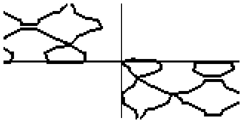
and then set **Style = IMPLICIT PLOT**.



pressing:
[I]

However, to return to IMPLICIT PLOT press:
[♦] [I]

- The viewing angle is set initially so that you are viewing the plot by looking down the z axis. You can change the viewing angle as necessary.
- The plot is shown in expanded view. To switch between expanded and normal view, press [×].
- The Labels format is set to OFF automatically.

Style	**$x^2-y^2=4$** **$z1(x,y)=x^2-y^2-4$**	**sin(x)+cos(y)= e(x∗y)** **z1(x,y)=sin(x)+cos(y)–e(x∗y)**
IMPLICIT PLOT		

Note: These examples use the same x, y, and z Window variable values as a **ZoomStd** viewing cube. If you use **ZoomStd**, press Z to look down the z axis.

Notes About Implicit Plots

For an implicit plot:

- The **ncontour** Window variable has no affect. Only the **z=0** contour is drawn, regardless of the value of ncontour. The displayed plot shows where the implicit form intersects the xy plane.
- You can use the cursor keys to animate the plot.
- You cannot trace (F3) the implicit plot itself. However, you can trace the unseen wire frame graph of the 3D equation.
- It may take awhile to evaluate the equation initially.
- Because of possible long evaluation times, you first may want to experiment with your 3D equation by using Style=WIRE FRAME. The evaluation time is much shorter. Then, after you're sure you have the correct Window variable values, set Style=IMPLICIT PLOT.
  ♦ I

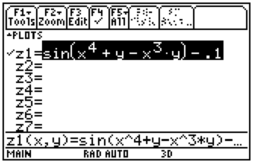
Example: Implicit Plot of a More Complicated Equation

You can use the IMPLICIT PLOT graph format style to plot and animate a complicated equation that cannot be graphed otherwise. Although it may take a long time to evaluate such a graph, the visual results can justify the time required.

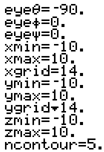
Graph the equation sin(x^4+y–x^3 y) = .1.

1. Use [MODE] to set **Graph=3D**.
2. Press [◆] [Y=], and define the equation:

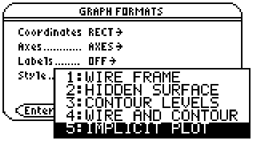
   z1(x,y)=sin(x^4+y– x^3y)–.1



3. Press [◆] [WINDOW], and set the Window variables as shown.

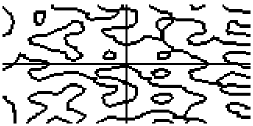


4. Press:
   [◆] [I] Turn on the axes, set **Style = IMPLICIT PLOT**, and return to the Window editor.



5. Press [◆] [GRAPH] to graph the equation.

   It will take awhile to evaluate the graph; so be patient.



The graph shows where sin(x^4+y–x^3y) = .1

6. Use the cursor keys to animate the graph and view it from different eye angles.

   Note: For more detail, increase the **xgrid** and **ygrid** Window variables. However, this increases the graph evaluation time.

In expanded view, this example shows eyeθ=⁻127.85, eyeϕ=52.86, and eyeψ=⁻18.26.

Note: When you animate the graph, the screen changes to normal view. Press [×] to switch between normal and expanded views.

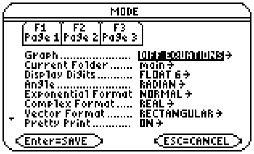
Differential Equation Graphing

Overview of Steps in Graphing Differential Equations

To graph differential equations, use the same general steps used for **y(x)** functions as described in *Basic Function Graphing*. Any differences are described on the following pages.

Graphing Differential Equations

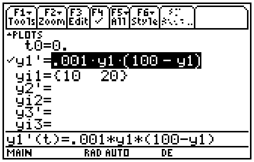
1. Set **Graph** mode ([MODE]) to **DIFF EQUATIONS**. Also set **Angle** mode, if necessary.



2. Define equations and, optionally, initial conditions on Y= Editor ([♦] [Y=]).

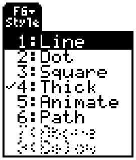
3. Select ([F4]) which defined functions to graph.

   Note: To turn off any stat data plots, press [F5] **5** or use [F4] to deselect them.



4. Set the display style for a function.

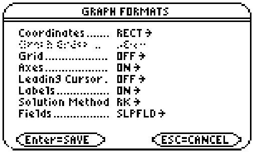
   [2nd] [F6]



5. Set the graph format. **Solution Method** and **Fields** are unique to differential equations.

   [F1] **9**
   — or —
   [♦] [I]

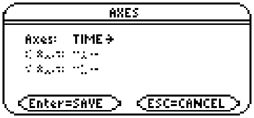
   Note: The **Fields** format is critical, depending on the order of the equation.



6. Set the axes as applicable, depending on the Fields format.

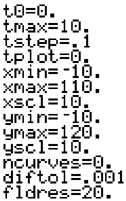
   [2nd] [F7]

   Note: Valid **Axes** settings depend on the Fields format.

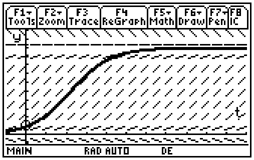


7. Define the viewing window ([♦] [WINDOW]).

   Note: Depending on the **Solution Method** and **Fields** formats, different Window variables are displayed. [F2] **Zoom** also changes the viewing window.



8. Graph the selected functions ([♦] [GRAPH]).



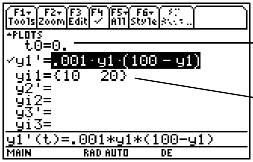
Differences in Diff Equations and Function Graphing

This module assumes that you already know how to graph **y(x)** functions as described in *Basic Function Graphing*. This section describes the differences.

Setting the Graph Mode

Use [MODE] to set **Graph = DIFF EQUATIONS** before you define differential equations or set Window variables. The Y= Editor and the Window Editor let you enter information for the current Graph mode setting only.

Defining Differential Equations on the Y= Editor



Use t0 to specify when initial conditions occur. You can also set t0 in the Window Editor.

Use yi to specify one or more initial conditions for the corresponding differential equation.

You can define differential equations y1'(t) through y99'(t).

Note: You can use the **Define** command from the Home screen to define functions and equations.

When entering equations in the Y= Editor, do not use **y(t)** formats to refer to results. For example:

Do not use implied multiplication between a variable and parenthetical expression. If you do, it is treated as a function call.

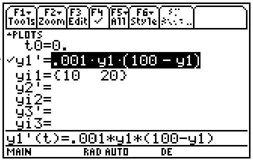
Enter: y1' = .001y1∗(100−y1)
Not: y1' = .001y1(t)∗(100−y1(t))

Only 1st-order equations can be entered in the Y= Editor. To graph 2nd- or higher-order equations, you must enter them as a system of 1st-order equations.

Detailed information is available on setting initial conditions.

Selecting Differential Equations

You can use [F4] to select a differential equation, but not its initial condition.



Important: Selecting y1' will graph the y1 solution curve, not the derivative y1', depending on the axis setting.

Selecting the Display Style

With the Style menu, only the **Line**, **Dot**, **Square**, **Thick**, **Animate**, and **Path** styles are available. **Dot** and **Square** mark only those discrete values (in **tstep** increments) at which a differential equation is plotted.

[2nd] [F6]

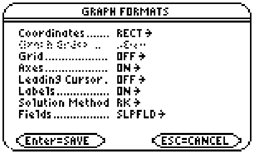
Setting Graph Formats

From the Y= Editor, Window Editor, or Graph screen, press:

[F1] **9**

— or —

[♦] [|]



The formats affected by differential equations are:

Graph format	Description
Graph Order	Not available.
Solution Method	Specifies the method used to solve the differential equations. • RK — Runge-Kutta method. For information about the algorithm used for this method, refer to the *Technical Reference* module. • EULER — Euler method. • The method lets you choose either greater accuracy or speed. Typically, RK is more accurate than EULER but takes longer to find the solution.

Graph format	Description
Fields	Specifies whether to draw a field for the differential equation. • SLPFLD — Draws a slope field for only one 1st-order equation, with t on the x axis and the solution on the y axis. • DIRFLD — Draws a direction field for only one 2nd-order equation (or system of two 1st-order equations), with axes determined by the custom axes settings. • FLDOFF — Does not display a field. This is valid for equations of any order, but you must use it for 3rd- or higher-order. You must enter the same number of initial conditions for all equations in the Y= Editor.

Important: The Fields graph format is critical in successfully graphing differential equations.

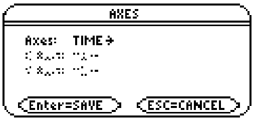
Note: If you press [ENTER] while a slope or direction field is being drawn, the graph pauses after the field is drawn but before the solutions are plotted. Press [ENTER] gain to continue. To cancel graphing, press [ON].

Setting Axes

In the Y= Editor, **Axes** may or may not be available, depending on the current graph format.

If it is available, you can select the axes that are used to graph the differential equations.

[2nd] [F7]



Axes	Description
TIME	Plots t on the x axis and y (the solutions to the selected differential equations) on the y axis.
CUSTOM	Lets you select the x and y axes.

Window Variables

Differential equation graphs use the following Window variables. Depending on the **Solution Method** and **Fields** graph formats, not all of these variables are listed in the Window Editor ([♦] [WINDOW]) at the same time.

Variable	Description
t0	Time at which the initial conditions entered in the Y= Editor occur. You can set **t0** in the Window Editor and Y= Editor. (If you set **t0** in the Y= Editor, **tplot** is set to the same value automatically.)
tmax, **tstep**	Used to determine the t values where the equations are plotted: y'(t0) y'(t0+tstep) y'(t0+2∗tstep) ... not to exceed ... y'(tmax) If **Fields = SLPFLD**, **tmax** is ignored. Equations are plotted from **t0** to both edges of the screen in **tstep** increments.

Variable	Description
tplot	First t value plotted. If this is not a **tstep** increment, plotting begins at the next **tstep** increment. In some situations, the first points evaluated and plotted starting at **t0** may not be interesting visually. By setting **tplot** greater than **t0**, you can start the plot at the interesting area, which speeds up the graphing time and avoids unnecessary clutter on the Graph screen.

Note: If **tmax < t0**, **tstep** must be negative. If **Fields=SLPFLD**, **tplot** is ignored and is assumed to be the same as **t0**.

Variable	Description
xmin, **xmax,** **ymin**, **ymax**	Boundaries of the viewing window.
xscl, **yscl**	Distance between tick marks on the x and y axes.
ncurves	Number of solution curves (0 through 10) that will be drawn automatically if you do not specify an initial condition. By default, **ncurves = 0**.

Variable	Description
	When **ncurves** is used, **t0** is set temporarily at the middle of the screen and initial conditions are distributed evenly along the y axis, where: $$increment = \frac{ymax - ymin}{ncurves + 1}$$ The y values for the initial conditions are: ymin + increment ymin + 2∗(increment) ⋮ ymin + ncurves∗(increment)
diftol	(**Solution Method = RK** only) Tolerance used by the **RK** method to help select a step size for solving the equation; must be ≥1E⁻14.
fldres	(**Fields = SLPFLD** or **DIRFLD** only) Number of columns (1 through 80) used to draw a slope or direction field across the full width of the screen.
Estep	(**Solution Method = EULER** only) Euler iterations between **tstep** values; must be an integer >0. For more accuracy, you can increase **Estep** without plotting additional points.
dtime	(**Fields = DIRFLD** only) Point in time at which a direction field is drawn.

Standard values (set when you select **6:ZoomStd** from the F2 **Zoom** toolbar menu) are:

t0 = 0.	xmin = ⁻1.	ymin = ⁻10.	ncurves = 0.
tmax = 10.	xmax = 10.	ymax = 10.	diftol = .001
tstep = .1	xscl = 1.	yscl = 1.	Estep = 1.
tplot = 0.			fldres = 14.
			dtime = 0.

You may need to change the standard values for the t variables to ensure that sufficient points are plotted.

The fldpic System Variable

When a slope or direction field is drawn, a picture of the field is stored automatically to a system variable named **fldpic**. If you perform an operation that regraphs the plotted equations but does not affect the field, the TI-89 Titanium reuses the picture in **fldpic** instead of having to redraw the field. This can speed up the regraphing time significantly.

fldpic is deleted automatically when you exit the differential equation graphing mode or when you display a graph with **Fields = FLDOFF**.

Exploring a Graph

As in function graphing, you can explore a graph by using the following tools. Any displayed coordinates are shown in rectangular or polar form as set in the graph format.

Tool	For Differential Equation Graphs:
Free-Moving Cursor	Works just as it does for function graphs.

Tool	For Differential Equation Graphs:
[F2] **Zoom**	Works just as it does for function graphs. • Only **x** (**xmin**, **xmax**, **xscl**) and **y** (**ymin**, **ymax**, **yscl**) Window variables are affected. • The **t** Window variables (**t0**, **tmax**, **tstep**, **tplot**) are not affected unless you select **6:ZoomStd** (which sets all Window variables to their standard values).
[F3] **Trace**	Lets you move the cursor along the curve one **tstep** at a time. To move approximately ten plotted points at a time, press [2nd] ▶ or [2nd] ◀. If you enter initial conditions in the Y= Editor or let the **ncurves** Window variable plot curves automatically, you can trace the curves. If you use: [2nd] [F8] IC from the Graph screen to select initial conditions interactively, you cannot trace the curves. QuickCenter applies to all directions. If you move the cursor off the screen (top or bottom, left or right), press [ENTER] to center the viewing window on the cursor location. Use ▲ or ▼ to view results on all plotted curves.
[F5] **Math**	Only **1:Value** is available. • With TIME axes, the **y(t)** solution value (represented by yc) is displayed for a specified t value. • With CUSTOM axes, the values that correspond to x and y depend on the axes you choose.

Note: During a trace, you can move the cursor to a particular point by typing a value for t and pressing [ENTER]. You can use QuickCenter at any time during a trace, even if the cursor is still on the screen.

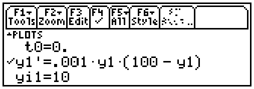
Setting the Initial Conditions

You can enter initial conditions in the Y= Editor, let the TI-89 Titanium calculate initial conditions automatically, or select them interactively from the Graph screen.

Entering Initial Conditions in the Y= Editor

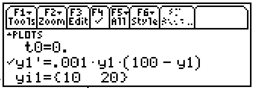
You can specify one or more initial conditions in the Y= Editor. To specify more than one, enter them as a list enclosed in braces { } and separated by commas.

To enter initial conditions for the **y1'** equation, use the **yi1** line, etc.



To specify when the initial conditions occur, use t0. This is also the first t evaluated for the graph.

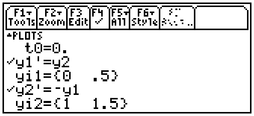
To graph a family of solutions, enter a list of initial conditions.



Enter {10,20} even though {10 20} is displayed.

For a 2nd- or higher-order differential equation, you must define a system of 1st-order equations in the Y= Editor.

If you enter initial conditions, you must enter the same number of initial conditions for each equation in the system. Otherwise, a Dimension error occurs.



If you do not enter initial conditions, the **ncurves** Window variable ([♦] [WINDOW]) specifies the number of solution curves graphed automatically. By default, **ncurves = 0**. You can enter a value from 0 through 10. However, the **Fields** graph format and the **Axes** setting determine whether **ncurves** is used.

If Fields =	Then:
SLPFLD	Uses **ncurves**, if not set to 0, to graph curves.
DIRFLD	Ignores **ncurves**. Does not graph any curves.
FLDOFF	Uses **ncurves** if Axes = TIME (or if Axes = Custom and the x axis is t). Otherwise, a Diff Eq setup error occurs.

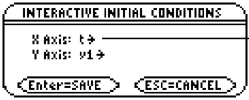
When **ncurves** is used, **t0** is set temporarily at the middle of the Graph screen. However, the value of **t0** as set in the Y= Editor or Window Editor is not changed.

Notes:

- Without entering initial conditions, use SLPFLD (with **ncurves=0**) or DIRFLD to display a slope or direction field only.
- SLPFLD is for a single 1st-order equation only. DIRFLD is for a 2nd-order equation (or system of two 1st-order equations) only.

When a differential equation is graphed (regardless of whether a solution curve is displayed), you can select a point on the Graph screen and use it as an initial condition.

If Fields =	Do this:
SLPFLD – or – DIRFLD	Press: [2nd] [F8] Specify an initial condition. Either: • Move the cursor to the applicable point and press [ENTER]. – or – • For each of the two coordinates, type a value and press [ENTER]. - For SLPFLD (1st-order only), enter values for **t0** and **y(t0)**. - For DIRFLD (2nd-order or system of two 1st-order equations only), enter values for both **y(t0)** initial conditions, where **t0** is the value set in the Y= Editor or Window Editor. A circle marks the initial condition and the solution curve is drawn.

If Fields =	Do this:
FLDOFF	• Press: [2nd] [F8] You are prompted to select the axes for which you want to enter initial conditions.  t is a valid selection. It will let you specify a value for t0. Your selections will be used as the axes for the graph. • You can accept the defaults or change them. Then press [ENTER]. • Specify an initial condition as described for SLPFLD or DIRFLD.

Note: With SLPFLD or DIRFLD, you can select initial conditions interactively regardless of whether you enter initial conditions in the Y= Editor. With FLDOFF, you can select initial conditions interactively. However, if three or more equations are entered, you must enter a single value (not a list) as the initial condition for each equation in the Y= Editor. Otherwise, a Dimension error occurs when graphing.

Note about Tracing a Solution Curve

When you enter initial conditions in the Y= Editor or let **ncurves** graph solution curves automatically, you can use [F3] to trace the curves. However, you cannot trace a curve drawn by selecting an initial condition interactively. These curves are drawn, not plotted.

Defining a System for Higher-Order Equations

In the Y= Editor, you must enter all differential equations as 1st-order equations. If you have an nth-order equation, you must transform it into a system of n 1st-order equations.

Transforming an Equation into a 1st-Order System

A system of equations can be defined in various ways, but the following is a general method.

1. Rewrite the original differential equation as necessary.

   $y'' + y' + y = e^x$

   a) Solve for the highest-ordered derivative.

   $y'' = e^x - y' - y$

   b) Express it in terms of y and t.

   $y'' = e^t - y' - y$

   c) On the right side of the equation only, substitute to eliminate any references to derivative values.

   Note: To produce a 1st-order equation, the right side must contain non-derivative variables only.

In place of:	Substitute:
y	y1
y'	y2
y''	y3
y'''	y4
y(4)	y5
⋮	⋮

y'' = e^t − y2 − y1

Do not substitute on the left side at this time.

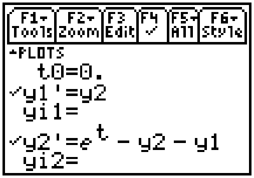
d) On the left side of the equation, substitute for the derivative value as shown below.

In place of:	Substitute:
y’	y1’
y’’	y2’
y'’’	y3’
y(4)	y4’
⋮	⋮

y2' = e^t − y2 − y1

2. On the applicable lines in the Y= Editor, define the system of equations as:

   y1' = y2
   y2' = y3
   y3' = y4
   – up to –
   yn ' = your nth-order equation



Note: Based on the above substitutions, the y' lines in the Y= Editor represent:

y1' = y'
y2' = y''
etc.

Therefore, this example’s 2nd-order equation is entered on the y2' line.

In a system such as this, the solution to the y1' equation is the solution to the nth-order equation. You may want to deselect any other equations in the system.

Example of a 2nd-Order Equation

The 2nd-order differential equation y''+y = 0 represents a simple harmonic oscillator. Transform this into a system of equations for the Y= Editor. Then, graph the solution for initial conditions y(0) = 0 and y'(0) = 1.

1. Press [MODE] and set **Graph=DIFF EQUATIONS**.

2. Define a system of equations for the 2nd-order equation.

   Rewrite the equation and make the necessary substitutions.

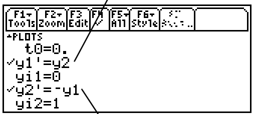
   y'' + y = 0
   y'' = ⁻y
   y'' = ⁻y1
   y2' = ⁻y1

3. In the Y= Editor ([♦] [Y=]), enter the system of equations.

4. Enter the initial conditions: **yi1=0** and **yi2=1**

   Note: t0 is the time at which the initial conditions occur. It is also the first t evaluated for the graph. By default, **t0=0**.

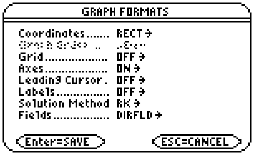
yi1 is the initial condition for y(0).



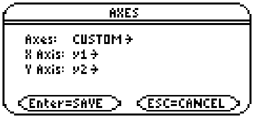
yi2 is the initial condition for y'(0).

5. Press:
   [F1] **9**
   — or —
   [♦] [|] and set **Axes = ON**, **Labels = OFF**, **Solution Method = RK**, and **Fields = DIRFLD**.

   Important: For 2nd-order equations, you must set **Fields=DIRFLD** or **FLDOFF**.



6. In the Y= Editor, press:
 [2nd] [F7]and make sure **Axes = CUSTOM** with **y1** and **y2** as the axes.



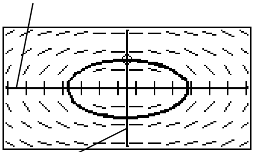
Important: Fields=DIRFLD cannot plot a time axis. An **Invalid Axes** error occurs if **Axes=TIME** or if t is set as a **CUSTOM** axis.

7. In the Window Editor ([♦] [WINDOW]), set the Window variables.

t0=0	xmin=⁻2	ncurves=0
tmax=10	xmax=2	diftol=.001
tstep=.1	xscl=1	fldres=14
tplot=0	ymin=⁻2	dtime=0
	ymax=2	
	yscl=1	

8. Display the Graph screen ([♦] [GRAPH]).

x axis = y1 = y



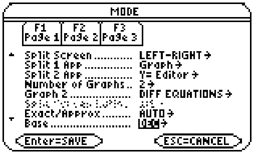
y axis = y2 = y'

If you select **ZoomSqr** ([F2] **5**), you can see that the phase-plane orbit is actually a circle. However, **ZoomSqr** will change your Window variables.

To examine this harmonic oscillator in more detail, use a split screen to graph the manner in which y and y' change with respect to time (t).

9. Press [MODE] and change the mode settings on **Page 2** as shown. Then close the **MODE** dialog box, which redraws the graph.

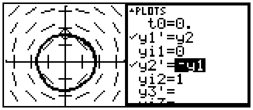
   Note: To display different graphs in both parts of a split screen, you must use the **2-graph** mode.



10. Press [2nd] [⇔] to switch to the right side of the split screen.

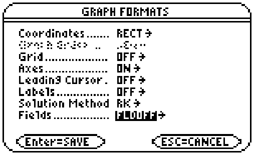
11. Use [F4] to select **y1'** and **y2'**.

    The right side uses the same equations as the left side. However, no equations are selected initially in the right side.

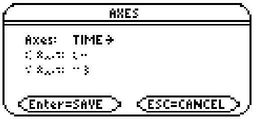


12. Press:
    [F1] **9**
    — or —
    [calculator icon] [♦] [|] Set **Fields = FLDOFF.**

    Important: Because **Fields=DIRFLD** cannot plot a time axis, you must change the Fields setting. **FLDOFF** turns off all fields.



13. In the Y= Editor, press:
    [calculator icon] [2nd] [F7]and make sure **Axes = TIME.**



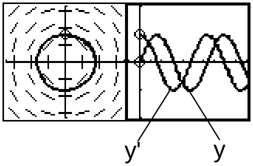
14. In the Window Editor, change **ymin** and **ymax** as shown to the right.

    Note: When you enter **2-graph** mode, Window variables for the right side are set to their defaults.

ymin=⁻2.
ymax=2.

15. Press [♦] [GRAPH] to display the Graph screen for graph #2.

    The left side shows the phase-plane orbit. The right side shows the solution curve and its derivative.



16. To return to a full screen of the original graph, press [2nd] [⊞] to switch to the left side. Then press [MODE] and change the **Split Screen** setting.

Split Screen = FULL

Example of a 3rd-Order Equation

For the 3rd-order differential equation **y'''+2y''+2y'+y = sin(x)**, write a system of equations to enter in the Y= Editor. Then graph the solution as a function of time. Use initial conditions **y(0) = 0**, **y'(0) = 1**, and **y''(0) = 1.**

1. Press [MODE] and set **Graph=DIFF EQUATIONS**.

2. Define a system of equations for the 3rd-order equation.

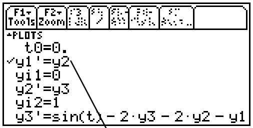
   Rewrite the equation and make the necessary substitutions.

   y''' + 2y'' + 2y' + y = sin(x)
   y''' = sin(x) − 2y'' − 2y' − y
   y''' = sin(t) − 2y'' − 2y' − y
   y''' = sin(t) − 2y3 − 2y2 − y1
   y3' = sin(t) − 2y3 − 2y2 − y1

3. In the Y= Editor ([♦] [Y=]), enter the system of equations.

4. Enter the initial conditions: **yi1=0**, **yi2=1**, and **yi3=1**

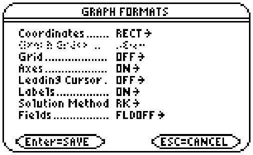
   Note: t0 is the time at which the initial conditions occur. By default, **t0=0.**



Important: The solution to the y1' equation is the solution to the 3rd-order equation.

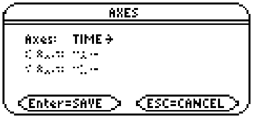
5. Be sure that only **y1'** is selected. Use [F4] to deselect any other equations.

6. Press:
   [F1] **9**
   — or —
   [◆] [|] Set **Axes = ON**, **Labels = ON**, **Solution Method = RK**, and **Fields = FLDOFF.**

   Important: For 3rd- or higher-order equations, you must set **Fields=FLDOFF.** Otherwise, an **Undefined variable** error occurs when graphing.

7. In the Y= Editor, press:
   [2nd] [F7] Set **Axes = TIME.**

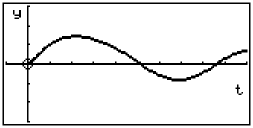
   Note: With **Axes=TIME**, the solution to the selected equation is plotted against time (t).

8. In the Window Editor ([◆] [WINDOW]), set the Window variables.

t0=0	xmin=-1	ncurves=0
tmax=10	xmax=10	diftol=.001
tstep=.1	xscl=1.	
tplot=0	ymin=-3	
	ymax=3	
	yscl=1	

9. Display the Graph screen ([♦] [GRAPH]).

   Note: To find the solution at a particular time, use [F3] to trace the graph.



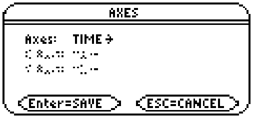
Setting Axes for Time or Custom Plots

Setting the axes can give you great flexibility in graphing differential equations. Custom axes are particularly effective for showing different kinds of relationships.

Displaying the AXES Dialog Box

From the Y= Editor, press:

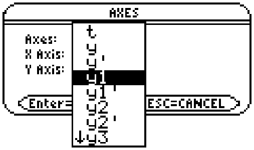
[2nd] [F7]



If **Fields = SLPFLD**, **Axes** is unavailable.

[2nd] [F7]

Item	Description
Axes	TIME — Plots t on the x axis and y (solutions to all selected differential equations) on the y axis. CUSTOM — Lets you select the x and y axes.

Item	Description
X Axis, **Y Axis**	Active only when Axes = CUSTOM, these let you select what you want to plot on the x and y axes. 
	t — time
	y — solutions (y1, y2, etc.) of all selected differential equations
	y' — values of all selected differential equations (y1', y2', etc.)
	y1, y2, etc. — the solution to the corresponding differential equation, regardless of whether that equation is selected
	y1', y2', etc. — the value of the right-hand side of the corresponding differential equation, regardless of whether that equation is selected

Note: t is not valid for either Axis when **Fields=DIRFLD**. If you select t, an Invalid axes error occurs when graphing.

Example of Time and Custom Axes

Using the predator-prey model from biology, determine the numbers of rabbits and foxes that maintain population equilibrium in a certain region. Graph the solution using both time and custom axes.

Use the two coupled 1st-order differential equations:

$y1' = {}^{-}y1 + 0.1y1 * y2$ and $y2' = 3y2 - y1 * y2$

where:

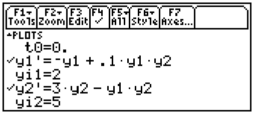
y1 = Population of foxes

yi1 = Initial population of foxes (2)

y2 = Population of rabbits

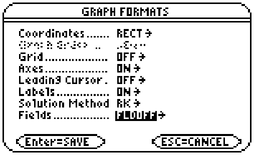
yi2 = Initial population of rabbits (5)

1. Use [MODE] to set **Graph = DIFF EQUATIONS**.
2. In the Y= Editor ([♦] [Y=]), define the differential equations and enter the initial conditions.

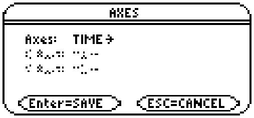


   Note: To speed up graphing times, clear any other equations in the Y= Editor. With **FLDOFF**, all equations are evaluated even if they are not selected.

3. Press:
   [F1] **9**
   — or —
   [♦] [I] Set **Axes = ON**, **Labels = ON**, **Solution Method = RK**, and **Fields = FLDOFF**.



4. In the Y= Editor, press:
   [calculator icon] 2nd [F7] Set **Axes = TIME.**



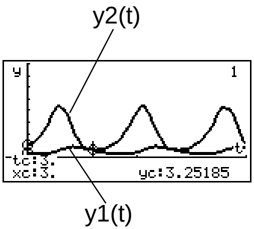
5. In the Window Editor (◆ [WINDOW]), set the Window variables.

t0=0	xmin=⁻1	ncurves=0
tmax=10	xmax=10	diftol=.001
tstep=π/24	xscl=5	
tplot=0	ymin=⁻10	
	ymax=40	
	yscl=5	

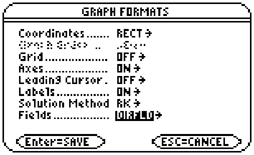
6. Graph the differential equations (◆ [GRAPH]).

7. Press [F3] to trace. Then press **3** [ENTER] to see the number of foxes (**yc** for **y1**) and rabbits (**yc** for **y2**) at **t=3**.

   Note: Use ⊖ and ⊖ to move the trace cursor between the curves for **y1** and **y2**.

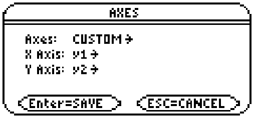


8. Return to the Y= Editor. Press:
   [F1] **9**
   — or —
   [♦] [I]
   Set **Fields = DIRFLD**.

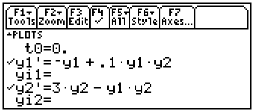
   

   Note: In this example, **DIRFLD** is used for two related differential equations that do not represent a 2nd-order equation.

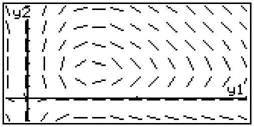
9. Press:
   [2nd] [F7]
   Confirm that the axes are set as shown.

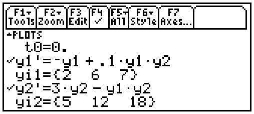
10. In the Y= Editor, clear the initial conditions for **yi1** and **yi2**.

11. Return to the Graph screen, which displays only the direction field.

12. To graph a family of solutions, return to the Y= Editor and enter the initial conditions shown below.

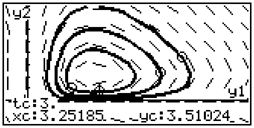
    

    yi1={2,6,7} and **yi2={5,12,18}**

    Note: Use a list to specify more than one initial condition.

13. Return to the Graph screen, which displays a curve for each pair of initial conditions.

14. Press [F3] to trace. Then press **3** [ENTER] to see the number of foxes (**xc**) and rabbits (**yc**) at **t=3**.



    Because **t0=0** and **tmax=10**, you can trace in the range $0 \le t \le 10$.

    Note: Use ↑ and ↓ to move the trace cursor from one initial condition curve to another.

Example Comparison of RK and Euler

Consider a logistic growth model **dP/dt = .001∗P∗(100−P)**, with the initial condition **P(0) = 10**. Use the **BldData** instruction to compare the graphing points calculated by the RK and Euler solution methods. Then plot those points along with a graph of the equation's exact solution.

Example

1. Press [MODE] and set **Graph=DIFF EQUATIONS**.

2. Express the 1st-order equation in terms of **y1'** and **y1**.

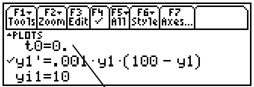
y1'=.001y1∗(100−y1)

Do not use implied multiplication between the variable and parentheses. If you do, it is treated as a function call.

3. Enter the equation in the Y= Editor ([◆] [Y=]).

4. Enter the initial condition:

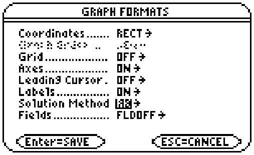
   yi1=10



t0 is the time at which the initial condition occurs. By default, t0=0.

5. Press:
   [F1] **9**
   — or —
   [◆] [|]
   Set **Solution Method = RK** and **Fields = FLDOFF**.

   Note: To speed up graphing times, clear any other equations in the Y= Editor. With **FLDOFF**, all equations are evaluated even if they are not selected.



6. In the Window Editor ([◆] [WINDOW]), set the Window variables.

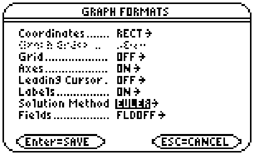
t0=0.	xmin=⁻1.	ncurves=0.
tmax=100.	xmax=100.	diftol=.001
❶ tstep=1.	xscl=1.	
tplot=0.	ymin=⁻10.	
	ymax=10	
	yscl=1.	

❶ **Important:** Change **tstep** from .1 (its default) to 1. Otherwise, **BldData** calculates too many rows for the data variable and a Dimension error occurs.

7. In the Home screen
   [HOME] [CALC HOME]
   use **BldData** to create a data variable containing the **RK** graphing points.

   BldData rklog

8. Return to the Y= Editor, press:
   [F1] **9**
   — or —
   [♦] [I]
   Set **Solution Method = EULER.**

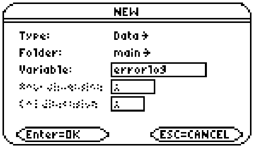
   Note: You do not need to graph the equation before using **BldData.** For more information about **BldData**, refer to the *Technical Reference* module.

9. Return to the Home screen, and use **BldData** to create a data variable containing the **Euler** graphing points.

   BldData eulerlog

10. Use the **Data/Matrix Editor** (APPS) to create a new data variable named **errorlog**.

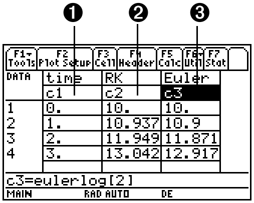
    Note: errorlog lets you combine the data in **rklog** and **eulerlog** so that you can view the two sets of data side by side.



11. In this new data variable, define the **c1**, **c2**, and **c3** column headers to refer to data in **rklog** and **eulerlog**. Also, enter column titles as shown.

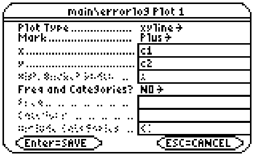
    To define a column header, move the cursor to that column, press F4, type the reference expression (such as **rklog[1]** for **c1**), and press ENTER.

    Note: rklog[1] and **rklog[2]** refer to column 1 and 2 in **rklog**, respectively. Likewise with **eulerlog[2]**.



❶ c1=rklog[1] or c1=eulerlog[1]
❷ c2=rklog[2]
❸ c3= eulerlog[2]

12. In the **Data/Matrix Editor**, press F2. Then press F1 and define **Plot 1** for the **RK** data, as shown to the right.



13. Define **Plot 2** for the **Euler** data. Use the values shown to the right.

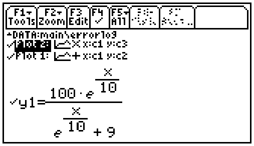
Plot Type=xyline
Mark=Cross
x=c1
y=c3

14. Return to the Y= Editor, press MODE, and set **Graph = FUNCTION**.

15. The exact solution to the differential equation is given below. Enter it as y1.

    y1 = (100∗e^(x/10))/(e^(x/10)+9)

    Note: You can use **deSolve( )** to find this exact, general solution. ,



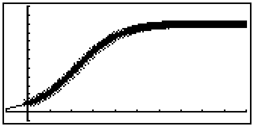
You can use ⊖ to scroll up to see Plot 1 and Plot 2.

16. In the Window Editor, set the Window variables.

xmin=⁻10	ymin=⁻10.	xres=2.
xmax=100	ymax=120.	
xscl=10	yscl=10.	

17. Display the Graph screen (◆ [GRAPH]).

    Note: The fuzzy line on the graph indicates differences between the **RK** and **Euler** values.

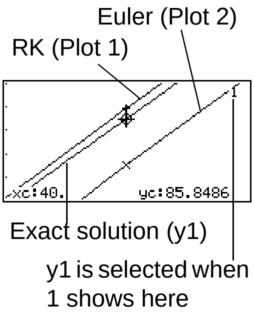


18. In the Window Editor, set the Window variables to zoom in so that you can examine the differences in more detail.

xmin=39.7	ymin=85.5	xres=2
xmax=40.3	ymax=86	
xscl=.1	yscl=.1	

19. Return to the Graph screen.

20. Press [F3] to trace, and then press ⊙ or ⊙ until **y1** is selected. (1 shows in upper right corner.) Then enter 40.



By moving the trace cursor to trace each solution to **xc = 40**, you can find that:

- The exact solution (**y1**) is 85.8486, rounded to six digits.
- The **RK** solution (**Plot 1**) is 85.8952.
- The **Euler** solution (**Plot 2**) is 85.6527.

You can also use the **Data/Matrix Editor** to open the errorlog data variable and scroll to **time = 40**.

Example of the deSolve( ) Function

The **deSolve( )** function lets you solve many 1st- and 2nd-order ordinary differential equations exactly.

Example

For a general solution, use the following syntax. For a particular solution, refer to the Technical Reference module.

deSolve(*1stOr2ndOrderODE, independentVar, dependentVar***)**

Using the logistic 1st-order differential equation, find the general solution for y with respect to t.

deSolve(y' = 1/1000 y∗(100−y),t,y**)**

For ', type [2nd] ['].

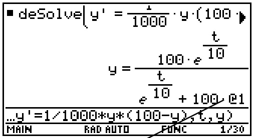
Do not use implied multiplication between the variable and parentheses. If you do, it will be treated as a function call.

Notes:

- For maximum accuracy, use 1/1000 instead of .001. A floating-point number can introduce round-off errors.
- This example does not involve graphing, so you can use any Graph mode.

Before using **deSolve( )**, clear any existing t and y variables. Otherwise, an error occurs.

1. In the Home screen
   [HOME] [CALC HOME]use **deSolve( )** to find the general solution.

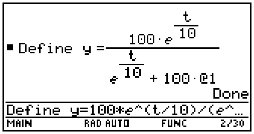


@1 represents a constant. You may get a different constant (@2, etc.).

2. Use the solution to define a function.

   a) Press ⏶ to highlight the solution in the history area. Then press [ENTER] to autopaste it into the entry line.

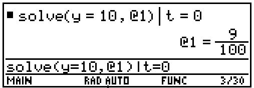
   b) Insert the **Define** instruction at the beginning of the line. Then press [ENTER].



Note: Press [2nd] ⏴ to move to the beginning of the entry line.

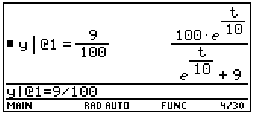
3. For an initial condition **y=10** with **t=0**, use **solve( )** to find the @1 constant.

   Note: If you got a different constant (@2, etc.), solve for that constant.



For @, type
[♦] [STO▶]
[2nd] **R**

4. Evaluate the general solution (**y**) with the constant **@1=9/100** to obtain the particular solution shown.



You can also use **deSolve( )** to solve this problem directly. Enter:

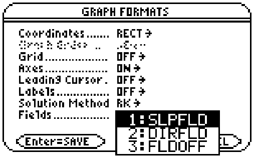
deSolve(y' = 1/1000 y∗(100−y) and y(0)=10,t,y**)**

Troubleshooting with the Fields Graph Format

If you have difficulties graphing a differential equation, this section can help you correct the problem. Many problems may be related to your Fields graph format setting.

Setting the Fields Graph Format

From the Y= Editor, Window Editor, or Graph screen, press:
F1 **9**
— or —
♦ |

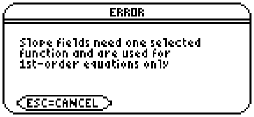


What Order Equation Are You Graphing?

If the equation is:	Valid Fields settings are:
1st-order	SLPFLD or FLDOFF

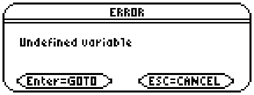
If the equation is:	Valid Fields settings are:
2nd-order (system of two 1st-order equations)	DIRFLD or FLDOFF
3rd- or higher-order (system of three or more 1st-order equations)	FLDOFF

Because **Fields = SLPFLD** is the default setting, a common error message is shown to the right.



When you see this or any other error message:

- For your order of equation, use the previous table to find the valid Fields settings. Change to the applicable setting.
- For a particular Fields setting, check the following for information that applies to that setting.

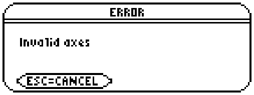
In the Y= Editor	Use [F4] to select one and only one 1st-order equation. You can enter mulNotele equations, but only one at a time can be selected. The selected equation must not refer to any other equation in the Y= Editor. For example: If **y1'=y2**, an Undefined variable error occurs when you graph. 
In the Graph screen	If the slope field is drawn but no solution curve is plotted, specify an initial condition.

In the Y= Editor

Enter a valid system of two 1st-order equations. For information about defining a valid system for a 2nd-order equation, refer to Example of a 2nd-Order Equation.

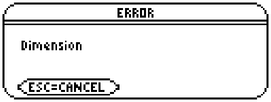
Set **Axes = CUSTOM**:

[2nd] [F7] If **Axes = TIME**, an Invalid axes error occurs when you graph.



If you enter initial conditions in the Y= Editor, the equations referenced by the custom axes must have the same number of initial conditions.

Otherwise, a Dimension error occurs when you graph.



With custom axes

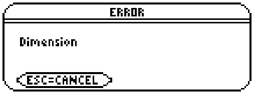
Set axes that are valid for your system of equations.

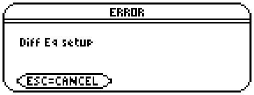
Do not select t for either axis. Otherwise, an Invalid axes error occurs when you graph.

The two axes must refer to different equations in your system of equations. For example, y1 vs. y2 is valid, but y1 vs. y1' gives an Invalid axes error.

In the Graph screen	If the direction field is drawn but no curve is plotted, enter initial conditions in the Y= Editor or select one interactively from the Graph screen. If you did enter initial conditions, select **ZoomFit**: [F2] [alpha] **A** The **ncurves** Window variable is ignored with DIRFLD. Default curves are not drawn automatically.
Notes	With DIRFLD, the equations referenced by the custom axes determine which equations are graphed, regardless of which equations are selected in the Y= Editor. If your system of equations refers to t, the direction field (not the plotted curves) is drawn with respect to one particular time, which is set by the dtime Window variable.

Fields=FLDOFF

In the Y= Editor	If you enter a 2nd- or higher-order equation, enter it as a valid system of equations. All equations (selected or not) must have the same number of initial conditions. Otherwise, a Dimension error occurs when you graph.  To set **Axes = TIME** or **CUSTOM**, press: [2nd] [F7]

With custom axes	If X Axis is not t, you must enter at least one initial condition for each equation in the Y= Editor (whether the equation is selected or not). Otherwise, a Diff Eq setup error occurs when you graph. 
In the Graph screen	If no curve is graphed, set an initial condition. If you did enter initial conditions in the Y= Editor, select **ZoomFit**: [F2] [alpha] **A** A 1st-order equation may look different with FLDOFF than with SLPFLD. This is because FLDOFF uses the **tplot** and **tmax** Window variables (page 9), which are ignored with SLPFLD.
Notes	For 1st-order equations, use FLDOFF and Axes = Custom to plot axes that are not possible with SLPFLD. For example, you can plot t vs. y1' (where SLPFLD plots t vs. y1). If you enter mulNotele 1st-order equations, you can plot one equation or its solution vs. another by specifying them as the axes.

If You Use the Table Screen to View Differential Equations

You can use the Table screen to view the points for a differential equation graph. However, the table may show different equations than those graphed. The table shows only the selected equations, regardless of whether those equations will be plotted with your current **Fields** and **Axes** settings

Tables

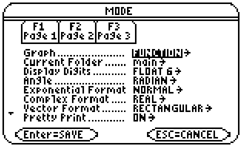
Overview of Steps in Generating a Table

To generate a table of values for one or more functions, use the general steps shown below. For specific information about setting table parameters and displaying the table, refer to the following pages.

Generating a Table

1. Set **Graph** mode and, if necessary, **Angle** mode ([MODE]).

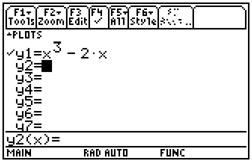
   Note: Tables are not available in **3D Graph** mode.



2. Define functions on Y= Editor ([♦] [Y=]).

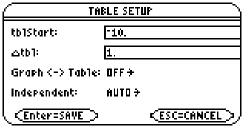
3. Select ([F4]) which defined functions to display in the table.

   Note: For information on defining and selecting functions with the Y= Editor, refer to *Basic Function Graphing*.

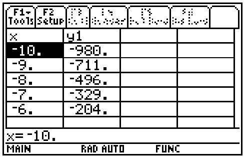


4. Set up the initial table parameters ([♦] [TBLSET]).

   Note: You can specify an automatic table that is based on initial values or that matches a graph, or a manual (ask) table.



5. Display the table (◆ [TABLE]).



Exploring the Table

From the Table screen, you can:

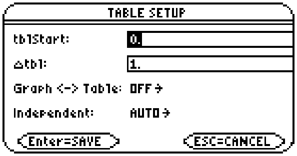
- Scroll through the table to see values on other pages.
- Highlight a cell to see its full value.
- Change the table's setup parameters. By changing the starting or incremental value used for the independent variable, you can zoom in or out on the table to see different levels of detail.
- Change the cell width.
- Edit selected functions.
- Build or edit a manual table to show only specified values of the independent variable.

Setting Up the Table Parameters

To set up the initial parameters for a table, use the TABLE SETUP dialog box. After the table is displayed, you can also use this dialog box to change the parameters.

Displaying the TABLE SETUP Dialog Box

To display the TABLE SETUP dialog box, press [♦] [TBLSET]. From the Table screen, you can also press [F2].



Setup Parameter	Description
tblStart	If **Independent = AUTO** and **Graph < - > Table = OFF**, this specifies the starting value for the independent variable.
Δtbl	If **Independent = AUTO** and **Graph < - > Table = OFF**, this specifies the incremental value for the independent variable. Δtbl can be positive or negative, but not zero.
Graph < - > Table	If **Independent = AUTO**: OFF — The table is based on the values you enter for **tblStart** and **Δtbl**. ON — The table is based on the same independent variable values that are used to graph the functions on the Graph screen. These values depend on the Window variables set in the Window Editor and the split screen size.

Setup Parameter	Description
Independent	AUTO — The TI-89 Titanium automatically generates a series of values for the independent variable based on **tblStart**, **Δtbl**, and **Graph < - > Table**. ASK — Lets you build a table manually by entering specific values for the independent variable.

Note: The table initially starts at **tblStart**, but you can use ⊖ to scroll to prior values.

Which Setup Parameters to Use

To generate:	tblStart	Δtbl	Graph < - > Table	Independent
An automatic table				
• Based on initial values	value	value	**OFF**	**AUTO**
• That matches Graph screen	–	–	**ON**	**AUTO**
A manual table	–	–	–	**ASK**

Note: "–" means that any value entered for this parameter is ignored for the indicated type of table.

In SEQUENCE graphing mode, use integers for **tblStart** and **Δtbl.**

Changing the Setup Parameters

From the **TABLE SETUP** dialog box:

1. Use ⊖ and ⊖ to highlight the value or setting to change.
2. Specify the new value or setting.

To change:	Do this:
tblStart or **∆tbl**	Type the new value. The existing value is erased when you start to type. — or — Press ◀ or ▶ to remove the highlighting. Then edit the existing value.
Graph < - > Table or **Independent**	Press ◀ or ▶ to display a menu of valid settings. Then either: • Move the cursor to highlight the setting and press [ENTER]. — or — • Press the number for that setting.

Note: To cancel a menu or exit the dialog box without saving any changes, press [ESC] instead of [ENTER].

3. After changing all applicable values or settings, press [ENTER] to save your changes and close the dialog box.

From the Home Screen or a Program

You can set up a table's parameters from the Home screen or a program. You can:

- Store values directly to the system variables **tblStart** and **∆tbl**. Refer to "Storing and Recalling Variable Values" in *Operating the Calculator*.

- Set **Graph < - > Table** and Independent by using the **setTable** function. Refer to the *Technical Reference* module.

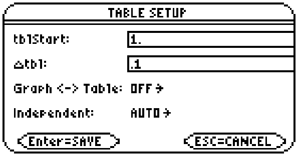
Displaying an Automatic Table

If **Independent = AUTO** on the **TABLE SETUP** dialog box, a table is generated automatically when you display the Table screen. If **Graph < - > Table = ON**, the table matches the trace values from the Graph screen. If **Graph < - > Table = OFF**, the table is based on the values you entered for **tblStart** and **∆tbl**.

Before You Begin

Define and select the applicable functions on the Y= Editor ([♦] [Y=]). This example uses **y1(x) = x^3−x/3.**

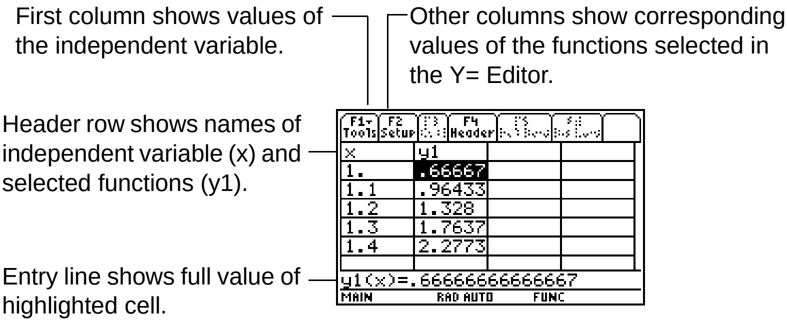
Then enter the initial table parameters ([♦] [TBLSET]).



Displaying the Table Screen

To display the Table screen, press [♦] [TABLE] or [APPS] **5.**

The cursor initially highlights the cell that contains the starting value of the independent variable. You can move the cursor to any cell that contains a value.



Note: You can scroll back from the starting value by pressing ▲ or [2nd] ▲.

To move the cursor:	Press:
One cell at a time	▼, ▲, ▶, or ◀
One page at a time	[2nd] and then ▼, ▲, ▶, or ◀

The header row and the first column are fixed so that they cannot scroll off the screen.

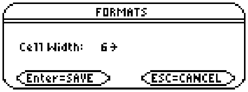
- When you scroll down or up, the variable and function names are always visible across the top of the screen.
- When you scroll right or left, the values of the independent variable are always visible along the left side of the screen.

Changing the Cell Width

Cell width determines the maximum number of digits and symbols (decimal point, minus sign, and “E” for scientific notation) that can be displayed in a cell. All cells in the table have the same width.

Note: By default, the cell width is **6**.

To change the cell width from the **Table** screen:



1. Press [F1] **9**
   — or —
   [♦] [|]

2. Press ▶ or ◀ to display a menu of valid widths (**3–12**).

3. Move the cursor to highlight a number and press [ENTER]. (For single-digit numbers, you can type the number and press [ENTER].)

4. Press [ENTER] to close the dialog box and update the table.

How Numbers Are Displayed in a Cell

Whenever possible, a number is shown according to the currently selected display modes (Display Digits, Exponential Format, etc.). The number may be rounded as necessary. However:

- If a number’s magnitude is too large for the current cell width, the number is rounded and shown in scientific notation.
- If the cell width is too narrow even for scientific notation, “...” is shown.

Notes:

- If a function is undefined at a particular value, undef is displayed in the cell.
- Use [MODE] to set the display modes.

By default, **Display Digits = FLOAT 6**. With this mode setting, a number is shown with up to six digits, even if the cell is wide enough to show more. Other settings similarly affect a displayed number.

	If cell width is:			
Full Precision	**3**	**6**	**9**	**12**
1.2345678901	1.2	1.2346	1.23457	1.23457*
$^{-}$123456.78	...	$^{-}$1.2E5	$^{-}$123457.	$^{-}$123457.*
.000005	...	5.E^{-}6	.000005	.000005
1.2345678E19	...	1.2E19	1.2346E19	1.23457E19*
$^{-}$1.23456789012E^{-}200	...	...	$^{-}$1.2E^{-}200	$^{-}$1.2346E^{-}200*

***Note:** Depending on display mode settings, some values are not shown in full precision even when the cell is wide

Note: To see a number in full precision, highlight the cell and look at the entry line.

If Results are Complex Numbers

A cell shows as much as possible of a complex number (according to the current display modes) and then shows “...” at the end of the displayed portion.

When you highlight a cell containing a complex number, the entry line shows the real and imaginary parts with a maximum of four digits each (FLOAT 4).

Editing a Selected Function

From a table, you can change a selected function without having to use the Y= Editor.

1. Move the cursor to any cell in the column for that function. The table’s header row shows the function names (**y1,** etc.).
2. Press [F4] to move the cursor to the entry line, where the function is displayed and highlighted.

   Note: You can use this feature to view a function without leaving the table.
3. Make any changes, as necessary.
   - Type the new function. The old function is erased when you begin typing.

     — or —
   - Press [CLEAR] to clear the old function. Then type the new one.

     — or —
   - Press ◀ or ▶ to remove the highlighting. Then edit the function.

   Note: To cancel any changes and return the cursor to the table, press [ESC] instead of [ENTER].

4. Press [ENTER] to save the edited function and update the table. The edited function is also saved in the Y= Editor.

If You Want to Change the Setup Parameters

After generating an automatic table, you can change its setup parameters as necessary.

Press [F2] or [♦] [TBLSET] to display the TABLE SETUP dialog box. Then make your changes.

Building a Manual (Ask) Table

If **Independent = ASK** on the TABLE SETUP dialog box, the TI-89 Titanium lets you build a table manually by entering specific values for the independent variable.

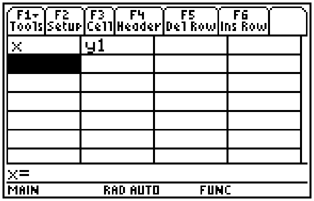
Displaying the Table Screen

To display the Table screen, press [♦] [TABLE].

If you set **Independent = ASK** (with [♦] [TBLSET]) before displaying a table for the first time, a blank table is displayed. The cursor highlights the first cell in the independent variable column.

Header row shows names of independent variable (x) and selected functions (y1).

Enter a value here.



If you first display an automatic table and then change it to **Independent = ASK**, the table continues to show the same values. However, you can no longer see additional values by scrolling up or down off the screen.

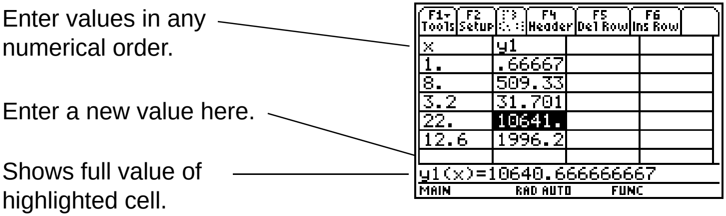
Entering or Editing an Independent Variable Value

You can enter a value in column 1 (independent variable) only.

1. Move the cursor to highlight the cell you want to enter or edit.
   - If you start with a blank table, you can enter a value in consecutive cells only (row 1, row 2, etc.). You cannot skip cells (row 1, row 3).
   - If a cell in column 1 contains a value, you can edit that value.
2. Press [F3] to move the cursor to the entry line.
3. Type a new value or expression, or edit the existing value.
4. Press [ENTER] to move the value to the table and update the corresponding function values.

   Note: To enter a new value in a cell, you do not need to press [F3]. Simply begin typing.

The cursor returns to the entered cell. You can use ⊙ to move to the next row.



Note: In this example, you can move the cursor to column 2, but you can enter values in column 1 only.

Entering a List in the Independent Variable Column

1. Move the cursor to highlight any cell in the independent variable column.
2. Press [F4] to move the cursor to the entry line.
3. Type a series of values, enclosed in braces { } and separated by commas. For example:

   ```
   x={1,1.5,1.75,2}
   ```

   You can also enter a list variable or an expression that evaluates to a list.

   Note: If the independent variable column contains existing values, they are shown as a list (which you can edit).
4. Press [ENTER] to move the values into the independent variable column. The table is updated to show the corresponding function values.

Adding, Deleting, or Clearing

To:	Do this:
Insert a new row above a specified row	Highlight a cell in the specified row and press: [calculator icon] [2nd] [F6] The new row is undefined (undef) until you enter a value for the independent variable.
Delete a row	Highlight a cell in the row and press [F5]. If you highlight a cell in the independent variable column, you can also press [←].
Clear the entire table (but not the selected Y= functions)	Press [F1] **8**. When prompted for confirmation, press [ENTER].

Cell Width and Display Formats

Several factors affect how numbers are displayed in a table.

From the Home Screen or a Program

System variable **tblInput** contains a list of all independent variable values entered in the table, even those not currently displayed. **tblInput** is also used for an automatic table, but it contains only the independent variable values that are currently displayed.

Before displaying a table, you can store a list of values directly to the **tblInput** system variable.

Additional Graphing Topics

Collecting Data Points from a Graph

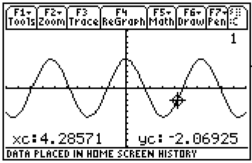
From the Graph screen, you can store sets of coordinate values and/or math results for later analysis. You can store the information as a single-row matrix (vector) on the Home screen or as data points in a system data variable that can be opened in the Data/Matrix Editor.

Collecting the Points

1. Display the graph. (This example shows **y1(x)=5∗cos(x)**.)
2. Display the coordinates or math results you want to collect.
3. Save the information to the Home screen or the *sysData* variable. [♦] [(-)] (Home screen) or [♦] [,] (*sysData* variable)
4. Repeat the process as necessary.

   Note: To display coordinates or math results, trace a function with [F3] or perform an [F5] **Math** operation (such as **Minimum** or **Maximum**). You can also use the free-moving cursor.

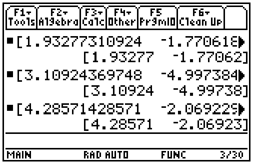
◆ (-)

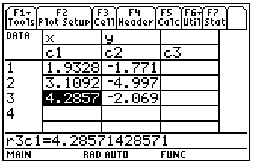


◆ ,

Displayed coordinates are added to the Home screen's history area (but not the entry line) as a single-row matrix or vector.

Displayed coordinates are stored in a data variable named *sysData*, which you can open in the Data/Matrix Editor.





Note: Use a split screen to show a graph and the Home screen or Data/Matrix Editor at the same time.

Notes about SysData Variable

- When you press:
  ◆ ,
  - If *sysData* does not exist, it is created in the **MAIN** folder.
  - If *sysData* already exists, new data is appended to the end of any existing data. Existing titles or column headers (for the affected columns) are cleared; titles are replaced with the applicable titles for the new data.
- The *sysData* variable can be cleared, deleted, etc., just as any other data variable. However, it cannot be locked.

- If the Graph screen contains a function or stat plot that references the current contents of *sysData*, this command will not operate.

Graphing a Function Defined on the Home Screen

In many cases, you may create a function or expression on the Home screen and then decide to graph it. You can copy an expression to the Y= Editor, or graph it directly from the Home screen without using the Y= Editor.

What Is the “Native” Independent Variable?

On the Y= Editor, all functions must be defined in terms of the current graph mode’s “native” independent variable.

Graph Mode	Native Independent Variable
Function	x
Parametric	t
Polar	θ
Sequence	n
3D	x, y
Differential Equation	t

If you have an expression on the Home screen, you can use any of the following methods to copy it to the Y= Editor.

Method	Description
Copy and paste	1. Highlight the expression on the Home screen. Press [F1] and select **5:Copy**. 2. Display the Y= Editor, highlight the desired function, and press [ENTER]. 3. Press [F1] and select **6:Paste**. Then press [ENTER]. **Note:** Instead of using [F1] **5** or [F1] **6** to copy and paste, use:[♦] [COPY] or [♦] [PASTE]
[STO▸]	Store the expression to a Y= function name. 2x^3+3x^2−4x+12→y1(x) Use the complete function name: y1(x), not just y1. **Note:** To copy an expression from the Home screen's history area to the entry line, use the auto-paste feature or copy and paste.
Define command	Define the expression as a user-defined Y= function. Define y1(x)=2x^3+3x^2−4x+12 **Note: Define** is available from the Home screen's [F4] toolbar menu.

Method	Description
[2nd] [RCL]	If the expression is already stored to a variable: 1. Display the Y= Editor, highlight the desired function, and press [ENTER]. 2. Press [2nd] [RCL]. Type the variable name that contains the expression, and press [ENTER] twice. **Important:** To recall a function variable such as **f1(x)**, type only **f1**, not the full function name. 3. Press [ENTER] to save the recalled expression in the Y= Editor's function list. **Note:** [2nd] [RCL] is useful if an expression is stored to a variable or function that does not correspond to the Y= Editor, such as **a1** or **f1(x)**.

Graphing Directly from the Home Screen

The **Graph** command lets you graph an expression from the Home screen without using the Y= Editor. Unlike the Y= Editor, **Graph** lets you specify an expression in terms of any independent variable, regardless of the current graphing mode.

If the expression is in terms of:	Use the Graph command as shown in this example:
The native independent variable	Graph 1.25x∗cos(x) For function graphing, x is the native variable.

If the expression is in terms of:	Use the Graph command as shown in this example:
A non-native independent variable	Graph 1.25a∗cos(a),a Specify the independent variable; otherwise, you may get an error.

Note: Graph uses the current Window variable settings and is available from the Home screen's [F4] toolbar menu.

Graph does not work with sequence graphs or differential equations. For parametric, polar, and 3D graphs, use the following variations.

In PARAMETRIC graphing mode:	**Graph** *xExpr*, *yExpr*, *t*
In POLAR graphing mode:	**Graph** *expr*, θ
In 3D graphing mode:	**Graph** *expr*, *x*, *y*

Note: To create a table from the Home screen, use the **Table** command. It is similar to **Graph**. Both share the same expressions.

Graph does not copy the expression to the Y= Editor. Instead, it temporarily suspends any functions selected on the Y= Editor. You can trace, zoom, or show and edit **Graph** expressions on the Table screen, just the same as Y= Editor functions.

Clearing the Graph Screen

Each time you execute **Graph**, the new expression is added to the existing ones. To clear the graphs:

- Execute the **ClrGraph** command (available from the Home screen's F4 **Other** toolbar menu).

  – or –

- Display the Y= Editor. The next time you display the Graph screen, it will use the functions selected on the Y= Editor.

Extra Benefits of User-Defined Functions

You can define a user-defined function in terms of any independent variable. For example:

Define f1(aa)=1.25aa cos(aa).

```
Define f1(aa)=1.25aa∗cos(aa)
Graph f1(x)
```

Refers to the function by using the native independent variable.

and:

```
Define f1(aa)=1.25aa∗cos(aa)
f1(x)→y1(x)
```

Graphing a Piecewise Defined Function

To graph a piecewise function, you must first define the function by specifying boundaries and expressions for each piece. The **when** function is extremely useful for two-piece functions. For three or more pieces, it may be easier to create a multi-statement, user-defined function.

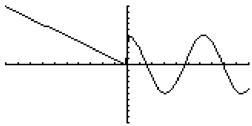
Using the When Function

To define a two-piece function, use the syntax:

when(*condition, trueExpression, falseExpression***)**

For example, suppose you want to graph a function with two pieces.

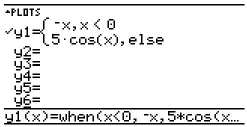
When:	Use expression:
x < 0	-x
x ≥ 0	5 cos(x)



In the Y= Editor:

The function is “pretty printed” in this form.

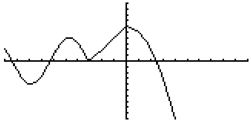
Enter the function in this form.



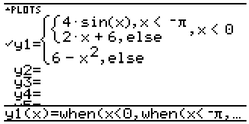
For three or more pieces, you can use nested **when** functions.

Note: To enter **when**, type it or use the **CATALOG**.

When:	Use expression:
x < ⁻π	4 sin(x)
x ≥ ⁻ π and x < 0	2x + 6
x ≥ 0	$6 - x^2$



In the Y= Editor:



where:

y1(x)=when(x<0,when(x<⁻π,4∗sin(x),2x+6),6−x^2)

This nested function is in effect when x<0.

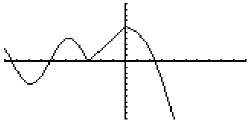
Nested functions quickly become complex and difficult to visualize.

Using a Multi-Statement, User-Defined Function

For three or more pieces, you may want to create a multi-statement, user-defined function.

For example, consider the previous three-piece function.

When:	Use expression:
x < -π	4 sin(x)
x ≥ - π and x < 0	2x + 6
x ≥ 0	$6 - x^2$



Note: For information about similarities and differences between functions and programs, refer to *Programming*.

A multi-statement, user-defined function can have many of the control and decision-making structures (**If**, **ElseIf**, **Return**, etc.) used in programming. When creating the structure of a function, it may be helpful to visualize it first in a block form.

```
❶ Func
    If x<-π Then
    Return 4*sin(x)
  ElseIf x>=-π and x<0 Then
    Return 2x+6
  Else
    Return 6–x^2
  EndIf
❶ EndFunc
```

❶ **Func** and **EndFunc** must begin and end the function.

When entering a multi-statement function on the Y= Editor or Home screen, you must enter the entire function on a single line.

Use a colon (:) to separate each statement.

Func:If x<⁻π Then:Return 4∗sin(x): ... :EndIf:EndFunc

In the Y= Editor:

Only **Func** is shown for a multi-statement function.

Enter a multi-statement function on one line. Be sure to include colons.

From the Home Screen or a Program

From the Home screen, you can also use the **Define** command to create a multi-statement, user-defined function.

Information is available on copying a function from the Home screen to the Y= Editor.

From the Program Editor, you can create a user-defined function. For example, use the Program Editor to create a function named **f1(xx)**. In the Y= Editor, set **y1(x) = f1(x)**.

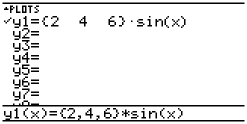
Graphing a Family of Curves

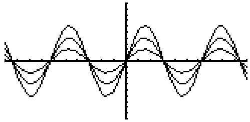
By entering a list in an expression, you can plot a separate function for each value in the list. (You cannot graph a family of curves in SEQUENCE or 3D graphing mode.)

Examples Using the Y= Editor

Enter the expression **{2,4,6} sin(x)** and graph the functions.

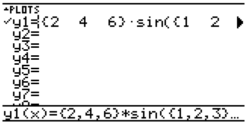
Note: Enclose list elements in braces ([2nd] [{] and [2nd] [}]) and separate them with commas.

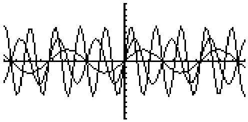




Graphs three functions:
2 sin(x), 4 sin(x), 6 sin(x)

Enter the expression **{2,4,6} sin({1,2,3} x)** and graph the functions.





Graphs three functions:
2 sin(x), 4 sin(2x), 6 sin(3x)

Note: The commas are shown in the entry line but not in the function list.

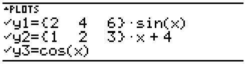
Example Using the Graph Command

Similarly, you can use the Graph command from the Home screen or a program.

```
graph {2,4,6}sin(x)
graph {2,4,6}sin({1,2,3}x)
```

Simultaneous Graphs with Lists

When the graph format is set for **Graph Order = SIMUL**, the functions are graphed in groups according to the element number in the list.



For these example functions, the TI-89 Titanium / Voyage™ 200 graphing calculator graphs three groups.

- 2 sin(x), x+4, cos(x)
- 4 sin(x), 2x+4
- 6 sin(x), 3x+4

The functions within each group are graphed simultaneously, but the groups are graphed sequentially.

Note: To set graph formats from the Y= Editor, Window Editor, or Graph screen, press: [◆] [|]

When Tracing a Family of Curves

Pressing ⊖ or ⊕ moves the trace cursor to the next or previous curve in the same family before moving to the next or previous selected function.

Using the Two-Graph Mode

In two-graph mode, the calculator's graph-related features are duplicated, giving you two independent graphing calculators. The two-graph mode is only available in split screen mode. For more information about split screens, refer to *Split Screens*.

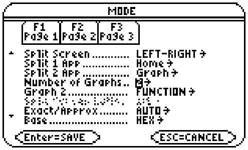
Setting the Mode

Several mode settings affect the two-graph mode, but only two settings are required. Both are on **Page 2** of the **MODE** dialog box.

1. Press [MODE]. Then press [F2] to display **Page 2**.

2. Set the following required modes.

   - **Split Screen = TOP-BOTTOM** or **LEFT-RIGHT**
   - **Number of Graphs = 2**



3. Optionally, you can set the following modes.

Page 1:
- **Graph = Graph** mode for top or left side of the split

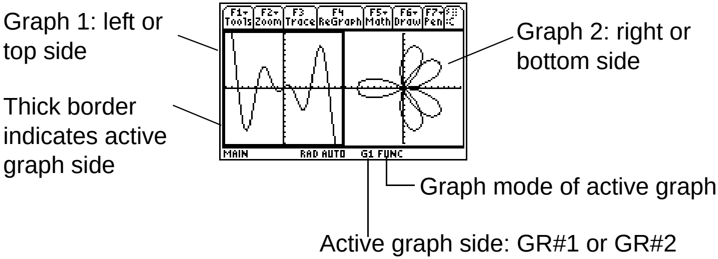
Page 2:

- **Split 1 App** = application for top or left side
- **Split 2 App** = application for bottom or right side
- **Graph 2 = Graph** mode for bottom or right side
-

4. Press [ENTER] to close the dialog box.

The Two-Graph Screen

A two-graph screen is similar to a regular split screen.



Independent Graph-Related Features

Both Graph 1 and Graph 2 have independent:

- Graph modes (FUNCTION, POLAR, etc.). Other modes such as **Angle**, **Display Digits**, etc., are shared and affect both graphs.

- Window Editor variables.
- Table setup parameters and Table screens.
- Graph formats such as **Coordinates**, **Axes**, etc.
- Graph screens.
- Y= Editors. However, both graphs share common function and stat plot definitions.

  Note: The Y= Editor is completely independent only when the two sides use different graphing modes (as described below).

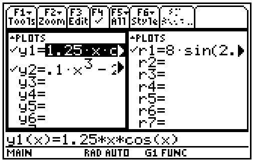
Independent graph-related applications (Y= Editor, Graph screen, etc.) can be displayed on both sides of the screen at the same time.

Non-graph-related applications (Home screen, Data/Matrix Editor, etc.) are shared and can be displayed on only one side at a time.

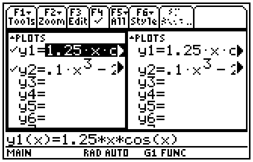
The Y= Editor in Two-Graph Mode

Even in two-graph mode, there is actually only one Y= Editor, which maintains a single function list for each Graph mode setting. However, if both sides use the same graphing mode, each side can select different functions from that single list.

- When both sides use different graphing modes, each side shows a different function list.



- When both sides use the same graphing mode, each side shows the same function list.
  - You can use [F4] to select different functions and stat plots (indicated by ✓) for each side.
  - If you set a display style for a function, that style is used by both sides.[2nd] [F6]



- Suppose Graph 1 and Graph 2 are set for function graphing. Although both sides show the same function list, you can select (✓) different functions for graphing

Note: If you make a change on the active Y= Editor (redefine a function, change a style, etc.), that change is not reflected on the inactive side until you switch to it.

Using a Split Screen

For more complete information about split screens, refer to *Split Screens*.

- To switch from one graph side to the other, press [2nd] [⊞] (second function of [APPS]).
- To display different applications:
  - Switch to the applicable graph side and display the application as you normally would.

    – or –
  - Use [MODE] to change **Split 1 App** and/or **Split 2 App**.
- To exit two-graph mode:

- Use [MODE] to set **Number of Graphs = 1**, or exit the split screen by setting **Split Screen = FULL**.

  – or –

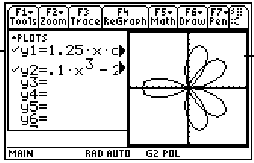
- Press [2nd] [QUIT] twice. This always exits a split screen and returns to a full-sized Home screen.

Note: You can display non-graph-related applications (such as the Home screen) on only one side at a time.

Remember that the Two Sides Are Independent

In two-graph mode, the two sides may appear to be related when, in fact, they are not. For example:

For Graph 1, the Y= Editor lists y(x) functions.



For Graph 2, the polar graph uses r(θ) equations that are not shown.

After the two-graph mode is set up, graph-related operations refer to the active graph side. For example:

```
10→xmax
```

affects either Graph 1 or Graph 2, depending on which is active when you execute the command.

To switch the active sides, press [2nd] [⊞] or use the **switch** function, **switch(1)** or **switch(2)**.

Drawing a Function or Inverse on a Graph

For comparison purposes, you may want to draw a function over your current graph. Typically, the drawn function is some variation of the graph. You can also draw the inverse of a function. (These operations are not available for 3D graphs.)

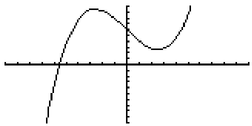
Drawing a Function, Parametric, or Polar Equation

Execute **DrawFunc**, **DrawParm**, or **DrawPol** from the Home screen or a program. You cannot draw a function or equation interactively from the Graph screen.

DrawFunc *expression*
DrawParm *expression1*, *expression2* [,*tmin*] [,*tmax*] [,*tstep*]
DrawPol *expression* [,θ*min*] [,θ*max*] [,θ*step*]

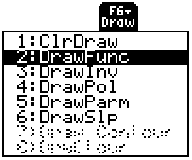
For example:

1. Define **y1(x)=.1x^3–2x+6** on the Y= Editor, and graph the function.



2. On the Graph screen, press: [2nd] [F6] and select **2:DrawFunc**.

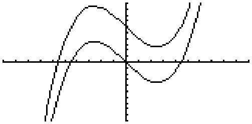
   To display the Home screen and put **DrawFunc** in the entry line, press:[2nd] [F6] **2**



3. On the Home screen, specify the function to draw.

   DrawFunc y1(x)−6

4. Press [ENTER] to draw the function on the Graph screen.



   You cannot trace, zoom, or perform a math operation on a drawn function.

   Note: To clear the drawn function, press
   [F4]
   – or –
   [2nd] [F6] and select **1:ClrDraw**

Drawing the Inverse of a Function

Execute **DrawInv** from the Home screen or a program. You cannot draw an inverse function interactively from the Graph screen.

DrawInv *expression*

For example, use the graph of **y1(x)=.1x**^{3}**–2x+6** as shown above.

1. On the Graph screen, press: [2nd] [F6] and select **3:DrawInv**

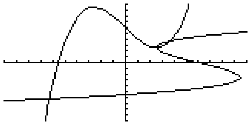
   To display the Home screen and put **DrawInv** in the entry line, press: [2nd] [F6] **3**

2. On the Home screen, specify the inverse function.

   DrawInv y1(x)

3. Press [ENTER].

   The inverse is plotted as **(y,x)** instead of **(x,y)**.



Drawing a Line, Circle, or Text Label on a Graph

You can draw one or more objects on the Graph screen, usually for comparisons. For example, draw a horizontal line to show that two parts of a graph have the same y value. (Some objects are not available for 3D graphs.)

Clearing All Drawings

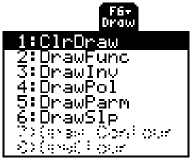
A drawn object is not part of the graph itself. It is drawn “on top of” the graph and remains on the screen until you clear it.

From the Graph screen:

- [2nd] [F6]
  and select **1:ClrDraw.**

  – or –

- Press [F4] to regraph.

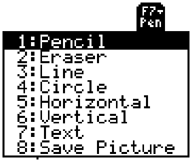


Note: You can also enter **ClrDraw** on the Home screen's entry line.

You can also do anything that causes the Smart Graph feature to redraw the graph (such as change the Window variables or deselect a function on the Y= Editor).

From the Graph screen:

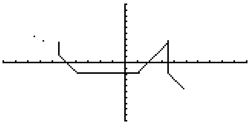
1. [2nd] [F7] and select **1:Pencil**.
2. Move the cursor to the applicable location.



To draw a:	Do this:
Point (pixel-sized)	Press [ENTER].
Freehand line	Press and hold [↑], and move the cursor to draw the line.

Note: When drawing a freehand line, you can move the cursor diagonally.

After drawing the point or line, you are still in **Pencil** mode.



- To continue drawing, move the cursor to another point.
- To quit, press [ESC].

Note: If you start drawing on a white pixel, the pencil draws a black point or line. If you start on a black pixel, the pencil draws a white point or line (which can act as an eraser).

Erasing Individual Parts of a Drawing Object

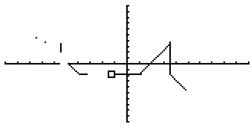
From the Graph screen:

1. [2nd] [F7]
   and select **2:Eraser**. The cursor is shown as a small box.
2. Move the cursor to the applicable location.

To erase:	Do this:
Area under the box	Press [ENTER].
Along a freehand line	Press and hold [↑], and move the cursor to erase the line.

Note: These techniques also erase parts of graphed functions.

After erasing, you are still in **Eraser** mode.



- To continue erasing, move the box cursor to another location.
- To quit, press [ESC].

Drawing a Line Between Two Points

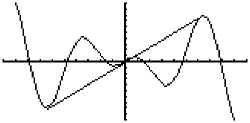
From the Graph screen:

1. [2nd] [F7]
   and select **3:Line**.

2. Move the cursor to the 1st point, and press [ENTER].
3. Move to the 2nd point, and press [ENTER]. (As you move, a line extends from the 1st point to the cursor.)

   Note: Use [2nd] to move the cursor in larger increments; [2nd] ▶, etc.

After drawing the line, you are still in **Line** mode.

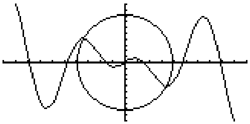


- To continue drawing another line, move the cursor to a new 1st point.
- To quit, press [ESC].

Drawing a Circle

From the Graph screen:

1. [2nd] [F7]
   and select **4:Circle**.
2. Move the cursor to the center of the circle, and press [ENTER].
3. Move the cursor to set the radius, and press [ENTER].



   Note: Use [2nd] to move the cursor in larger increments; [2nd] ▶, etc.

Drawing a Horizontal or Vertical Line

From the Graph screen:

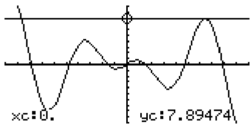
1. [2nd] [F7]
   and select **5:Horizontal** or **6:Vertical**. A horizontal or vertical line and a flashing cursor are displayed on the screen.

   If the line is initially displayed on an axis, it may be difficult to see. However, you can easily see the flashing cursor.

2. Use the cursor pad to move the line to the appropriate position. Then press [ENTER].

   After drawing the line, you are still in "line" mode.

   - To continue, move the cursor to another location.
   - To quit, press [ESC].



Note: Use [2nd] to move the cursor in larger increments; [2nd] ▶, etc.

Drawing a Tangent Line

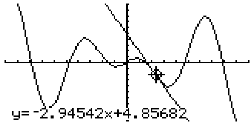
To draw a tangent line, use the [F5] **Math** toolbar menu. From the Graph screen:

1. Press [F5] and select **A:Tangent**.

2. As necessary, use ⊙ and ⊙ to select the applicable function.

3. Move the cursor to the tangent point, and press [ENTER].

   The tangent line is drawn, and its equation is displayed.



Note: To set the tangent point, you can also type its **x** value and press [ENTER].

Drawing a Line Based on a Point and a Slope

To draw a line through a specified point with a specified slope, execute the **DrawSlp** command from the Home screen or a program. Use the syntax:

DrawSlp *x*, *y*, *slope*

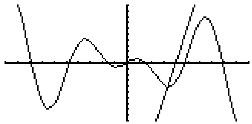
You can also access **DrawSlp** from the Graph screen.

1. [2nd] [F6]
   and select **6:DrawSlp**. This switches to the Home screen and puts **DrawSlp** in the entry line.

2. Complete the command, and press [ENTER].

   The calculator automatically switches to the **Graph** screen and draws the line.

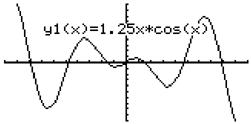
DrawSlp 4,0,6.37



Typing Text Labels

From the Graph screen:

1. [2nd] [F7] and select **7:Text**.
2. Move the text cursor to the location where you want to begin typing.
3. Type the text label.

   After typing the text, you are still in "text" mode.

   - To continue, move the cursor to another location.
   - To quit, press [ENTER] or [ESC].

Note: The text cursor indicates the upper-left corner of the next character you type.

From the Home Screen or a Program

Commands are available for drawing any of the objects described in this section. There are also commands (such as **PxlOn**, **PxlLine**, etc.) that let you draw objects by specifying exact pixel locations on the screen.

For a list of the available drawing commands, refer to "Drawing on the Graph Screen" in *Programming*.

Saving and Opening a Picture of a Graph

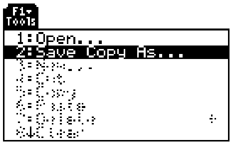
You can save an image of the current Graph screen in a PICTURE (or PIC) variable. Then, at a later time, you can open that variable and display the image. This saves the image only, not the graph settings used to produce it.

Saving a Picture of the Whole Graph Screen

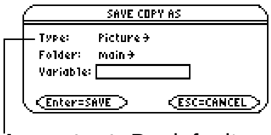
A picture includes any plotted functions, axes, tick marks, and drawn objects. The picture does not include lower and upper bound indicators, prompts, or cursor coordinates.

Display the Graph screen as you want to save it; and then:

1. Press [F1] and select **2:Save Copy As**.



2. Specify the type (**Picture**), folder, and a unique variable name.
3. Press [ENTER]. After typing in an input box such as **Variable**, you must press [ENTER] twice.



Important: By default, Type = GDB (for graph database). You must set Type = Picture.

Saving a Portion of the Graph Screen

You can define a rectangular box that encloses only the portion of the Graph screen that you want to save.

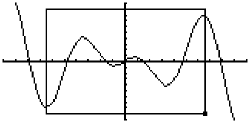
1. [2nd] [F7] and select **8:Save Picture**.

   A box is shown around the outer edge of the screen.

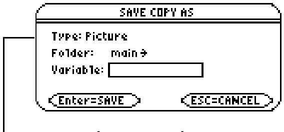
   Note: You cannot save a portion of a 3D graph.

2. Set the 1st corner of the box by moving its top and left sides. Then press [ENTER].

   Note: Use ↓ and ↑ to move the top or bottom, and use → and ← to move the sides.



3. Set the 2nd corner by moving the bottom and right sides. Then press [ENTER].
4. Specify the folder and a unique variable name.
5. Press [ENTER]. After typing in an input box such as **Variable**, you must press [ENTER] twice.



Note: When saving a portion of a graph, Type is automatically fixed as Picture.

Opening a Graph Picture

When you open a graph picture, it is superimposed over the current Graph screen. To display only the picture, use the Y= Editor to deselect any other functions before opening the graph picture.

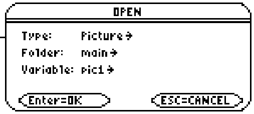
From the Graph screen:

1. Press [F1] and select **1:Open**.

2. Select the type (**Picture**), folder, and variable that contain the graph picture you want to open.

   Note: If a variable name is not shown on the dialog box, there are no graph pictures in the folder.

3. Press [ENTER].



Important: By default, Type = GDB (for graph database). Be sure to set Type = Picture.

A graph picture is a drawing object. You cannot trace any curve on a picture.

For Pictures Saved from a Portion of the Graph Screen

When you press [F1] and select **1:Open**, the picture is superimposed starting at the upper-left corner of the Graph screen. If the picture was saved from a portion of the Graph screen, it may appear shifted from the underlying graph.

You can specify which screen pixel to use as the upper-left corner.

Deleting a Graph Picture

Unwanted Picture variables take up calculator memory. To delete a variable, use the VAR-LINK screen ([2nd] [VAR-LINK]) as described in *Memory and Variable Management*.

From a Program or the Home Screen

To save (**store**) and open (**recall**) a graph picture, use the **StoPic**, **RclPic**, **AndPic**, **XorPic**, and **RplcPic** commands as described in the *Technical Reference* module.

To display a series of graph pictures as an animation, use the **CyclePic** command. For an example, refer to CyclePic Command.

Animating a Series of Graph Pictures

As described earlier in this module, you can save a picture of a graph. By using the **CyclePic** command, you can flip through a series of graph pictures to create an animation.

CyclePic Command

Before using **CyclePic**, you must have a series of graph pictures that have the same base name and are sequentially numbered starting with 1 (such as pic1, pic2, pic3, . . .).

To cycle the pictures, use the syntax:

CyclePic *picNameString*, *n* [,*wait*] [,*cycles*] [,*direction*]
❶ ❷ ❸ ❹ ❺

❶ base name of pictures in quotes, such as "pic"
❷ # of pictures to cycle
❸ seconds between
❹ # of times to repeat cycle

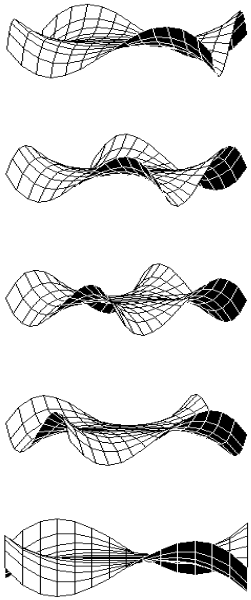
Example

This example program (named **cyc**) generates 10 views of a 3D graph, with each view rotated 10 ° further around the Z axis. For information about each command, refer to the

Technical Reference module. For information about using the Program Editor, refer to *Programming*.

Program Listing	Every Other Graph from Program

```
:cyc()
:Prgm
:local I
:©Set mode and Window variables
:setMode("graph","3d")
:70→eyeφ
:-10→xmin
:10→xmax
:14→xgrid
:-10→ymin
:10→ymax
:14→ygrid
:-10→zmin
:10→zmax
:1→zscl
:©Define the function
:(x^3*y-y^3*x)/390→z1(x,y)
:©Generate pics and rotate
:For i,1,10,1
:  i*10→eyeθ
:  DispG
:  StoPic #("pic" & string(i))
:EndFor
:©Display animation
:CyclePic "pic",10,.5,5,-1
:EndPrgm
```



Comments start with ©. Press:
[♦] [)]

Note: Due to its complexity, this program takes several minutes to run.

After entering this program on the Program Editor, go to the Home screen and enter **cyc( )**.

Saving and Opening a Graph Database

A graph database is the set of all elements that define a particular graph. By saving a graph database as a GDB variable, you can recreate that graph at a later time by opening its stored database variable.

Elements in a Graph Database

A graph database consists of:

- Mode settings ([MODE]) for **Graph**, **Angle**, **Complex Format**, and **Split Screen** (only if you are using the two-graph mode).
- All functions in the Y= Editor ([♦] [Y=]), including display styles and which functions are selected.
- Table parameters ([♦] [TBLSET]), Window variables ([♦] [WINDOW]), and graph formats:
  [F1] **9**
  – or –
  [♦] [|]

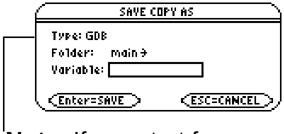
A graph database does not include drawn objects or stat plots.

Note: In two-graph mode, the elements for both graphs are saved in a single database.

Saving the Current Graph Database

From the Y= Editor, Window Editor, Table screen, or Graph screen:

1. Press [F1] and select **2:Save Copy As**.
2. Specify the folder and a unique variable name.
3. Press [ENTER]. After typing in an input box such as Variable, you must press [ENTER] twice.



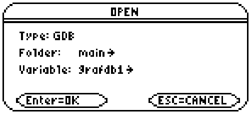
Note: If you start from the Graph screen, be sure to use Type=GDB.

Opening a Graph Database

Caution: When you open a graph database, all information in the current database is replaced. You may want to store the current graph database before opening a stored database.

From the Y= Editor, Window Editor, Table screen, or Graph screen:

1. Press [F1] and select **1:Open**.
2. Select the folder and variable that contain the graph database you want to open.
3. Press [ENTER].



Note: If you start from the Graph screen, be sure to use Type=GDB.

Deleting a Graph Database

Unused GDB variables take up calculator memory. To delete them, use the VAR-LINK screen ([2nd] [VAR-LINK]) described in *Memory and Variable Management*.

From a Program or the Home Screen

You can save (**store**) and open (**recall**) a graph database by using the **StoGDB** and **RclGDB** commands as described in the *Technical Reference* module.

Split Screens

Setting and Exiting the Split Screen Mode

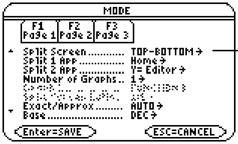
To set up a split screen, use the MODE dialog box to specify the applicable mode settings. After you set up the split screen, it remains in effect until you change it.

Setting the Split Screen Mode

1. Press [MODE] to display the **MODE** dialog box.
2. Because the modes related to split screens are listed on the second page of the **MODE** dialog box, either:
   - Use ⊖ to scroll down.

     — or —
   - Press [F2] to display **Page 2**.
3. Set the **Split Screen** mode to either of the following settings. For the procedure used to change a mode setting, refer to *Operating the Calculator*.

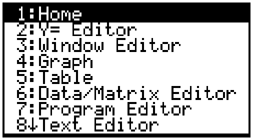
Split Screen Settings
TOP-BOTTOM
LEFT-RIGHT



When you set Split Screen = TOP-BOTTOM or LEFT-RIGHT, previously dimmed modes such as Split 2 App become active.

Setting the Initial Applications

Before pressing [ENTER] to close the MODE dialog box, you can use the **Split 1 App** and **Split 2 App** modes to select the applications you want to use.



Mode	Specifies the application in the:
Split 1 App	Top or left part of the split screen.
Split 2 App	Bottom or right part of the split screen.

If you set **Split 1 App** and **Split 2 App** to the same application, the calculator exits the split screen mode and displays the application full screen.

You can open different applications after the split screen is displayed.

Note: In two-graph mode, described in *Additional Graphing Topics*, the same application can be in both parts of a split screen.

Other Modes that Affect a Split Screen

Mode	Description
Number of Graphs **Note:** Leave this set to 1 unless you have read the applicable section in *Additional Graphing Topics*.	Lets you set up and display two independent sets of graphs. This is an advanced graphing feature as described in “Using the Two-Graph Mode” in *Additional Graphing Topics*.

Split Screens and Pixel Coordinates

The calculator has commands that use pixel coordinates to draw lines, circles, etc., on the Graph screen. The following charts show how the **Split Screen** and **Split Screen Ratio** mode settings affect the number of pixels available on the Graph screen.

Note:

- For a list of drawing commands, refer to “Drawing on the Graph Screen” in *Programming*.
- Due to the border that indicates the active application, split screens have a smaller displayable area than a full screen.

TI-89 Titanium:

		Split 1 App		Split 2 App	
Split	**Ratio**	**x**	**y**	**x**	**y**
FULL	N/A	0 – 158	0 – 76	N/A	N/A

		Split 1 App		Split 2 App	
Split	**Ratio**	**x**	**y**	**x**	**y**
TOP–BOTTOM	1:1	0 – 154	0 – 34	0 – 154	0 – 34
LEFT–RIGHT	1:1	0 – 76	0 – 72	0 – 76	0 – 72

Voyage™ 200:

		Split 1 App		Split 2 App	
Split	**Ratio**	**x**	**y**	**x**	**y**
FULL	N/A	0 – 238	0 – 102	N/A	N/A
TOP–BOTTOM	1:1	0 – 234	0 – 46	0 – 234	0 – 46
	1:2	0 – 234	0 – 26	0 – 234	0 – 68
	2:1	0 – 234	0 – 68	0 – 234	0 – 26
LEFT–RIGHT	1:1	0 – 116	0 – 98	0 –116	0 – 98
	1:2	0 – 76	0 – 98	0 – 156	0 – 98
	2:1	0 – 156	0 – 98	0 – 76	0 – 98

Exiting the Split Screen Mode

Method 1: Press [MODE] to display the MODE dialog box. Then set **Split Screen = FULL.** When you press [ENTER] to close the dialog box, the full-sized screen shows the application specified in **Split 1 App**.

Method 2: Press [2nd] [QUIT] twice to display a full-sized Home screen.

When You Turn Off the Calculator

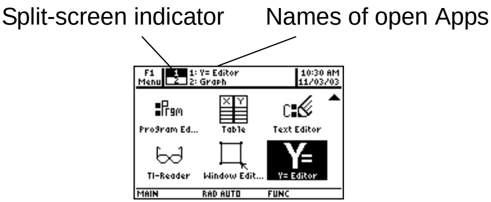
Turning the calculator off does not exit the split screen mode.

If the calculator is turned off:	When you turn the calculator on again:
When you press [2nd] [OFF]	The split screen is still in effect, but the Home screen is always displayed in place of the application that was active when you pressed [2nd] [OFF].
By the Automatic Power Down™ (APD™) feature, or when you press [♦] [OFF].	The split screen is just as you left it.

Split-Screen Status Indicators on the Apps Desktop

To return to the Apps desktop, press [APPS]. The split-screen status appears at the top of the Apps desktop with the names of the open Apps and the portions of the screen in which each App is displayed.

Note: The Apps desktop always appears in the full-screen view.



Split screen indicator	Description
	Top-bottom split screen • **1** indicates the application that will appear in the top portion of the screen. • **2** indicates the application that will appear in the bottom portion of the screen. The highlighted numeral indicates the active portion of the split screen.
	Left-right split screen • **1** indicates the application that will appear in the left portion of the screen. • **2** indicates the application that will appear in the right portion of the screen. The highlighted numeral indicates the active portion of the split screen.

Selecting the Active Application

With a split screen, only one of the two applications can be active at a time. You can easily switch between existing applications, or you can open a different application.

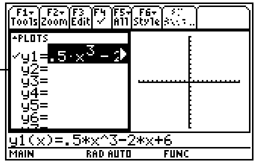
The Active Application

- The active application is indicated by a thick border.
- The toolbar and status line, which are always the full width of the display, are associated with the active application.
- For applications that have an entry line (such as the Home screen and Y= Editor), the entry line is the full width of the display only when that application is active.

Toolbar is for Y= Editor.

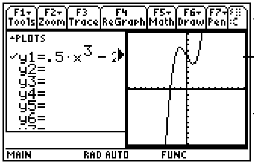
Thick border indicates the Y= Editor is active.

Entry line is full width when Y= Editor is active.



Switching between Applications

Press [2nd] [⇹] (second function of [APPS]) to switch from one application to the other.



Toolbar is for Graph screen.

Thick border indicates the Graph screen is active.

Graph screen does not have an entry line.

Opening a Different Application

Method 1:

1. Use [2nd] [⇔] to switch to the application you want to replace.
2. Use [APPS] or [♦] (such as [♦] [WINDOW]) to select the new application.

If you select an application that is already displayed, the calculator switches to that application.

Method 2:

3. Press [MODE] and then [F2].
4. Change **Split 1 App** and/or **Split 2 App**.

   If you set **Split 1 App** and **Split 2 App** to the same application, the calculator exits the split screen mode and displays the application full screen.

Note: In two-graph mode, described in *Additional Graphing Topics*, the same application can be in both parts of a split screen.

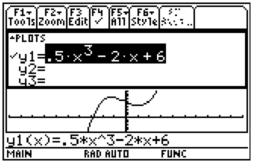
Note: Pressing [2nd] [QUIT] twice always exits the split screen mode.

If the Home screen:	**Pressing [2nd] [QUIT]:**
Is not already displayed	Opens the Home screen in place of the active application.
Is displayed, but is not the active application	Switches to the Home screen and makes it the active application.
Is the active application	Exits the split screen mode and displays a full-sized Home screen.

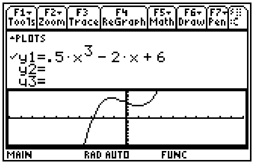
When Using a Top-Bottom Split

When you select a TOP-BOTTOM split, remember that the entry line and the toolbar are always associated with the active application. For example:

Entry line is for the active Y= Editor, *not* the Graph screen.



Toolbar is for the active Graph screen, *not* the Y= Editor.



Note: Both **Top-Bottom** and **Left-Right** splits use the same methods to select an application.

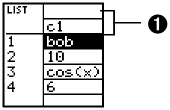
Data/Matrix Editor

Overview of List, Data, and Matrix Variables

To use the Data/Matrix Editor effectively, you must understand list, data, and matrix variables.

List Variable

A list is a series of items (numbers, expressions, or character strings) that may or may not be related. Each item is called an element. In the Data/Matrix Editor, a list variable:



- Is shown as a single column of elements, each in a separate cell.
- Must be continuous; blank or empty cells are not allowed within the list.
- Can have up to 999 elements.

❶ Column title and header cells are not saved as part of the list.

If you enter more than one column of elements in a list variable, it is converted automatically into a data variable.

On the Home screen (or anywhere else you can use a list), you can enter a list as a series of elements enclosed in braces { } and separated by commas.

Although you must use commas to separate elements on the entry line, spaces separate the elements in the history area.

To refer to a specified element in a list, use the format shown to the right.

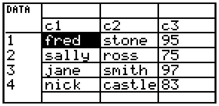
❶ Name of list variable
❷ Element number (or index number)

Note: After creating a list in the Data/Matrix Editor, you can use the list in any application (such as the Home screen).

Data Variable

A data variable is essentially a collection of lists that may or may not be related. In the Data/Matrix Editor, a data variable:

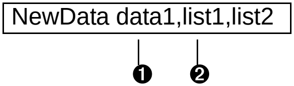
- Can have up to 99 columns.
- Can have up to 999 elements in each column. Depending on the kind of data, all columns may not have to be the same length.



- Must have continuous columns; blank or empty cells are not allowed within a column.

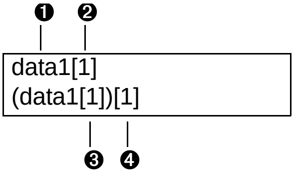
Note: For stat calculations, columns must have the same length.

From the Home screen or a program, you can use the **NewData** command to create a data variable that consists of existing lists.



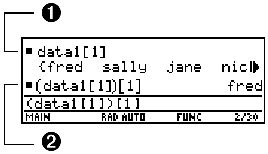
❶ Name of data variable to create
❷ Names of existing list

Although you cannot directly display a data variable on the Home screen, you can display a specified column or element.



❶ Name of data variable
❷ Column number
❸ Column number
❹ Element number in the column

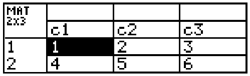
For example:



❶ Displays column 1 of the variable data1.
❷ Displays element 1 in column 1 of the variable data1.

A matrix is a rectangular array of elements. When you create a matrix in the Data/Matrix Editor, you must specify the number of rows and columns (although you can add or delete rows and columns later). In the Data/Matrix Editor, a matrix variable:

- Looks similar to a data variable, but all columns must have the same length.
- Is initially created with 0 in each cell. You can then enter the applicable value in place of 0.



Shows the size of the matrix.

From the Home screen or a program, you can use [STO▸] to store a matrix with either of the equivalent methods shown to the right.

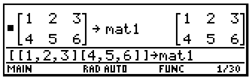
❶ ❷

[[1,2,3][4,5,6]]→mat1
[1,2,3;4,5,6]→mat1

❸ ❹

❶ row 1
❷ row 2
❸ row 1
❹ row 2

Although you enter the matrix as shown above, it is pretty printed in the history area in traditional matrix form.



After creating a matrix in the Data/Matrix Editor, you can use the matrix in any application (such as the Home screen).

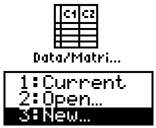
Note: Use brackets to refer to a specific element in a matrix. For example, enter **mat1[2,1]** to access the 1st element in the 2nd row.

Starting a Data/Matrix Editor Session

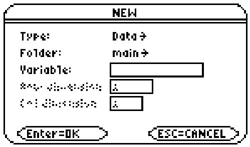
Each time you start the Data/Matrix Editor, you can create a new variable, resume using the current variable (the variable that was displayed the last time you used the Data/Matrix Editor), or open an existing variable.

Creating a New Data, Matrix, or List Variable

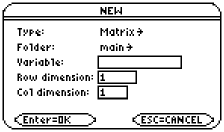
1. Press [APPS] and then select the Data/Matrix icon. Press [ENTER].
2. Select **3:New.**



3. Specify the applicable information for the new variable.



Item	Lets you:
Type	Select the type of variable to create. Press ▸ to display a menu of available types. 
Folder	Select the folder in which the new variable will be stored. Press ▸ to display a menu of existing folders. For information about folders, refer to *the Calculator Home Screen module*.

Item	Lets you:
Variable	Type a new variable name. If you specify a variable that already exists, an error message will be displayed when you press [ENTER]. When you press [ESC] or [ENTER] to acknowledge the error, the **NEW** dialog box is redisplayed.
Row dimension and Col dimension	If Type = Matrix, type the number of rows and columns in the matrix. 

Note: If you do not type a variable name, your calculator displays the Home screen.

4. Press [ENTER] (after typing in an input box such as **Variable**, press [ENTER] twice) to create and display an empty variable in the Data/Matrix Editor.

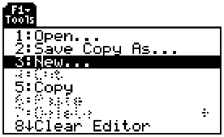
Using the Current Variable

You can leave the Data/Matrix Editor and go to another application at any time. To return to the variable that was displayed when you left the Data/Matrix Editor, launch Data/Matrix Editor again and select **1:Current**.

Creating a New Variable from the Data/Matrix Editor

From the Data/Matrix Editor:

1. Press [F1] and select **3:New.**
2. Specify the type, folder, and variable name. For a matrix, also specify the number of rows and columns.



Opening Another Variable

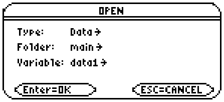
You can open another variable at any time.

1. From the Data/Matrix Editor, press [F1] and select **1:Open**.

   – or –

   From any application, launch Data/Matrix Editor again and select **2:Open.**

2. Select the type, folder, and variable to open.
3. Press [ENTER].



Note: Variable shows the first existing variable in alphabetic order. If there are no existing variables, nothing is displayed.

Deleting a Variable

Because all Data/Matrix Editor variables are saved automatically, you can accumulate quite a few variables, which take up memory.

To delete a variable, use the VAR-LINK screen ([2nd] [VAR-LINK]). For information about VAR-LINK, refer to *Memory and Variable Management*.

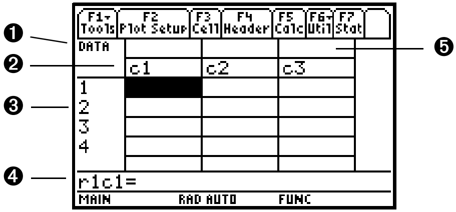
Entering and Viewing Cell Values

If you create a new variable, the Data/Matrix Editor is initially blank (for a list or data variable) or filled with zeros (for a matrix). If you open an existing variable, the values in that variable are displayed. You can then enter additional values or edit the existing ones.

The Data/Matrix Editor Screen

A blank Data/Matrix Editor screen is shown below. When the screen is displayed initially, the cursor highlights the cell at row 1, column 1.

❶ Variable type
❷ Column headers
❸ Row numbers
❹ Row and column number of highlighted cell
❺ Column title cells, used to type a title for each column



When values are entered, the entry line shows the full value of the highlighted cell.

Note: Use the title cell at the very top of each column to identify the information in that column.

Entering or Editing a Value in a Cell

You can enter any type of expression in a cell (number, variable, function, string, etc.).

1. Move the cursor to highlight the cell you want to enter or edit.
2. Press [ENTER] or [F3] to move the cursor to the entry line.
3. Type a new value or edit the existing one.
4. Press [ENTER] to enter the value into the highlighted cell.

When you press [ENTER], the cursor automatically moves to highlight the next cell so that you can continue entering or editing values. However, the variable type affects the direction that the cursor moves.

Note: To enter a new value, you can start typing without pressing [ENTER] or [F3] first. However, you must use [ENTER] or [F3] to edit an existing value.

Variable Type	After pressing [ENTER], the cursor moves:
List or data	Down to the cell in the next row.
Matrix	Right to the cell in the next column. From the last cell in a row, the cursor automatically moves to the first cell in the next row. This lets you enter values for row1, row2, etc.

To move the cursor:	Press:
One cell at a time	⊽, ⊼, ▷, or ◁
One page at a time	[2nd] and then ⊽, ⊼, ▷, or ◁
Go to row 1 in the current column or to the last row that contains data for any column on the screen, respectively. If the cursor is in or past that last row, [♦] ⊽ goes to row 999.	[♦] ⊼ or [♦] ⊽
Go to column 1 or to the last column that contains data, respectively. If the cursor is in or past that last column, [♦] ▷ goes to column 99.	[♦] ◁ or [♦] ▷

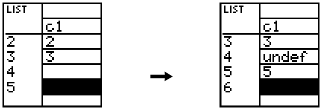
Note: To enter a value from the entry line, you can also use ⊽ or ⊼.

When you scroll down/up, the header row remains at the top of the screen so that the column numbers are always visible. When you scroll right/left, the row numbers remain on the left side of the screen so that they are always visible.

How Rows and Columns Are Filled Automatically

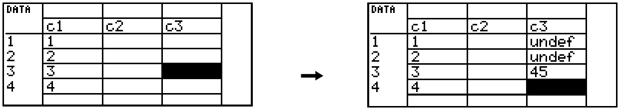
When you enter a value in a cell, the cursor moves to the next cell. However, you can move the cursor to any cell and enter a value. If you leave gaps between cells, your device handles the gaps automatically.

- In a list variable, a cell in the gap is undefined until you enter a value for the cell.

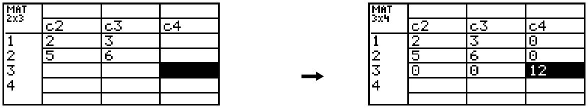


Note: If you enter more than one column of elements in a list variable, it is converted automatically into a data variable.

- In a data variable, gaps in a column are handled the same as a list. However, if you leave a gap between columns, that column is blank.



- In a matrix variable, when you enter a value in a cell outside the current boundaries, additional rows and/or columns are added automatically to the matrix to include the new cell. Other cells in the new rows and/or columns are filled with zeros.

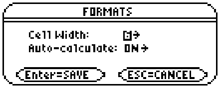


Note: Although you specify the size of a matrix when you create it, you can easily add additional rows and/or columns.

Changing the Cell Width

The cell width affects how many characters are displayed in any cell. To change the cell width in the Data/Matrix Editor:

1. To display the **FORMATS** dialog box, press:
   [F1] **9**
   – or –
   [♦] [I]

   Cell width is the maximum number of characters that can be displayed in a cell. All cells have the same cell width.

   Note: Remember, to see a number in full precision, you can always highlight the cell and look at the entry line.

2. With the current **Cell Width** setting highlighted, press ▶ or ◀ to display a menu of digits (**3** through **12**).
3. Move the cursor to highlight a number and press [ENTER]. (For single-digit numbers, you can type the number and press [ENTER].)
4. Press [ENTER] to close the dialog box.

Clearing a Column or all Columns

This procedure erases the contents of a column. It does not delete the column.

To clear:	Do this:
A column	1. Move the cursor to any cell in the column. 2. Press: [2nd] [F6] and select **5:Clear Column**. (This item is not available for a matrix.)
All columns	Press [F1] and select **8:Clear Editor**. When prompted for confirmation, press [ENTER] (or [ESC] to cancel).

Note: For a list or data variable, a clear column is empty. For a matrix, a clear column contains zeros.

Defining a Column Header with an Expression

For a list variable or a column in a data variable, you can enter a function in the column header that automatically generates a list of elements. In a data variable, you can also define one column in terms of another.

Entering a Header Definition

In the Data/Matrix Editor:

1. Move the cursor to any cell in the column and press [F4].

   – or –

   Move the cursor to the header cell (**c1**, **c2**, etc.) and press [ENTER].

   Notes:

   - [ENTER] is not required if you want to type a new definition or replace the existing one. However, if you want to edit the existing definition, you must press [ENTER].
   - To view an existing definition, press [F4] or move the cursor to the header cell and look at the entry line.

2. Type the new expression, which replaces any existing definition.

   If you used [F4] or [ENTER] in Step 1, the cursor moved to the entry line and highlighted the existing definition, if any. You can also:

   - Press [CLEAR] to clear the highlighted expression. Then type the new expression.

     – or –

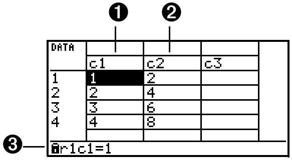
- Press ◀ or ▶ to remove the highlighting. Then edit the old expression.

Note: To cancel any changes, press [ESC] before pressing [ENTER].

You can use an expression that:	For example:
Generates a series of numbers.	c1=seq(x^2,x,1,5) c1={1,2,3,4,5}
Refers to another column.	c2=2∗c1 c4=c1∗c2–sin(c3)

Note: The seq function is described in the Technical Reference module. If you refer to an empty column, you will get an error message unless **Auto-calculate = OFF**.

3. Press [ENTER], ▼, or ▲ to save the definition and update the columns.



❶ c1=seq(x,x,1,7); ❷ c2=2∗c1; ❸ You cannot directly change a locked cell (🔒) since it is defined by the column header.

Note: For a data variable, header definitions are saved when you leave the Data/ Matrix Editor. For a list variable, the definitions are not saved (only their resulting cell values).

Clearing a Header Definition

1. Move the cursor to any cell in the column and press [F4].

   – or –

   Move the cursor to the header cell (**c1**, **c2**, etc.) and press [ENTER].
2. Press [CLEAR] to clear the highlighted expression.
3. Press [ENTER], ⊝, or ⊜.

Using an Existing List as a Column

Suppose you have one or more existing lists, and you want to use those existing lists as columns in a data variable.

From the:	Do this:
Data/Matrix Editor	In the applicable column, use [F4] to define the column header. Refer to the existing list variable. For example: c1=list1
Home screen or a program	Use the **NewData** command as described in the *Technical Reference* module. For example: **NewData** *datavar*, *list1* [, *list2*] [, *list3*] ... ❶ ❷

❶ Data variable. If this data variable already exists, it will be redefined based on the specified lists.
❷ Existing list variables to copy to columns in the data variable.

Note: If you have a CBL 2™ or CBR™, use these techniques for your collected lists. Use [2nd] [VAR-LINK] to see existing list variables.

To Fill a Matrix with a List

You cannot use the Data/Matrix Editor to fill a matrix with a list. However, you can use the **list▶mat** command from the Home screen or a program. For information, refer to the *Technical Reference* module.

The Auto-calculate Feature

For list and data variables, the Data/Matrix Editor has an Auto-calculate feature. By default, Auto-calculate = ON. Therefore, if you make a change that affects a header definition (or any column referenced in a header definition), all header definitions are recalculated automatically. For example:

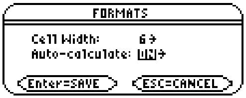
- If you change a header definition, the new definition is applied automatically.
- If column 2's header is defined as c2=2∗c1, any change you make in column 1 is automatically reflected in column 2.

To turn Auto-calculate off and on from the Data/Matrix Editor:

1. Press:
   [F1] **9**

   – or –

2. [♦] [|] Change **Auto-Calculate** to **OFF** or **ON**.



3. Press [ENTER] to close the dialog box.

If **Auto-calculate = OFF** and you make changes as described above, the header definitions are not recalculated until you set **Auto-calculate = ON**.

Note: You may want to set **Auto-calculate = OFF** to make changes without recalculating each time, enter a definition such as c1=c2+c3 before you enter columns 2 and 3, or override any errors in a definition until you can debug the error.

Using Shift and CumSum Functions in a Column Header

When defining a column header, you can use the shift and **cumSum** functions as described below. These descriptions differ slightly from the *Technical Reference* module. This section describes how to use the functions in the Data/Matrix Editor. The *Technical Reference* module gives a more general description for the Home screen or a program.

Using the Shift Function

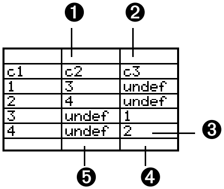
The **shift** function copies a base column and shifts it up or down by a specified number of elements. Use [F4] to define a column header with the syntax:

shift (*column* [,*integer*]**)**

❶ ❷

❶ Column used as the base for the shift.
❷ Number of elements to shift (positive shifts up; negative shifts down).
Default is ⁻1.

For example, for a two-element shift up and down:



❶ c2=shift(c1,2)
❷ c3=shift(c1,⁻2)
❸ Shifted columns have the same length as the base
❹ Last two elements of c1 shift down and out the bottom; undefined elements shift into the top.
❺ First two elements of c1 shift up and out the top; undefined elements shift into the bottom.

Note: To enter shift, type it from the keyboard or select it from the CATALOG.

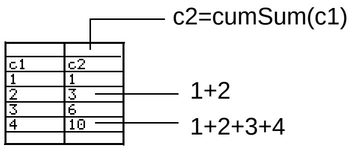
Using the CumSum Function

The **cumSum** function returns a cumulative sum of the elements in a base column. Use [F4] to define a column header with the syntax:

cumSum (*column***)**

Column used as the base for the cumulative sum.

For example:



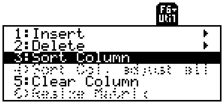
Note: To enter **cumSum**, type it, select it from the CATALOG, or press [2nd] [MATH] and select it from the List submenu.

Sorting Columns

After entering information in a data, list, or matrix variable, you can easily sort a specified column in numeric or alphabetical order. You can also sort all columns as a whole, based on a “key” column.

Sorting a Single Column

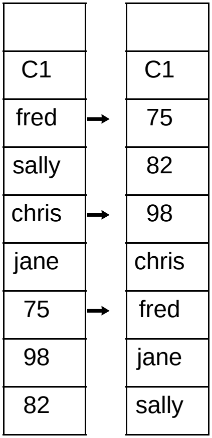
In the Data/Matrix Editor:



1. Move the cursor to any cell in the column.
2. Press:
   [2nd] [F6] and select **3:Sort Column.**

Numbers are sorted in ascending order.

Character strings are sorted in alphabetical order.



Consider a database structure in which each column along the same row contains related information (such as a student’s first name, last name, and test scores). In such a case, sorting only a single column would destroy the relationship between the columns.

In the Data/Matrix Editor:

1. Move the cursor to any cell in the “key” column.
2. In this example, move the cursor to the second column (**c2**) to sort by last name.

   Note: For a **list** variable, this is the same as sorting a single column.

c1	c2	c3
fred	stone	95
sally	ross	75
jane	smith	97
nick	castle	93

3. Press:
   [2nd] [F6] and select **4:Sort Col**, adjust all.

   Note: This menu item is not available if any column is locked.

c1	c2	c3
nick	castle	93
sally	ross	75
jane	smith	97
fred	stone	95

When using this procedure for a data variable:

- All columns must have the same length.
- None of the columns can be locked (defined by a function in the column header). When the cursor is in a locked column, is shown at the beginning of the entry line.

Saving a Copy of a List, Data, or Matrix Variable

You can save a copy of a list, data, or matrix variable. You can also copy a list to a data variable, or you can select a column from a data variable and copy that column to a list.

Valid Copy Types

You can copy a:	To a:
List	List or data
Data	Data
Data column	List
Matrix	Matrix

Note: A list is automatically converted to a data variable if you enter more than one column of information.

Procedure

From the Data/Matrix Editor:

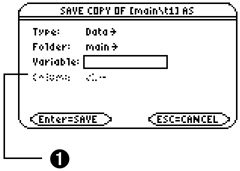
1. Display the variable that you want to copy.

2. Press [F1] and select **2:Save Copy As.**

3. In the dialog box:

   - Select the **Type** and **Folder** for the copy.
   - Type a variable name for the copy.
   - When available, select the column to copy from.

   Note: If you type the name of an existing variable, its contents will be replaced.

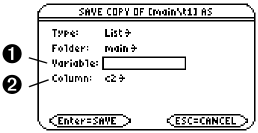


❶ Column is dimmed unless you copy a data column to a list. The column information is not used for other types of copies.

4. Press [ENTER] (after typing in an input box such as Variable, you must press [ENTER] twice).

To Copy a Data Column to a List

A data variable can have multiple columns, but a list variable can have only one column. Therefore, when copying from a data variable to a list, you must select the column that you want to copy.



❶ List variable to copy to.

❷ Data column that will be copied to the list. By default, this shows the column that contains the cursor.

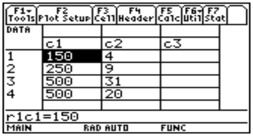
Statistics and Data Plots

Overview of Steps in Statistical Analysis

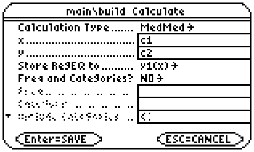
This section gives an overview of the steps used to perform a statistical calculation or graph a statistical plot. For detailed descriptions, refer to the following pages.

1. Set Graph mode ([MODE]) to **FUNCTION**.
2. Enter stat data in the Data/Matrix Editor.

   Note: Refer to the Data/Matrix Editor module for details on entering data in the Data/Matrix Editor.

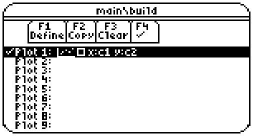


3. Perform stat calculations to find stat variables or fit data to a model ([F5]).



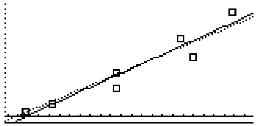
4. Define and select stat plots ([F2] and then [F1]).

   Note: You can also use the Y= Editor to define and select stat plots and **y(x)** functions.



5. Define the viewing window ([♦] [WINDOW]).

6. Change the graph format if necessary.



7. [F1] **9**
   — or —
   [calculator icon] [♦] [I]
   Graph the selected equations
   ([♦] [GRAPH]).

Performing a Statistical Calculation

From the Data/Matrix Editor, use the [F5] **Calc** toolbar menu to perform statistical calculations. You can analyze one-variable or two-variable statistics, or perform several types of regression analyses.

The Calculate Dialog Box

You must have a data variable opened. The Data/Matrix Editor will not perform statistical calculations with a list or matrix variable.

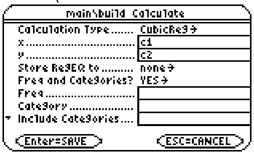
From the Data/Matrix Editor:

1. Press [F5] to display the **Calculate** dialog box.

   This example shows all items as active. On your calculator, items are active only if they are valid for the current settings of **Calculation Type** and **Freq and Categories.**

   Note: If an item is not valid for the current settings, it will appear dimmed. You cannot move the cursor to a dimmed item

Pathname of the data variable



2. Specify applicable settings for the active items.

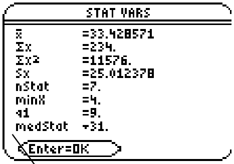
Item	Description
Calculation Type	Select the type of calculation.
x	Type the column number in the Data/Matrix Editor (**C1**, **C2**, etc.) used for x values, the independent variable.
Y	Type the column number used for y values, the dependent variable. This is required for all **Calculation Types** except **OneVar**.
Store RegEQ to	If Calculation Type is a regression analysis, you can select a function name (**y1(x)**, **y2(x)**, etc.). This lets you store the regression equation so that it will be displayed in the Y= Editor.
Use Freq and Categories?	Select **NO** or **YES**. Note that Freq, Category, and Include Categories are active only when Use Freq and Categories? = YES.
Freq	Type the column number that contains a “weight” value for each data point. If you do not enter a column number, all data points are assumed to have the same weight (1).
Category	Type the column number that contains a category value for each data point.
Include Categories	If you specify a Category column, you can use this item to limit the calculation to specified category values. For example, if you specify {1,4}, the calculation uses only data points with a category value of 1 or 4.

Note: To use an existing list variable for x, y, Freq, or Category, type the list name instead of a column number. An example using Freq, Category, and Include Categories is available.

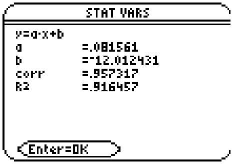
3. Press [ENTER] after typing in an input box, press [ENTER] twice).

   The results are displayed on the **STAT VARS** screen. The format depends on the **Calculation Type**. For example:

For Calculation Type = OneVar	For Calculation Type = LinReg



When ▼ is shown instead of =, you can scroll for additional results.



Note: Any undefined data points (shown as **undef**) are ignored in a stat calculation.

4. To close the **STAT VARS** screen, press [ENTER].

Redisplaying the STAT VARS Screen

The Data/Matrix Editor’s Stat toolbar menu redisplays the previous calculation results (until they are cleared from memory).

[2nd] [F7]

Previous results are cleared when you:

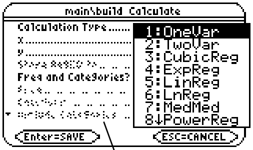
- Edit the data points or change the Calculation Type.
- Open another data variable or reopen the same data variable (if the calculation referred to a column in a data variable). Results are also cleared if you leave and then reopen the Data/Matrix Editor with a data variable.
- Change the current folder (if the calculation referred to a list variable in the previous folder).

Statistical Calculation Types

As described in the previous section, the Calculate dialog box lets you specify the statistical calculation you want to perform. This section gives more information about the calculation types.

From the Calculate dialog box (F5), highlight the current setting for the **Calculation Type** and press ►.

You can then select from a menu of available types.



If an item is dimmed, it is not valid for the current Calculation Type.

Calc Type	Description
OneVar	One-variable statistics — Calculates the statistical variables.
TwoVar	Two-variable statistics — Calculates the statistical variables.
CubicReg	Cubic regression — Fits the data to the third-order polynomial **$y=ax^3+bx^2+cx+d$**. You must have at least four data points. • For four points, the equation is a polynomial fit. • For five or more points, it is a polynomial regression.
ExpReg	Exponential regression — Fits the data to the model equation **$y=ab^x$** (where a is the y-intercept) using a least-squares fit and transformed values **x** and **ln(y)**.

Calc Type	Description
LinReg	Linear regression — Fits the data to the model **y=ax+b** (where a is the slope, and b is the y-intercept) using a least-squares fit and x and y.
LnReg	Logarithmic regression — Fits the data to the model equation **y=a+b ln(x)** using a least-squares fit and transformed values **ln(x)** and **y**.
Logistic	Logistic regression — Fits the data to the model **y=a/(1+b∗e^(c∗x))+d** and updates all the system statistics variables.
MedMed	Median-Median — Fits the data to the model **y=ax+b** (where a is the slope, and b is the y-intercept) using the median-median line, which is part of the resistant line technique. Summary points **medx1**, **medy1**, **medx2**, **medy2**, **medx3**, and **medy3** are calculated and stored to variables, but they are not displayed on the STAT VARS screen.
PowerReg	Power regression — Fits the data to the model equation **y=ax**^{b} using a least-squares fit and transformed values **ln(x)** and **ln(y)**.
QuadReg	Quadratic regression — Fits the data to the second-order polynomial **y=ax**^{2}**+bx+c**. You must have at least three data points. • For three points, the equation is a polynomial fit. • For four or more points, it is a polynomial regression.

Calc Type	Description
QuartReg	Quartic regression — Fits the data to the fourth-order polynomial $\mathbf{y=ax^4+bx^3+cx^2+dx+e}$. You must have at least five data points. • For five points, the equation is a polynomial fit. • For six or more points, it is a polynomial regression.
SinReg	Sinusoidal regression — Calculates the sinusoidal regression and updates all the system statistics variables. The output is always in radians, regardless of the angle mode setting.

Note: For **TwoVar** and all regression calculations, the columns that you specify for x and y (and optionally, Freq or Category) must have the same length.

From the Home Screen or a Program

Use the applicable command for the calculation that you want to perform. The commands have the same name as the corresponding Calculation Type. Refer to the *Technical Reference* module for information about each command.

Important: These commands perform a statistical calculation but do not automatically display the results. Use the **ShowStat** command to show the calculation results.

Statistical Variables

Statistical calculation results are stored to variables. To access these variables, type the variable name or use the VAR-LINK screen as described in *Memory and Variable*

Management. All statistical variables are cleared when you edit the data or change the calculation type. Other conditions that clear the variables are listed.

Calculated Variables

Statistical variables are stored as system variables. However, **regCoef** and **regeq** are treated as a list and a function variable, respectively.

	One Var	Two Var	Regressions
mean of x values	$\bar{x}$	$\bar{x}$	
sum of x values	Σx	Σx	
sum of x^2 values	Σx^2	Σx^2	
sample std. deviation of x	Sx	Sx	
population std. deviation of x	σx	σx	
number of data points	nStat	nStat	
mean of y values		$\bar{y}$	
sum of y values		Σy	
sum of y^2 values		Σy^2	
sample standard deviation of y		Sy	
population std. deviation of y		σy	
sum of x ∗y values		Σxy	
minimum of x values	minX	minX	

	One Var	Two Var	Regressions
maximum of x values	maxX	maxX	
minimum of y values		minY	
maximum of y values		maxY	
1st quartile	q1		
median	medStat		
3rd quartile	q3		
regression equation			regeq
regression coefficients (a, b, c, d, e)			regCoef
correlation coefficient ††			corr
coefficient of determination ††			R^2
summary points (MedMed only) †			medx1, medy1, medx2, medy2, medx3, medy3

†† **corr** is defined for a linear regression only; $\mathbf{R}^2$ is defined for all polynomial regressions.

Note:

- If **regeq** is 4x + 7, then **regCoef** is {4 7}**.** To access the “*a*” coefficient (the 1st element in the list), use an index such as **regCoef[1].**

- 1st quartile is the median of points between **minX** and **medStat**, and 3rd quartile is the median of points between **medStat** and **maxX**.

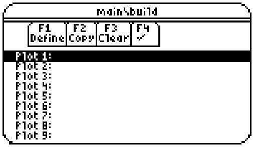
Defining a Statistical Plot

From the Data/Matrix Editor, you can use the entered data to define several types of statistical plots. You can define up to nine plots at a time.

Procedure

From the **Data/Matrix Editor**:

1. Press [F2] to display the **Plot Setup** screen. Initially, none of the plots are defined.
2. Move the cursor to highlight the plot number that you want to define.

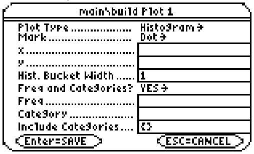


3. Press [F1] to define the plot.

   This example shows all items as active. On your calculator, items are active only if they are valid for the current setting of **Plot Type** and use **Freq and Categories?**.

   Note: If an item is not valid for the current settings, it will appear dimmed. You cannot move the cursor to a dimmed item.

Pathname of the data variable



4. Specify applicable settings for the active items.

Item	Description
Plot Type	Select the type of plot.
Mark	Select the symbol used to plot the data points: Box (☐), Cross (x), Plus (+), Square (⊟), or Dot (•).
x	Type the column number in the Data/Matrix Editor (**C1**, **C2**, etc.) used for x values, the independent variable.
y	Type the column number used for y values, the dependent variable. This is active only for Plot Type = Scatter or xyline.
Hist. Bucket Width	Specifies the width of each bar in a histogram.
Freq and Categories?	Select **NO** or **YES**. Note that Freq, Category, and Include Categories are active only when Freq and Categories? = YES. (Freq is active only for Plot Type = Box Plot or Histogram.)
Freq	Type the column number that contains a “weight” value for each data point. If you do not enter a column number, all data points are assumed to have the same weight (1).
Category	Type the column number that contains a category value for each data point.
Include Categories	If you specify a Category, you can use this to limit the calculation to specified category values. For example, if you specify {1,4}, the plot uses only data points with a category value of 1 or 4.

Note:

- Plots defined with column numbers always use the last data variable in the Data/Matrix Editor, even if that variable was not used to create the definition.

- To use an existing list variable for x, y, Freq, or Category, type the list name instead of the column number.

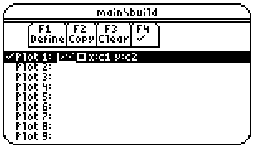
- An example using Freq, Category, and Include Categories is available.

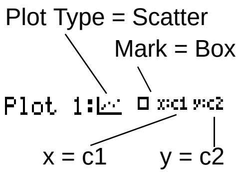
5. Press [ENTER] (after typing in an input box, press [ENTER] twice).

   The **Plot Setup** screen is redisplayed.

   The plot you just defined is automatically selected for graphing.

   Notice the shorthand definition for the plot.





Note: Any undefined data points (shown as **undef**) are ignored in a stat plot.

Selecting or Deselecting a Plot

From Plot Setup, highlight the plot and press [F4] to toggle it on or off. If a stat plot is selected, it remains selected when you:

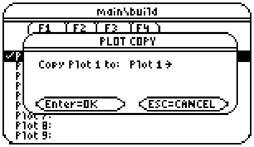
- Change the graph mode. (Stat plots are not graphed in 3D mode.)

- Execute a Graph command.
- Open a different variable in the Data/Matrix Editor.

Copying a Plot Definition

From **Plot Setup**:

1. Highlight the plot and press F2.
2. Press ▶ and select the plot number that you want to copy to.
3. Press ENTER.



Note: If the original plot was selected (✓), the copy is also selected.

Clearing a Plot Definition

From Plot Setup, highlight the plot and press F3. To redefine an existing plot, you do not necessarily need to clear it first; you can make changes to the existing definition. To prevent a plot from graphing, you can deselect it.

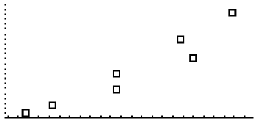
Statistical Plot Types

When you define a plot as described in the previous section, the Plot Setup screen lets you select the plot type. This section gives more information about the available plot types.

Scatter

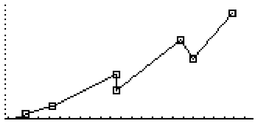
Data points from x and y are plotted as coordinate pairs. Therefore, the columns or lists that you specify for x and y must be the same length.

- Plotted points are shown with the symbol that you select as the Mark.
- If necessary, you can specify the same column or list for both x and y.



Xyline

This is a scatter plot in which data points are plotted and connected in the order in which they appear in x and y.

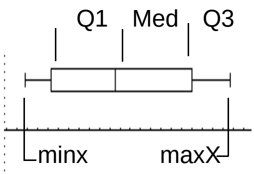


You may want to sort all the columns in the Data/Matrix Editor before plotting.

[2nd] [F6] **3** or [2nd] [F6] **4**

This plots one-variable data with respect to the minimum and maximum data points (**minX** and **maxX**) in the set.

- A box is defined by its first quartile (**Q1**), median (**Med**), and third quartile (**Q3**).
- Whiskers extend from **minX** to **Q1** and from **Q3** to **maxX**.



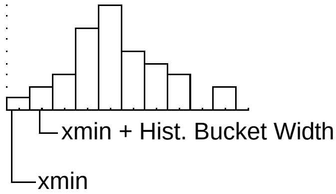
- When you select multiple box plots, they are plotted one above the other in the same order as their plot numbers.
- Use NewPlot to show statistical data as a modified box plot.
- Select Mod Box Plot as the Plot Type when you define a plot in the Data/Matrix Editor.

A modified box plot excludes points outside the interval [**Q1−X, Q3+X**], where X is defined as **1.5 (Q3−Q1)**. These points, called outliers, are plotted individually beyond the box plot's whiskers, using the mark that you select.

This plots one-variable data as a histogram. The x axis is divided into equal widths called buckets or bars. The height of each bar (its y value) indicates how many data points fall within the bar’s range.

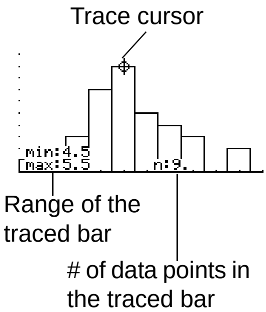
- When defining the plot, you can specify the **Hist. Bucket Width** (default is 1) to set the width of each bar.
- A data point at the edge of a bar is counted in the bar to the right.

$$\text{Number of bars} = \frac{\text{xmax} - \text{xmin}}{\text{Hist. Bucket Width}}$$



- **ZoomData** (F2 **9** from the Graph screen, Y= Editor, or Window Editor) adjusts **xmin** and **xmax** to include all data points, but it does not adjust the y axis.
  - Use ♦ [WINDOW] to set **ymin = 0** and **ymax** = the number of data points expected in the tallest bar.

- When you trace (F3) a histogram, the screen shows information about the traced bar.

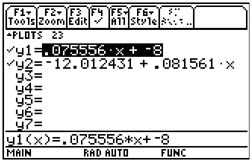


Using the Y= Editor with Stat Plots

The previous sections described how to define and select stat plots from the Data/Matrix Editor. You can also define and select stat plots from the Y= Editor.

Press [♦] [Y=] to display the Y= Editor. Initially, the nine stat plots are located “off the top” of the screen, above the **y(x)** functions. However, the PLOTS indicator provides some information.

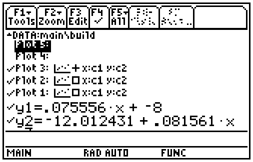
For example, PLOTS 23 means that Plots 2 & 3 are selected.



To see the list of stat plots, use ⊙ to scroll above the **y(x)** functions.

If a Plot is highlighted, this shows the data variable that will be used for the plots.

If a Plot is defined, it shows the same shorthand notation as the Plot Setup screen.



From the Y= Editor, you can perform most of the same operations on a stat plot as you can on any other **y(x)** function.

Note: Plots defined with column numbers always use the last data variable in the Data/Matrix Editor, even if that variable was not used to create the definition.

To:	Do this:
Edit a plot definition	Highlight the plot and press [F3]. You will see the same definition screen that is displayed in the Data/Matrix Editor.
Select or deselect a plot	Highlight the plot and press [F4].
Turn all plots and/or functions off	Press [F5] and select the applicable item. You can also use this menu to turn all functions on.

Note: You can not use [2nd] [F6] to set a plot's display style. However, the plot definition lets you select the mark used for the plot.

To Graph Plots and Y= Functions

As necessary, you can select and graph stat plots and **y(x)** functions at the same time.

Graphing and Tracing a Defined Stat Plot

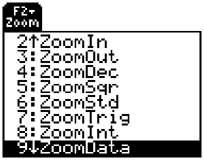
After entering the data points and defining the stat plots, you can graph the selected plots by using the same methods you used to graph a function from the Y= Editor (as described in *Basic Function Graphing*).

Stat plots are displayed on the current graph, and they use the Window variables that are defined in the Window Editor.

Use [♦] [WINDOW] to display the Window Editor. You can either:

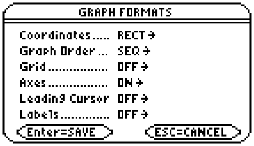
- Enter appropriate values.
  — or —
- Select **9:ZoomData** from the [F2] **Zoom** toolbar menu. (Although you can use any zoom, **ZoomData** is optimized for st plots.)

ZoomData sets the viewing window to display all statistical data points.



For histograms and box plots, only **xmin** and **xmax** are adjusted. If the top of a histogram is not shown, trace the histogram to find the value for **ymax**.

Note: [F2] **Zoom** is available on the Y= Editor, Window Editor, and Graph screen.



Press:
[F1] **9**
— or —
[calculator] [◆] [|]
from the Y= Editor, Window Editor, or Graph screen.

Then change the settings as necessary.

Tracing a Stat Plot

From the Graph screen, press [F3] to trace a plot. The movement of the trace cursor depends on the Plot Type.

Plot Type	Description
Scatter or xyline	Tracing begins at the first data point.
Box plot	Tracing begins at the median. Press ◀ to trace to **Q1** and **minX**. Press ▶ to trace to **Q3** and **maxX**.
Histogram	The cursor moves from the top center of each bar, starting from the leftmost bar.

Note: When a stat plot is displayed, the Graph screen does not automatically pan if you trace off the left or right side of the screen. However, you can still press [ENTER] to center the screen on the trace cursor.

When you press ⊖ or ⊖ to move to another plot or **y(x)** function, tracing moves to the current or beginning point on that plot (not to the nearest pixel).

Using Frequencies and Categories

To manipulate the way in which data points are analyzed, you can use frequency values and/or category values. Frequency values let you “weight” particular data points. Category values let you analyze a subset of the data points.

Example of a Frequency Column

In a data variable, you can use any column in the Data/Matrix Editor to specify a frequency value (or weight) for the data points on each row. A frequency value must be an integer ≥ 0 if Calculation Type = OneVar or MedMed or if Plot Type = Box Plot. For other statistical calculations or plots, the frequency value can be any number ≥ 0.

For example, suppose you enter a student’s test scores, where:

- The mid-semester exam is weighted twice as much as other tests.
- The final exam is weighted three times as much.

- In the Data/Matrix Editor, you can enter the test scores and frequency values in two columns.

Test scores	Frequency values
c1	c2
85	1
97	1
92	2
89	1
91	1
95	3

These weighted scores are equivalent to the single column of scores listed to the right.

c1
85
97
92 ❶
92 ❶
89
91
95 ❷
95 ❷
95 ❷

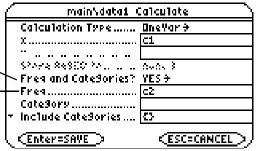
❶ Frequency of 2
❷ Frequency of 3

Note: A frequency value of 0 effectively removes the data point from analysis.

To use frequency values, specify the frequency column when you perform a statistical calculation or define a stat plot. For example:

Set this to YES.

Type the column number (or list name) that contains the frequency values.



Note: You can also use frequency values from a list variable instead of a column.

Example of a Category Column

In a data variable, you can use any column to specify a category (or subset) value for the data points on each row. A category value can be any number.

Suppose you enter the test scores from a class that has 10th and 11th grade students. You want to analyze the scores for the whole class, but you also want to analyze categories such as 10th grade girls, 10th grade boys, 10th grade girls and boys, etc.

First, determine the category values you want to use.

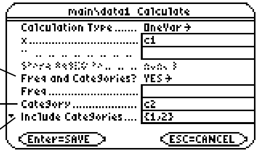
Category Value	Used to indicate:
1	10th grade girl
2	10th grade boy
3	11th grade girl
4	11th grade boy

Note: You do not need a category value for the whole class. Also, you do not need category values for all 10th graders or all 11th graders since they are combinations of other categories.

In the Data/Matrix Editor, you can enter the scores and the category values in two columns.

Test scores	Category values
c1	c2
85	1
97	3
92	2
88	3
90	2
95	1
79	4
68	2
92	4
84	3
82	1

To use category values, specify the category column and the category values to include in the analysis when you perform a statistical calculation or define a stat plot.



Note: You can also use category values from a list variable instead of a column.

To analyze:	Include Categories:
10th grade girls	{1}
10th grade boys	{2}
10th grade girls and boys	{1,2}
11th grade girls	{3}
11th grade boys	{4}
11th grade girls and boys	{3,4}
all girls (10th and 11th)	{1,3}
all boys (10th and 11th)	{2,4}

Note: To analyze the whole class, leave the Category input box blank. Any category values are ignored.

If You Have a CBL 2™ or CBR™

The Calculator-Based Laboratory™ System (CBL 2) and Calculator-Based Ranger™ System (CBR) are optional accessories, available separately, that let you collect data from a variety of real-world experiments. TI-89 Titanium, CBL 2 and CBR programs are available from the TI web site at education.ti.com.

How CBL 2™ Data Is Stored

When you collect data with the CBL 2, that data is initially stored in the CBL 2 unit itself. You must then retrieve the data (transfer it to the TI-89 Titanium) by using the **Get** command, which is described in the *Technical Reference* module.

Although each set of retrieved data can be stored in several variable types (list, real, matrix, pic), using list variables makes it easier to perform statistical calculations.

When you transfer the collected information to the TI-89 Titanium, you can specify the list variable names that you want to use.

For example, you can use the CBL 2 to collect temperature data over a period of time. When you transfer the data, suppose you store:

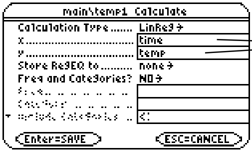
- Temperature data in a list variable called *temp*.
- Time data in a list variable called *time*.

After you store the CBL 2 information on the TI-89 Titanium, there are two ways to use the CBL 2 list variables.

Note: For specifics about using the CBL 2 and retrieving data to the TI-89 Titanium, refer to the guidebook that comes with the CBL 2 unit.

Referring to the CBL 2™ Lists

When you perform a statistical calculation or define a plot, you can refer explicitly to the CBL 2 list variables. For example:



Type the CBL 2 list variable name instead of a column number

Creating a Data Variable with the CBL 2™ Lists

You can create a new data variable that consists of the necessary CBL 2 list variables.

- From the Home screen or a program, use the **NewData** command.

  NewData *dataVar*, *list1* [,*list2*] [,*list3*] ...

  list1 [,*list2*] [,*list3*] ...: CBL 2 list variable names. In the new data variable, list1 will be copied to column 1, list 2 to column 2, etc.

  dataVar: Name of the new data variable that you want to create.

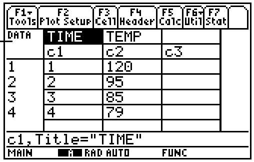
  For example:

  NewData *temp1*, *time*, *temp*

  creates a data variable called *temp1* in which time is in column 1 and *temp* is in column 2.

- From the Data/Matrix Editor, create a new, empty data variable with the applicable name. For each CBL 2 list that you want to include, define a column header as that list name.

  For example, define column 1 as time, column 2 as temp.

  Note: To define or clear a column header, use [F4]. For more information, refer to the *Data/Matrix Editor* module.

  At this point, the columns are linked to the CBL 2 lists. If the lists are changed, the columns will be updated automatically. However, if the lists are deleted, the data will be lost.

  To make the data variable independent of the CBL 2 lists, clear the column header for each column. The information remains in the column, but the column is no longer linked to the CBL 2 list.

CBR™

You can also use the Calculator-Based Ranger™ (CBR) to explore the mathematical and scientific relationships between distance, velocity, acceleration, and time using data collected from activities you perform.

Programming

Running an Existing Program

After a program is created (as described in the remaining sections of this module), you can run it from the Home screen. The program's output, if any, is displayed on the Program I/O screen, in a dialog box, or on the Graph screen.

Running a Program

On the Home screen:

1. Type the name of the program.

2. You must always type a set of parentheses after the name.

   Some programs require you to pass an argument to the program.

   Note: Use [2nd] [VAR-LINK] to list existing **PRGM** variables. Highlight a variable and press [ENTER] to paste its name to the entry line.

   prog1()
   └ If arguments are not required

   prog1(x,y)
   └ If arguments are required

3. Press [ENTER].

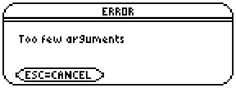
Note: Arguments specify initial values for a program.

When you run a program, the TI-89 Titanium automatically checks for errors. For example, the following message is displayed if you:

- Do not enter ( ) after the program name.

This error message appears if you:

- Do not enter enough arguments, if required.



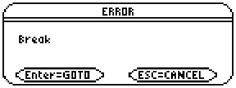
To cancel program execution if an error occurs, press ESC. You can then correct any problems and run the program again.

Note: The TI-89 Titanium also checks for run-time errors that are found within the program itself.

“Breaking” a Program

When a program is running, the BUSY indicator is displayed in the status line.

Press [ON] to stop program execution. A message is then displayed.



- To display the program in the Program Editor, press [ENTER]. The cursor appears at the command where the break occurred.
- To cancel program execution, press [ESC].

Where Is the Output Displayed?

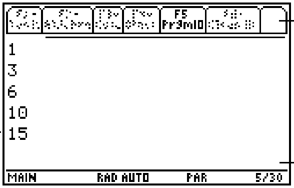
Depending on the commands in the program, the TI-89 Titanium automatically displays information on the applicable screen.

- Most output and input commands use the Program I/O screen. (Input commands prompt the user to enter information.)
- Graph-related commands typically use the Graph screen.

After the program stops, the TI-89 Titanium shows the last screen that was displayed.

The Program I/O Screen

On the Program I/O screen, new output is displayed below any previous output (which may have been displayed earlier in the same program or a different program). After a full page of output, the previous output scrolls off the top of the screen.



Last output

On the Program I/O screen: [F5] toolbar is available; all others are dimmed. There is no entry line.

Note: To clear any previous output, enter the **ClrIO** command in your program. You can also execute **ClrIO** from the Home screen.

When a program stops on the Program I/O screen, you need to recognize that it is not the Home screen (although the two screens are similar). The Program I/O screen is used only to display output or to prompt the user for input. You cannot perform calculations on this screen.

Note: If Home screen calculations don't work after you run a program, you may be on the Program I/O screen.

Leaving the Program I/O Screen

From the Program I/O screen:

- Press [F5] to toggle between the Home screen and the Program I/O screen.

  – or –

- Press [ESC], [2nd] [QUIT], or
  [HOME]
  [CALC HOME]to display the Home screen.

  – or –

- Display any other application screen (with [APPS], [♦] [Y=], etc.).

Starting a Program Editor Session

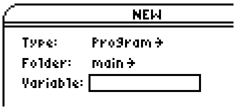
Each time you start the Program Editor, you can resume the current program or function (that was displayed the last time you used the Program Editor), open an existing program or function, or start a new program or function.

Starting a New Program or Function

1. Press [APPS] and then select **Program Editor**.

2. Select **3:New.**

3. Specify the applicable information for the new program or function.

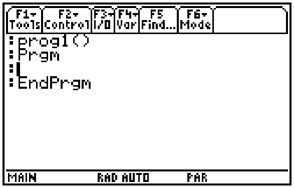


Item	Lets you:
Type	Select whether to create a new program or function. 
Folder	Select the folder in which the new program or function will be stored. For information about folders, refer to the *Calculator Home Screen* module.

Item	Lets you:
Variable	Type a variable name for the program or function. If you specify a variable that already exists, an error message will be displayed when you press [ENTER]. When you press [ESC] or [ENTER] to acknowledge the error, the **NEW** dialog box is redisplayed.

4. Press [ENTER] (after typing in an input box such as **Variable**, you must press [ENTER] twice) to display an empty “template.”

This is the template for a program. Functions have a similar template.



You can now use the Program Editor as described in the remaining sections of this module.

Note: A program (or function) is saved automatically as you type. You do not need to save it manually before leaving the Program Editor, starting a new program, or opening a previous one.

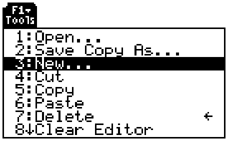
Resuming the Current Program

You can leave the Program Editor and go to another application at any time. To return to the program or function that was displayed when you left the Program Editor, launch Program Editor again and select **1:Current**.

Starting a New Program from the Program Editor

To leave the current program or function and start a new one:

1. Press [F1] and select **3:New**.
2. Specify the type, folder, and variable for the new program or function.
3. Press [ENTER] twice.



Opening a Previous Program

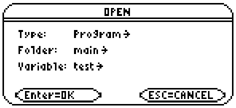
You can open a previously created program or function at any time.

1. From within the **Program Editor**, press [F1] and select **1:Open**.

   – or –

   From another application, launch Program Editor again and select **2:Open**.

2. Select the applicable type, folder, and variable.
3. Press [ENTER].



Note: By default, Variable shows the first existing program or function in alphabetical order.

Copying a Program

In some cases, you may want to copy a program or function so that you can edit the copy while retaining the original.

1. Display the program or function you want to copy.
2. Press [F1] and select **2:Save Copy As**.
3. Specify the folder and variable for the copy.
4. Press [ENTER] twice.

Note about Deleting a Program

Because all Program Editor sessions are saved automatically, you can accumulate quite a few previous programs and functions, which take up memory storage space.

To delete programs and functions, use the VAR-LINK screen ([2nd] [VAR-LINK]). For information about VAR-LINK, refer to the *Memory and Variable Management* module.

Overview of Entering a Program

A program is a series of commands executed in sequential order (although some commands alter the program flow). In general, anything that can be executed from the Home screen can be included in a program. Program execution continues until it reaches the end of the program or a **Stop** command.

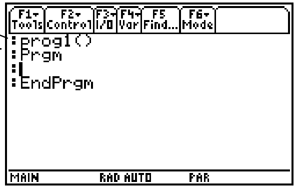
Entering and Editing Program Lines

On a blank template, you can begin entering commands for your new program.

Program name, which you specify when you create a new program.

Enter your program commands between **Prgm** and **EndPrgm**.

All program lines begin with a colon.



Note: Use the cursor pad to scroll through the program for entering or editing commands. Use [♦] ▲ or [♦] ▼ to go to the top or bottom of a program, respectively.

You enter and edit program commands in the Program Editor by using the same techniques used to enter and edit text in the Text Editor. Refer to “Entering and Editing Text” in the *Text Editor* module.

After typing each program line, press [ENTER]. This inserts a new blank line and lets you continue entering another line. A program line can be longer than one line on the screen; if so, it will wrap to the next screen line automatically.

Note: Entering a command does not execute that command. It is not executed until you run the program.

Entering Multi- Command Lines

To enter more than one command on the same line, separate them with a colon by pressing [2nd] [:].

Entering Comments

A comment symbol (©) lets you enter a remark in a program. When you run the program, all characters to the right of © are ignored.

```
  :prog1()
  :Prgm
❶ :©Displays sum of 1 thru n
  :Request "Enter an integer",n
❷ :expr(n)→n:©Convert to numeric expression
  :------
```

❶ Description of the program .
❷ Description of **expr**.

Note: Use comments to enter information that is useful to someone reading the program code.

To enter the comment symbol, press:

- [♦] [)]

  – or –

- Press [F2] and select **9**:©

Controlling the Flow of a Program

When you run a program, the program lines are executed in sequential order. However, some commands alter the program flow. For example:

- Control structures such as **If...EndIf** commands use a conditional test to decide which part of a program to execute.
- Loops commands such as **For...EndFor** repeat a group of commands.

Using Indentation

For more complex programs that use **If...EndIf** and loop structures such as **For...EndFor**, you can make the programs easier to read and understand by using indentation.

```
:If x>5 Then
:  Disp "x is > 5"
:Else
:  Disp "x is < or = 5"
:EndIf
```

Displaying Calculated Results

In a program, calculated results are not displayed unless you use an output command. This is an important difference between performing a calculation on the Home screen and in a program.

These calculations will not display a result in a program (although they will on the Home screen).

```
:12*6
:cos(π/4)
:solve(x^2-x-2=0,x)
```

Output commands such as **Disp** will display a result in a program.

```
:Disp 12*6
:Disp cos(π/4)
:Disp solve(x^2-x-2=0,x)
```

Displaying a calculation result does not store that result. If you need to refer to a result later, store it to a variable.

```
:cos(π/4)→maximum
:Disp maximum
```

Note: A list of output commands is available.

Getting Values into a Program

To input values into a program, you can:

- Require the users to store a value (with [STO▸]) to the necessary variables before running the program. The program can then refer to these variables.

- Enter the values directly into the program itself.

```
:Disp 12*6
:cos(π/4)→maximum
```

- Include input commands that prompt the users to enter the necessary values when they run the program.

```
:Input "Enter a value",i
:Request "Enter an
integer",n
```

- Require the users to pass one or more values to the program when they run it.

```
prog1(3,5)
```

Note: A list of input commands is available.

The following program draws a circle on the Graph screen and then draws a horizontal line across the top of the circle. Three values must be passed to the program: x and y coordinates for the circle's center and the radius r.

- When you write the program in the Program Editor:

In the ( ) beside the program name, specify the variables that will be used to store the passed values.

Notice that the program also contains commands that set up the Graph screen.

```
:circ(x,y,r) ❶
:Prgm
:FnOff
:ZoomStd
:ZoomSqr
:Circle x,y,r
:LineHorz y+r
:EndPrgm
```

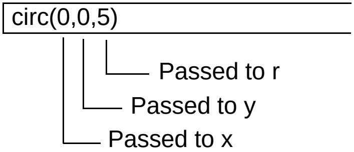
❶ Only **circ( )** is initially displayed on the blank template; be sure to edit this line.

Note: In this example, you cannot use circle as the program name because it conflicts with a command name.

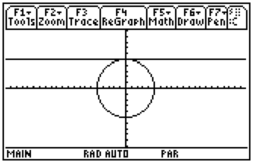
Before drawing the circle, the program turns off any selected Y= Editor functions, displays a standard viewing window, and "squares" the window.

- To run the program from the Home screen:

  The user must specify the applicable values as arguments within the ( ).



  The arguments, in order, are passed to the program.



Note: This example assumes that the user enters values that can be displayed by the viewing window set up by **ZoomStd** and **ZoomSqr**.

Overview of Entering a Function

A function created in the Program Editor is very similar to the functions and instructions that you typically use from the Home screen.

Why Create a User-Defined Function?

Functions (as well as programs) are ideal for repetitive calculations or tasks. You only need to write the function once. Then you can reuse it as many times as necessary. Functions, however, have some advantages over programs.

- You can create functions that expand on the TI-89 Titanium's built-in functions. You can then use the new functions the same as any other function.
- Functions return values that can be graphed or entered in a table; programs cannot.
- You can use a function (but not a program) within an expression. For example: **3∗func1(3)** is valid, but not **3∗prog1(3)**.
- Because you pass arguments to a function, you can write generic functions that are not tied to specific variable names.

Note: You can create a function from the Home screen, but the Program Editor is more convenient for complex, multi-line functions.

Differences Between Functions and Programs

This guidebook sometimes uses the word command as a generic reference to instructions and functions. When writing a function, however, you must differentiate between instructions and functions.

A user-defined function:

- Can use the following instructions only. Any others are invalid.

Cycle	**Define**	**Exit**
For...EndFor	**Goto**	**If...EndIf** (all forms)
Lbl	**Local**	**Loop...EndLoop**
Return	**While...EndWhile**	**→** ([STO▸] key)

- Can use all built-in TI-89 Titanium / Voyage™ 200 functions except:

setFold	**setGraph**	**setMode**
setTable	**switch**	

- Can refer to any variable; however, it can store a value to a local variable only.
  - The arguments used to pass values to a function are treated as local variables automatically. If you store to any other variables, you must declare them as local from within the function.
- Cannot call a program as a subroutine, but it can call another user-defined function.
- Cannot define a program.
- Cannot define a global function, but it can define a local function.

Note: Information about local variables is available.

Entering a Function

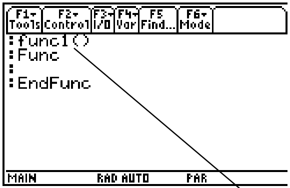
When you create a new function in the Program Editor, the TI-89 Titanium displays a blank “template.”

Note: Use the cursor pad to scroll through the function for entering or editing commands.

Function name, which you specify when you create a new function.

Enter your commands between **Func** and **EndFunc**.

All function lines begin with a colon.



Be sure to edit this line to include any necessary arguments. Remember to use argument names in the definition that will never be used when calling the function.

If the function requires input, one or more values must be passed to the function. (A user-defined function can store to local variables only, and it cannot use instructions that prompt the user for input.)

How to Return a Value from a Function

There are two ways to return a value from a function:

- As the last line in the function (before **EndFunc**), calculate the value to be returned.

```
:cube(x)
:Func
:x^3
:EndFunc
```

- Use **Return.** This is useful for exiting a function and returning a value at some point other than the end of the function.

```
:cube(x)
:Func
:If x<0
:  Return 0
:x^3
:EndFunc
```

Note: This example calculates the cube if **x**≥**0**; otherwise, it returns a `0`.

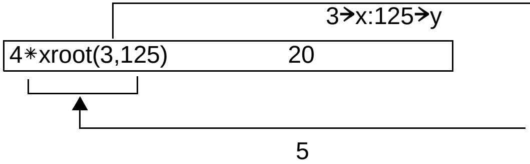
The argument x is automatically treated as a local variable. However, if the example needed another variable, the function would need to declare it as local by using the Local command.

There is an implied **Return** at the end of the function. If the last line is not an expression, an error occurs.

Example of a Function

The following function returns the xth root of a value y ($\sqrt[x]{y}$)**.** Two values must be passed to the function: x and y.

Note: Because x and y in the function are local, they are not affected by any existing x or y variable.

Function as called from the Home Screen	Function as defined in the Program Editor
	:xroot(x,y) :Func :y^(1/x) :EndFunc

Calling One Program from Another

One program can call another program as a subroutine. The subroutine can be external (a separate program) or internal (included in the main program). Subroutines are useful when a program needs to repeat the same group of commands at several different places.

Calling a Separate Program

To call a separate program, use the same syntax used to run the program from the Home screen.

```
:subtest1()
:Prgm
:For i,1,4,1
:  subtest2(i,i*1000)
:EndFor
:EndPrgm
```

```
:subtest2(x,y)
:Prgm
:  Disp x,y
:EndPrgm
```

Calling an Internal Subroutine

To define an internal subroutine, use the **Define** command with **Prgm...EndPrgm**. Because a subroutine must be defined before it can be called, it is a good practice to define subroutines at the beginning of the main program.

An internal subroutine is called and executed in the same way as a separate program.

```
  :subtest1()
  :Prgm
❶ :local subtest2
❷ :Define subtest2(x,y)=Prgm
⋮ :  Disp x,y
❷ :EndPrgm
  :©Beginning of main program
  :For i,1,4,1
❸ :  subtest2(i,I*1000)
  :EndFor
  :EndPrgm
```

❶ Declares the subroutine as a local variable.
❷ Defines the subroutine.
❸ Calls the subroutine.

Note: Use the Program Editor's F4 **Var** toolbar menu to enter the Define and **Prgm...EndPrgm** commands.

Notes about Using Subroutines

At the end of a subroutine, execution returns to the calling program. To exit a subroutine at any other time, use the **Return** command.

A subroutine cannot access local variables declared in the calling program. Likewise, the calling program cannot access local variables declared in a subroutine.

Lbl commands are local to the programs in which they are located. Therefore, a **Goto** command in the calling program cannot branch to a label in a subroutine or vice versa.

Using Variables in a Program

Programs use variables in the same general way that you use them from the Home screen. However, the “scope” of the variables affects how they are stored and accessed.

Scope of Variables

Scope	Description
System (Global) Variables	Variables with reserved names that are created automatically to store data about the state of the calculator. For example, Window variables (**xmin**, **xmax**, **ymin**, **ymax**, etc.) are globally available from any folder. • You can always refer to these variables by using the variable name only, regardless of the current folder. • A program cannot create system variables, but it can use the values and (in most cases) store new values.

Scope	Description
Folder Variables	Variables that are stored in a particular folder. • If you store to a variable name only, it is stored in the current folder. For example: 5→start • If you refer to a variable name only, that variable must be in the current folder. Otherwise, it cannot be found (even if the variable exists in a different folder). • To store or refer to a variable in another folder, you must specify a path name. For example: 5→class\start (class = Variable name; start = Folder Name) After the program stops, any folder variables created by the program still exist and still take up memory.
Local Variables	Temporary variables that exist only while a program is running. When the program stops, local variables are deleted automatically. • To create a local variable in a program, use the **Local** command to declare the variable. • A local variable is treated as unique even if there is an existing folder variable with the same name. • Local variables are ideal for temporarily storing values that you do not want to save.

Note: If a program has local variables, a graphed function cannot access them. For example:

```
Local a
5→a
Graph a*cos(x)
```

may display an error or an unexpected result (if **a** is an existing variable in the current folder).

Circular Definition Errors

When evaluating a user-defined function or running a program, you can specify an argument that includes the same variable that was used to define the function or create the program. However, to avoid Circular definition errors, you must assign a value for x or i variables that are used in evaluating the function or running the program. For example:

```
❶ x+1→x

  – or –

  For i,i,10,1
❶ Disp i
  EndFor
```

❶ Causes a `Circular definition` error message if x or i does not have a value. The error does not occur if x or i has already been assigned a value.

Command	Description
[STO▸] key	Stores a value to a variable. As on the Home screen, pressing [STO▸] enters a → symbol.
Archive	Moves specified variables from RAM to user data archive memory.
BldData	Lets you create a data variable based on the graph information entered in the Y= Editor, Window Editor, etc.
CopyVar	Copies the contents of a variable.
Define	Defines a program (subroutine) or function variable within a program.
DelFold	Deletes a folder. All variables in that folder must be deleted first.
DelType	Deletes unarchived variables of the specified type in all folders.
DelVar	Deletes a variable.
getFold	Returns the name of the current folder.
getType	Returns a string that indicates the data type (EXPR, LIST, etc.) of a variable.
isArchiv()	Indicates if the variable is archived or not.
isLocked()	Indicates if the variable is locked or not.
isVar()	Indicates if the variable is in the symbol table or not.
Local	Declares one or more variables as local variables.

Command	Description
Lock	Locks a variable so that it cannot be accidentally changed or deleted without first being unlocked.
MoveVar	Moves a variable from one folder to another.
NewData	Creates a data variable whose columns consist of a series of specified lists.
NewFold	Creates a new folder.
NewPic	Creates a picture variable based on a matrix.
Rename	Renames a variable.
Unarchiv	Moves specified variables from user data archive memory to RAM.
Unlock	Unlocks a locked variable.

Note: The **Define**, **DelVar**, and **Local** commands are available from the Program Editor's [F4] **Var** toolbar menu.

Using Local Variables in Functions or Programs

A local variable is a temporary variable that exists only while a user-defined function is being evaluated or a user-defined program is running.

Example of a Local Variable

The following program segment shows a **For...EndFor loop** (which is discussed later in this module). The variable *i* is the loop counter. In most cases, the variable *i* is used only while the program is running.

```
❶ :Local I
  :For i,0,5,1
  :  Disp I
  :EndFor
  :Disp i
```

❶ Declares variable *i* as local.

Note: As often as possible, use local variables for any variable that is used only within a program and does not need to be stored after the program stops.

If you declare variable *i* as local, it is deleted automatically when the program stops so that it does not use up memory.

What Causes an Undefined Variable Error Message?

An **Undefined** variable error message displays when you evaluate a user-defined function or run a user-defined program that references a local variable that is not initialized (assigned a value).

This example is a multi-statement function, rather than a program. Line breaks are shown here, but you would type the text in the entry line as one continuous line, such as: **Define fact(n)=Func:Local…** where the ellipsis indicates the entry line text continues off-screen.

For example:

```
  Define fact(n)=Func:
❶ Local m:
  While n>1:
    n∗m→m: n–1→n:
  EndWhile:
  Return m:
  EndFunc
```

❶ Local variable *m* is not assigned an initial value.

In the example above, the local variable *m* exists independently of any variable *m* that exists outside of the function.

You Must Initialize Local Variables

All local variables must be assigned an initial value before they are referenced.

```
  Define fact(n)=Func:
❶ Local m: 1→m:
  While n>1:
    n∗m→m: n–1→n:
  EndWhile:
  Return m:
  EndFunc
```

❶ 1 is stored as the initial value for *m*.

The calculator cannot use a local variable to perform symbolic calculations.

To Perform Symbolic Calculations

If you want a function or program to perform symbolic calculations, you must use a global variable instead of a local. However, you must be certain that the global variable does not already exist outside of the program. The following methods can help.

- Refer to a global variable name, typically with two or more characters, that is not likely to exist outside of the function or program.
- Include **DelVar** within the function or program to delete the global variable, if it exists, before referring to it. (**DelVar** does not delete locked or archived variables.)

String Operations

Strings are used to enter and display text characters. You can type a string directly, or you can store a string to a variable.

A string is a sequence of characters enclosed in "quotes." In programming, strings allow the program to display information or prompt the user to perform some action. For example:

```
Disp "The result is",answer
```

– or –

```
Input "Enter the angle in degrees",ang1
```

– or –

```
"Enter the angle in degrees”→str1
Input str1,ang1
```

Some input commands (such as **InputStr**) automatically store user input as a string and do not require the user to enter quotation marks.

A string cannot be evaluated mathematically, even if it appears to be a numeric expression. For example, the string "`61`" represents the characters "`6`" and "`1`", not the number `61`.

Although you cannot use a string such as "`61`" or "`2x+4`" in a calculation, you can convert a string into a numeric expression by using the **expr** command.

Note: See the *Technical Reference* module for syntax for all commands and functions.

Command	Description
#	Converts a string into a variable name. This is called indirection.
&	Appends (concatenates) two strings into one string.
char	Returns the character that corresponds to a specified character code. This is the opposite of the **ord** command.
dim	Returns the number of characters in a string.
expr	Converts a string into an expression and executes that expression. This is the opposite of the **string** command. **Important:** Some user input commands store the entered value as a string. Before you can perform a mathematical operation on that value, you must convert it to a numeric expression.
format	Returns an expression as a character string based on the format template (fixed, scientific, engineering, etc.)
inString	Searches a string to see if it contains a specified substring. If so, **inString** returns the character position where the first occurrence of the substring begins.
left	Returns a specified number of characters from the left side (beginning) of a string.
mid	Returns a specified number of characters from any position within a string.
ord	Returns the character code of the first character within a string. This is the opposite of the **char** command.

Command	Description
right	Returns a specified number of characters from the right side (end) of a string.
rotate	Rotates the characters in a string. The default is ⁻1 (rotate right one character).
shift	Shifts the characters in a string and replaces them with spaces. The default is ⁻1 (shift right one character and replace with one space). Examples: shift("abcde",2)⇒"cde " and shift("abcde")⇒" abcd"
string	Converts a numeric expression into a string. This is the opposite of the **expr** command.

Conditional Tests

Conditional tests let programs make decisions. For example, depending on whether a test is true or false, a program can decide which of two actions to perform. Conditional tests are used with control structures such as **If...EndIf** and loops such as **While...EndWhile** (described later in this module).

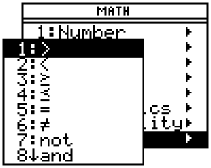
Entering a Test Operator

- Type the operator directly from the keyboard.
  – or –

- Press [2nd] [MATH] and select **8:Test**. Then select the operator from the menu.

  – or –

- Display the built-in functions. Press: [CATALOG] The test operators are listed near the bottom of the [F2] Built-in menu.



Relational Tests

Relational operators let you define a conditional test that compares two values. The values can be numbers, expressions, lists, or matrices (but they must match in type and dimension).

Operator	True if:	Example
>	Greater than	a>8
<	Less than	a<0
≥	Greater than or equal to	a+b≥100
≤	Less than or equal to	a+6≤b+1
=	Equal	list1=list2
≠	Not equal to	mat1≠mat2

Note: From the keyboard, you can type:
>= for ≥
<= for ≤
/= for ≠
(To get the / character, press [÷].)

Boolean Tests

Boolean operators let you combine the results of two separate tests.

Operator	True if:	Example
and	Both tests are true	a>0 and a≤10
or	At least one test is true	a≤0 or b+c>10
xor	One test is true and the other is false	a+6<b+1 xor c<d

The Not Function

The **not** function changes the result of a test from true to false and vice versa. For example:

not x>2	is true if	x≤2
	is false if	x>2

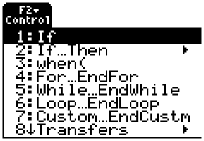
Note: If you use **not** from the Home screen, it is shown as **~** in the history area. For example, **not x>2** is shown as **~(x>2).**

Using If, Lbl, and Goto to Control Program Flow

An **If...EndIf** structure uses a conditional test to decide whether or not to execute one or more commands. **Lbl** (label) and **Goto** commands can also be used to branch (or jump) from one place to another in a program.

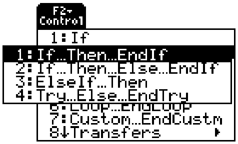
F2 Control Toolbar Menu

To enter **If...EndIf** structures, use the Program Editor’s [F2] **Control** toolbar menu.



The If command is available directly from the [F2] menu.

To see a submenu that lists other **If** structures, select **2:If...Then.**



When you select a structure such as **If...Then...EndIf**, a template is inserted at the cursor location.

```
:If | Then ❶
:EndIf
```

❶ The cursor is positioned so that you can enter a conditional test.

If Command

To execute only one command if a conditional test is true, use the general form:

```
  :If x>5
❶ :   Disp "x is greater than 5"
❷ :Disp x
```

❶ Executed only if x>5; otherwise, skipped.
❷ Always displays the value of x.

In this example, you must store a value to x before executing the **If** command.

Note: Use indentation to make your programs easier to read and understand.

If...Then...EndIf Structures

To execute one group of commands if a conditional test is true, use the structure:

```
  :If x>5 Then
❶ :  Disp "x is greater than 5"
❶ :  2∗x→x
❷ :EndIf
  :Disp x
```

❶ Executed only if x>5.
❷ Displays value of:

- 2x if x>5
- x if x≤5

Note: EndIf marks the end of the **Then** block that is executed if the condition is true.

If...Then...Else... EndIf Structures

To execute one group of commands if a conditional test is true and a different group if the condition is false, use this structure:

```
  :If x>5 Then
❶ :  Disp "x is greater than 5"
❶ :  2∗x→x
  :Else
❷ :  Disp "x is less than or
❷    equal to 5"
  :  5∗x→x
  :EndIf
❸ :Disp x
```

❶ Executed only if x>5.
❷ Executed only if x≤5.
❸ Displays value of:
- 2x if x>5
- 5x if x≤5

If...Then...ElseIf... EndIf Structures

A more complex form of the If command lets you test a series of conditions. Suppose your program prompts the user for a number that corresponds to one of four options. To test for each option (**If Choice=1**, **If Choice = 2**, etc.), use the **If...Then...ElseIf...EndIf** structure.

Refer to the *Technical Reference* module for more information and an example.

Lbl and Goto Commands

You can also control the flow of your program by using **Lbl** (label) and **Goto** commands.

Use the **Lbl** command to label (assign a name to) a particular location in the program.

Lbl *labelName*

└── name to assign to this location (use the same naming convention as a variable name)

You can then use the **Goto** command at any point in the program to branch to the location that corresponds to the specified label.

Goto *labelName*

└── specifies which **Lbl** command to branch to

Because a **Goto** command is unconditional (it always branches to the specified label), it is often used with an **If** command so that you can specify a conditional test. For example:

```
  :If x>5
❶ :  Goto GT5
❷ :Disp x
  :--------
  :--------
  :Lbl GT5
  :Disp "The number was > 5”
```

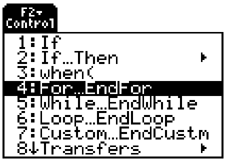
❶ If x>5, branches directly to label GT5.
❷ For this example, the program must include commands (such as **Stop**) that prevent **Lbl** GT5 from being executed if x≤5.

Using Loops to Repeat a Group of Commands

To repeat the same group of commands successively, use a loop. Several types of loops are available. Each type gives you a different way to exit the loop, based on a conditional test.

F2 Control Toolbar Menu

To enter most of the loop-related commands, use the Program Editor’s [F2] **Control** toolbar menu.



When you select a loop, the loop command and its corresponding **End** command are inserted at the cursor location.

```
:For | ❶
:EndFor
```

❶ If the loop requires arguments, the cursor is positioned after the command.

You can then begin entering the commands that will be executed in the loop.

Note: A loop command marks the start of the loop. The corresponding **End** command marks the end of the loop.

For...EndFor Loops

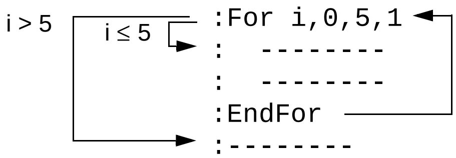
A **For...EndFor loop** uses a counter to control the number of times the loop is repeated. The syntax of the **For** command is:

Note: The ending value can be less than the beginning value, but the increment must be negative.

For(*variable*, *begin*, *end* [, *increment*]**)**
❶ ❷ ❸ ❹

- ❶ variable used as a counter
- ❷ counter value used the first time **For** is executed
- ❸ exits the loop when variable exceeds this value
- ❹ added to the counter each subsequent time **For** is executed

When **For** is executed, the variable value is compared to the end value. If variable does not exceed end, the loop is executed; otherwise, program control jumps to the command following **EndFor**.



Note: The **For** command automatically increments the counter variable so that the program can exit the loop after a certain number of repetitions.

At the end of the loop (**EndFor**), program control jumps back to the **For** command, where variable is incremented and compared to *end*.

For example:

```
  :For i,0,5,1
❶ :  Disp I
  :EndFor
❷ :Disp i
```

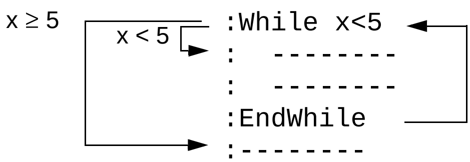
❶ Displays 0, 1, 2, 3, 4, and 5.
❷ Displays 6. When *variable* increments to 6, the loop is not executed.

Note: You can declare the counter variable as local if it does not need to be saved after the program stops.

A **While...EndWhile** loop repeats a block of commands as long as a specified condition is true. The syntax of the **While** command is:

While *condition*

When **While** is executed, the condition is evaluated. If condition is true, the loop is executed; otherwise, program control jumps to the command following **EndWhile.**



Note: The **While** command does not automatically change the condition. You must include commands that allow the program to exit the loop.

At the end of the loop (**EndWhile**), program control jumps back to the **While** command, where condition is re-evaluated.

To execute the loop the first time, the condition must initially be true.

- Any variables referenced in the condition must be set before the **While** command. (You can build the values into the program or prompt the user to enter the values.)
- The loop must contain commands that change the values in the condition, eventually causing it to be false. Otherwise, the condition is always true and the program cannot exit the loop (called an infinite loop).

For example:

```
❶ :0→x
  :While x<5
❷ :  Disp x
❸ :  x+1→x
  :EndWhile
❹ :Disp x
```

❶ Initially sets x.
❷ Displays 0, 1, 2, 3, and 4.
❸ Increments x.
❹ Displays 5. When x increments to 5, the loop is not executed.

Loop...EndLoop Loops

A **Loop...EndLoop** creates an infinite loop, which is repeated endlessly. The **Loop** command does not have any arguments.

```
:Loop
:  --------
:  --------
:EndLoop
:--------
```

Typically, the loop contains commands that let the program exit from the loop. Commonly used commands are: **If**, **Exit**, **Goto**, and **Lbl** (label). For example:

```
  :0→x
  :Loop
  :  Disp x
  :  x+1→x
➊ :  If x>5
  :    Exit
  :EndLoop
➋ :Disp x
```

➊ An **If** command checks the condition.
➋ Exits the loop and jumps to here when x increments to 6.

Note: The **Exit** command exits from the current loop.

In this example, the **If** command can be anywhere in the loop.

When the If command is:	The loop is:
At the beginning of the loop	Executed only if the condition is true.
At the end of the loop	Executed at least once and repeated only if the condition is true.

The **If** command could also use a **Goto** command to transfer program control to a specified **Lbl** (label) command.

Repeating a Loop Immediately

The **Cycle** command immediately transfers program control to the next iteration of a loop (before the current iteration is complete). This command works with **For...EndFor**, **While...EndWhile**, and **Loop...EndLoop**.

Lbl and Goto Loops

Although the **Lbl** (label) and **Goto** commands are not strictly loop commands, they can be used to create an infinite loop. For example:

```
:Lbl START     <-+
:  --------      |
:  --------      |
:Goto START    --+
:--------
```

As with **Loop...EndLoop**, the loop should contain commands that let the program exit from the loop.

Configuring the TI-89 Titanium

Programs can contain commands that change the configuration of the calculator. Because mode changes are particularly useful, the Program Editor's **Mode** toolbar menu makes it easy to enter the correct syntax for the **setMode** command.

Command	Description
getConfg	Returns a list of calculator characteristics.
getFold	Returns the name of the current folder.
getMode	Returns the current setting for a specified mode.
getUnits	Returns a list of default units.
setFold	Sets the current folder.
setGraph	Sets a specified graph format (**Coordinates**, **Graph Order**, etc.).
setMode	Sets any mode except **Current Folder**.
setTable	Sets a specified table setup parameter (**tblStart**, **∆tbl**, etc.)
setUnits	Sets default units for displayed results.
switch	Sets the active window in a split screen, or returns the number of the active window.

Note: The parameter/mode strings used in the **setMode( )**, **getMode( )**, **setGraph( )**, and **setTable( )** functions do not translate into other languages when used in a program. See the *Technical Reference* module.

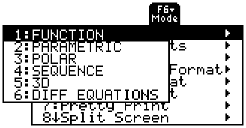
Entering the SetMode Command

In the Program Editor:

1. Position the cursor where you want to insert the **setMode** command.

2. Press:
   [2nd] [F6] to display a list of modes.

   Note: The **Mode** menu does not let you set the **Current Folder** mode. To set this mode, use the **setFold** command.



3. Select a mode to display a menu of its valid settings.
4. Select a setting.

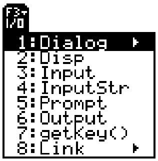
The correct syntax is `:setMode("Graph","FUNCTION")` inserted into your program.

Getting Input from the User and Displaying Output

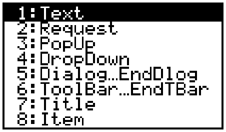
Although values can be built into a program (or stored to variables in advance), a program can prompt the user to enter information while the program is running. Likewise, a program can display information such as the result of a calculation.

F3 I/O Toolbar Menu

To enter most of the commonly used input/output commands, use the Program Editor's [F3] **I/O** toolbar menu.



To see a submenu that lists additional commands, select **1:Dialog.**



Input Commands

Command	Description
getKey	Returns the key code of the next key pressed. See the *Technical Reference* module for a listing of key codes.
Input	Prompts the user to enter an expression. The expression is treated according to how it is entered. For example: • A numeric expression is treated as an expression. • An expression enclosed in "quotes" is treated as a string. **Input** can also display the Graph screen and let the user update the variables xc and yc (rc and θc in polar mode) by positioning the graph cursor.
InputStr	Prompts the user to enter an expression. The expression is always treated as a string; the user does not need to enclose the expression in "quotes".
PopUp	Displays a pop-up menu box and lets the user select an item.
Prompt	Prompts the user to enter a series of expressions. As with **Input**, each expression is treated according to how it is entered.

Command	Description
Request	Displays a dialog box that prompts the user to enter an expression. **Request** always treats the entered expression as a string.

Note: String input cannot be used in a calculation. To convert a string to a numeric expression, use the **expr** command.

Output Commands

Command	Description
ClrIO	Clears the Program I/O screen.
Disp	Displays an expression or string on the Program I/O screen. **Disp** can also display the current contents of the Program I/O screen without displaying additional information.
DispG	Displays the current contents of the Graph screen.
DispHome	Displays the current contents of the Home screen.
DispTbl	Displays the current contents of the Table screen.
Output	Displays an expression or string starting at specified coordinates on the Program I/O screen.
Format	Formats the way in which numeric information is displayed.
Pause	Suspends program execution until the user presses [ENTER]. Optionally, you can display an expression during the pause. A pause lets users read your output and decide when they are ready to continue.

Command	Description
Text	Displays a dialog box that contains a specified character string.

Notes:

- In a program, simply performing a calculation does not display the result. You must use an output command.
- After **Disp** and **Output**, the program immediately continues. You may want to add a **Pause** command.

Graphical User Interface Commands

Command	Description
Dialog... EndDlog	Defines a program block (consisting of **Title**, **Request**, etc., commands) that displays a dialog box.
Toolbar... EndTbar	Defines a program block (consisting of Title, Item, etc., commands) that replaces the toolbar menus. The redefined toolbar is in effect only while the program is running and only until the user selects an item. Then the original toolbar is redisplayed.
CustmOn... CustmOff	Activates or removes a custom toolbar.
Custom... EndCustm	Defines a program block that displays a custom toolbar when the user presses 2nd [CATALOG]. That toolbar remains in effect until the user presses 2nd [CATALOG] again or changes applications.

Command	Description
DropDown	Displays a drop-down menu within a dialog box.
Item	Displays a menu item for a redefined toolbar.
Request	Creates an input box within a dialog box.
Text	Displays a character string within a dialog box.
Title	Displays the title of a dialog box or a menu title within a toolbar.

Notes:

- When you run a program that sets up a custom toolbar, that toolbar is still available even after the program has stopped.
- **Request** and **Text** are stand-alone commands that can also be used outside of a dialog box or toolbar program block.

Creating a Custom Menu

The custom menu feature lets you create your own toolbar menu. A custom menu can contain any available function, instruction, or set of characters. The calculator has a default custom menu that you can modify or redefine.

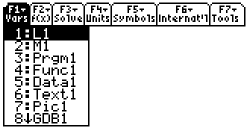
When you create a custom menu, you can let the user turn it on and off manually, or you can let a program turn it on and off automatically.

To:	Do this:
Turn on the custom menu	From the Home screen or any other application: • Press [2nd] [CATALOG]. From the Home screen or a program: • Execute the **CustmOn** command.
Turn off the custom menu	From any application: • Press [2nd] [CATALOG] again. – or – • Go to a different application. Using the default custom menu on the Home screen: 1. Select the **Tools** menu: [2nd] [F7] Select **3:CustmOff**. This pastes **CustmOff** in the entry line. 2. Press [ENTER]. You can also use **CustmOff** in a program.  CustmOff

Note: When the custom menu is turned on, it replaces the normal toolbar menu. Unless a different custom menu has been created, the default custom menu is displayed.

To create a custom menu, use the following general structure.

Custom
: **Title** *title of F1 menu:*
:   **Item** *item 1*
:   **Item** *item 2*
:   ...
: **Title** *title of F2 menu*
:   ...
: **Title** *title of F3 menu*
:   ...



Note: When the user selects a menu item, the text defined by that **Item** command is pasted to the current cursor location.

For example:

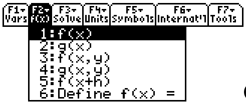
```
:Custom
:Title "Vars"
:Item "L1":Item "M1":Item "Prgm1":Item "Func1":Item "Data1"
:Item "Text1":Item "Pic1":Item "GDB1":Item "Str1"
```

❶ `:Title "f(x)"`
❶ `:Item "f(x)":Item "g(x)":Item "f(x,y)":Item "g(x,y)"`
❶ `:Item "f(x+h)":Item "Define f(x) ="`

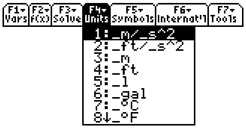
```
:Title "Solve"
:Item "Solve(":Item " and ":Item "{x,y}"
:Item "Solve( and ,{x,y})"
```

```
❷  :Title "Units"
❷  :Item "_m/_s^2":Item "_ft/_s^2":Item "_m":Item "_ft":Item "_l"
❷  :Item "_gal":Item "_\o\C":Item "_\o\F":Item "_kph":Item "_mph"

   :Title "Symbols"
   :Item "#":Item "\beta\":Item "?":Item "~":Item "&"

   :Title "Internat'l"
   :Item "\e`\":Item "\e'\":Item "\e^\":Item "\a`\"
   :Item "\u`\":Item "\u^\":Item "\o^\":Item "\c,\":Item "\u..\"

   :Title "Tools"
   :Item "ClrHome":Item "NewProb":Item "CustmOff"

:EndCustm

:CustmOn
```

Note: The following may be slightly different than the default custom menu on your calculator.



❶



❷

Note: See how "_\o\C" and "_\o\F" display as °C and °F in the menu. Similarly, see the international accented characters.

To modify the default custom menu, use **3:Restore custom default** (as described below) to get the commands for the default menu. Copy those commands, use the Program

Editor to create a new program, and paste them into the blank program. Then modify the commands as necessary.

Note: This inserts all the commands on a single line. You do not need to split them into separate lines.

You can create and use only one custom menu at a time. If you need more, write a separate program for each custom menu. Then run the program for the menu you need.

Restoring the Default Custom Menu

To restore the default:

1. From the Home screen's normal menu (not the custom menu), select **Clean Up**: [2nd] [F6]

2. Select **3:Restore custom default.**

   This pastes the commands used to create the default menu into the entry line.

3. Press [ENTER] to execute the commands and restore the default.

When you restore the default, any previous custom menu is erased. If the previous menu was created with a program, you can run the program again if you want to reuse the menu later.

Creating a Table or Graph

To create a table or a graph based on one or more functions or equations, use the commands listed in this section.

Table Commands

Command	Description
DispTbl	Displays the current contents of the Table screen.
setTable	Sets the Graph <–> Table or Independent table parameters. (To set the other two table parameters, you can store the applicable values to the **tblStart** and **∆tbl** system variables.)
Table	Builds and displays a table based on one or more expressions or functions.

Graphing Commands

Command	Description
ClrGraph	Erases any functions or expressions that were graphed with the **Graph** command.
Define	Creates a user-defined function.
DispG	Displays the current contents of the Graph screen.
FnOff	Deselects all (or only specified) Y= functions.
FnOn	Selects all (or only specified) Y= functions.

Command	Description
Graph	Graphs one or more specified expressions, using the current graphing mode.
Input	Displays the Graph screen and lets the user update the variables xc and yc (rc and θc in polar mode) by positioning the graph cursor.
NewPlot	Creates a new stat plot definition.
PlotsOff	Deselects all (or only specified) stat data plots.
PlotsOn	Selects all (or only specified) stat data plots.
setGraph	Changes settings for the various graph formats (**Coordinates**, **Graph Order**, etc.).
setMode	Sets the Graph mode, as well as other modes.
Style	Sets the display style for a function.
Trace	Lets a program trace a graph.
ZoomBox – to – **ZoomTrig**	Perform all of the **Zoom** operations that are available from the F2 toolbar menu on the Y= Editor, Window Editor, and Graph screen.

Note: More information is available about using **setMode.**

Graph Picture and Database Commands

Command	Description
AndPic	Displays the Graph screen and superimposes a stored graph picture by using **AND** logic.

Command	Description
CyclePic	Animates a series of stored graph pictures.
NewPic	Creates a graph picture variable based on a matrix.
RclGDB	Restores all settings stored in a graph database.
RclPic	Displays the Graph screen and superimposes a stored graph picture by using **OR** logic.
RplcPic	Clears the Graph screen and displays a stored graph picture.
StoGDB	Stores the current graph settings to a graph database variable.
StoPic	Copies the Graph screen (or a specified rectangular portion) to a graph picture variable.
XorPic	Displays the Graph screen and superimposes a stored graph picture by using **XOR** logic.

Note: For information about graph pictures and databases, also refer to *Additional Graphing Topics*.

Drawing on the Graph Screen

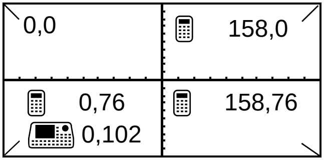
To create a drawing object on the Graph screen, use the commands listed in this section.

Pixel vs. Point Coordinates

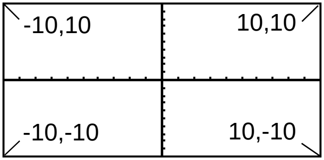
When drawing an object, you can use either of two coordinate systems to specify a location on the screen.

- **Pixel coordinates** — Refer to the pixels that physically make up the screen. These are independent of the viewing window because the screen is always:
  159 (0 to 158) pixels wide and 77 (0 to 76) pixels tall.

- **Point coordinates** — Refer to the coordinates in effect for the current viewing window (as defined in the Window Editor).



Pixel coordinates
(independent of viewing window)



Point coordinates
(for standard viewing window)

Note: For information about pixel coordinates in split screens, refer to the *Data/Matrix Editor* module.

Many drawing commands have two forms: one for pixel coordinates and one for point coordinates.

Note: Pixel commands start with **Pxl**, such as **PxlChg.**

Erasing Drawn Objects

Command	Description
ClrDraw	Erases all drawn objects from the Graph screen.

Drawing a Point or Pixel

Command	Description
PtChg or **PxlChg**	Toggles (inverts) a pixel at the specified coordinates. **PtChg**, which uses point coordinates, affects the pixel closest to the specified point. If the pixel is off, it is turned on. If the pixel is on, it is turned off.
PtOff or **PxlOff**	Turns off (erases) a pixel at the specified coordinates. **PtOff**, which uses point coordinates, affects the pixel closest to the specified point.
PtOn or **PxlOn**	Turns on (displays) a pixel at the specified coordinates. **PtOn**, which uses point coordinates, affects the pixel closest to the specified point.
PtTest or **PxlTest**	Returns true or false to indicate if the specified coordinate is on or off, respectively.
PtText or **PxlText**	Displays a character string at the specified coordinates.

Drawing Lines and Circles

Command	Description
Circle or **PxlCrcl**	Draws, erases, or inverts a circle with a specified center and radius.
DrawSlp	Draws a line with a specified slope through a specified point.
Line or **PxlLine**	Draws, erases, or inverts a line between two sets of coordinates.

Command	Description
LineHorz or **PxlHorz**	Draws, erases, or inverts a horizontal line at a specified row coordinate.
LineTan	Draws a tangent line for a specified expression at a specified point. (This draws the tangent line only, not the expression.)
LineVert or **PxlVert**	Draws, erases, or inverts a vertical line at a specified column coordinate.

Drawing Expressions

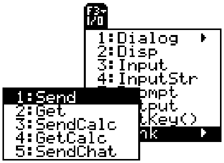
Command	Description
DrawFunc	Draws a specified expression.
DrawInv	Draws the inverse of a specified expression.
DrawParm	Draws a parametric equation using specified expressions as its x and y components.
DrawPol	Draws a specified polar expression.
DrwCtour	Draws contours in 3D graphing mode.
Shade	Draws two expressions and shades the areas where *expression1* < *expression2*.

Accessing Another TI-89 Titanium, a CBL 2, or a CBR

If you link two graphing calculators (described in the *Connectivity* module), programs on both units can transmit variables between them. If you link a TI-89 Titanium to a Calculator-Based Laboratory™ (CBL 2™) or a Calculator-Based Ranger™ (CBR™), a program on the TI-89 Titanium can access the CBL 2 or CBR.

F3 I/O Toolbar Menu

Use the Program Editor's [F3] **I/O** toolbar menu to enter the commands in this section.



1. Press [F3] and select **8:Link**.
2. Select a command.

Accessing Another TI-89 Titanium

When two calculators are linked, one acts as a receiving unit and the other as a sending unit.

Command	Description
GetCalc	Executed on the receiving unit. Sets up the unit to receive a variable via the I/O port. • After the receiving unit executes **GetCalc**, the sending unit must execute **SendCalc**. • After the sending unit executes **SendCalc**, the sent variable is stored on the receiving unit (in the variable name specified by **GetCalc**).
SendCalc	Executed on the sending unit. Sends a variable to the receiving unit via the I/O port. • Before the sending unit executes **SendCalc**, the receiving unit must execute **GetCalc**.

Note: For a sample program that synchronizes the receiving and sending units so that **GetCalc** and **SendCalc** are executed in the proper sequence, refer to “Transmitting Variables under Program Control” in *Connectivity*.

Accessing a CBL 2 or CBR

For additional information, refer to the manual that comes with the CBL 2 or CBR unit.

Command	Description
Get	Gets a variable from an attached CBL 2 or CBR and stores it in the graphing calculator.

Command	Description
Send	Sends a list variable from the graphing calculator to the CBL 2 or CBR.

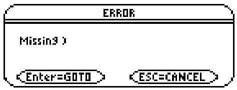
Debugging Programs and Handling Errors

After you write a program, you can use several techniques to find and correct errors. You can also build an error-handling command into the program itself.

Run-Time Errors

The first step in debugging your program is to run it. The graphing calculator automatically checks each executed command for syntax errors. If there is an error, a message indicates the nature of the error.

- To display the program in the Program Editor, press [ENTER]. The cursor appears in the approximate area of the error.



- To cancel program execution and return to the Home screen, press [ESC].

If your program allows the user to select from several options, be sure to run the program and test each option.

Debugging Techniques

Run-time error messages can locate syntax errors but not errors in program logic. The following techniques may be useful.

- During testing, do not use local variables so that you can check the variable values after the program stops. When the program is debugged, declare the applicable variables as local.
- Within a program, temporarily insert **Disp** and **Pause** commands to display the values of critical variables.
  - **Disp** and **Pause** cannot be used in a user-defined function. To temporarily change the function into a program, change **Func** and **EndFunc** to **Prgm** and **EndPrgm**. Use **Disp** and **Pause** to debug the program. Then remove **Disp** and **Pause** and change the program back into a function.
- To confirm that a loop is executed the correct number of times, display the counter variable or the values in the conditional test.
- To confirm that a subroutine is executed, display messages such as `Entering subroutine` and `Exiting subroutine` at the beginning and end of the subroutine.

Error-Handling Commands

Command	Description
Try...EndTry	Defines a program block that lets the program execute a command and, if necessary, recover from an error generated by that command.
ClrErr	Clears the error status and sets the error number in system variable Errornum to zero.
PassErr	Passes an error to the next level of the **Try...EndTry** block.

Example: Using Alternative Approaches

The example in the *Previews* module shows a program that prompts the user to enter an integer, sums all integers from 1 to the entered integer, and displays the result. This section gives several approaches that you can use to achieve the same goal.

Example 1

This example uses **InputStr** for input, a **While...EndWhile** loop to calculate the result, and **Text** to display the result.

```
  :prog1()
  :Prgm
❶ :InputStr "Enter an integer",n
❷ :expr(n)→n
  :0→temp:1→I
❸ :While i≤n
⋮ :  temp+i→temp
⋮ :  i+1→I
❸ :EndWhile
❹ :Text "The answer is "&string(temp)
  :EndPrgm
```

❶ Prompts for input on Program I/O screen.
❷ Converts string entered with **InputStr** to an expression.
❸ Loop calculation.
❹ Displays output in a dialog box.

Note: For ≤, type [♦] [0] (zero). For **&**, press:
[♦] [×] (times)

Example 2

This example uses **Prompt** for input, **Lbl**, and **Goto** to create a loop, and **Disp** to display the result.

```
  :prog2()
  :Prgm
❶ :Prompt n
  :0→temp:1→I
❷ :Lbl top
⋮ :  temp+i→temp
⋮ :  i+1→I
⋮ :  If i≤n
❷ :    Goto top
❸ :Disp temp
  :EndPrgm
```

❶ Prompts for input on Program I/O screen.
❷ Loop calculation.
❸ Displays output on Program I/O screen.

Note: Because **Prompt** returns n as a number, you do not need to use **expr** to convert n.

Example 3

This example uses **Dialog...EndDlog** to create dialog boxes for input and output. It uses **Loop...EndLoop** to calculate the result.

```
  :prog3()
  :Prgm
❶ :Dialog
⋮ :  Title "Enter an integer"
⋮ :  Request "Integer",n
❶ :EndDlog
❷ :expr(n)→n
  :0→temp:0→I
❸ :Loop
⋮ :  temp+i→temp
⋮ :  i+1→I
⋮ :  If i>n
⋮ :    Exit
❸ :EndLoop
❹ :Dialog
⋮ :  Title "The answer is"
⋮ :  Text string(temp)
❹ :EndDlog
  :EndPrgm
```

❶ Defines a dialog box for input.
❷ Converts string entered with **Request** to an expression.
❸ Loop calculation.
❹ Defines a dialog box for output.

Example 4

This example uses built-in functions to calculate the result without using a loop.

```
  :prog4()
  :Prgm
❶ :Input "Enter an integer",n
❷ :sum(seq(i,i,1,n))→temp
❸ :Disp temp
  :EndPrgm
```

❶ Prompts for input on Program I/O.
❷ Calculates sum.
❸ Displays output on Program I/O screen.

Note: Because **Input** returns n as a number, you do not need to use **expr** to convert n.

Function	Used in this example to:
seq	Generate the sequence of integers from **1** to **n**. **seq(***expression*, *var*, *low*, *high* [,*step*]**)** ❶ ❷ ❸ ❸ ❹ ❶ expression used to generate the sequence ❷ variable that will be incremented ❸ initial and final values of *var* ❹ increment for *var* ; if omitted, uses 1
sum	Sum the integers in the list generated by **seq**.

Assembly-Language Programs

You can run programs written for the TI-89 Titanium in assembly language. Typically, assembly-language programs run much faster and provide greater control than the keystroke programs that you write with the built-in Program Editor.

Where to Get Assembly-Language Programs

Assembly-language programs, as well as keystroke programs, are available on the Texas Instruments web site at education.ti.com.

The programs available from this site provide additional functions or features that are not built into the TI-89 Titanium. Check the Texas Instruments web site for up-to-date information.

After downloading a program from the web to your computer, use a USB cable or TI-GRAPH LINK™ computer-to-calculator cable and TI Connect software to send the program to your TI-89 Titanium.

For Flash App installation instructions, see education.ti.com/guides.

Note about TI-GRAPH LINK

If you have a TI-GRAPH LINK™ computer-to-calculator cable and software for the TI-89 or TI-92 Plus, be aware that the TI-GRAPH LINK software is not compatible with the TI-89 Titanium. The cable, however, works with all units. Use TI Connect software on your computer.

You can purchase computer-to-calculator and unit-to-unit cables from the TI Online Store at education.ti.com/buy.

Running an Assembly-Language Program

After a TI-89 Titanium assembly-language program is stored on your unit, you can run the program from the Home screen just as you would any other program.

- If the program requires one or more arguments, type them within the ( ). Refer to the program's documentation to find out about required arguments.

- If the program is not in the current folder, be sure to specify the pathname.

You can call an assembly-language program from another program as a subroutine, delete it, or use it the same as any other program.

Shortcuts to Run a Program

On the Home screen, you can use keyboard shortcuts to run up to six user-defined or assembly-language programs. However, the programs must have the following names.

On Home screen, press:	To run a program, if any, named:
[♦] 1	kbdprgm1( )
⋮	⋮
[♦] 6	kbdprgm6( )

The programs must be stored in the MAIN folder. Also, you cannot use a shortcut to run a program that requires an argument.

If you have a program with a different name and you would like to run it with a keyboard shortcut, copy or rename the existing program to **kbdprgm1( )**, etc.

You Cannot Edit an Assembly-Language Program

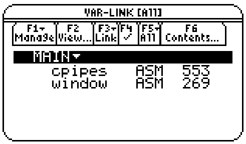
You cannot use your TI-89 Titanium to edit an assembly-language program. The built-in Program Editor will not open assembly-language programs.

Displaying a List of Assembly-Language Programs

To list the assembly-language programs stored in memory:

1. Display the **VAR-LINK** screen ([2nd] [VAR-LINK]).
2. Press [F2] **View.**
3. Select the applicable folder (or All folders) and set **Var Type = Assembly.**

4. Press [ENTER] to display the list of assembly-language programs.



Note: Assembly-language programs have an ASM data type.

For Information about Writing an Assembly-Language Program

The information required to teach a novice programmer how to write an assembly-language program is beyond the scope of this book. However, if you have a working knowledge of assembly language, please check the Texas Instruments web site (education.ti.com) for specific information about how to access TI-89 Titanium features.

The graphing calculator also includes an **Exec** command that executes a string consisting of a series of Motorola 68000 op-codes. These codes act as another form of an assembly-language program. Check the Texas Instruments web site for available information.

Note: You must use a computer to write assembly-language programs. You cannot create assembly-language programs from the calculator keyboard.

Warning: Exec gives you access to the full power of the microprocessor. Please be aware that you can easily make a mistake that locks up the calculator and causes you to lose your data. We suggest you make a backup of the calculator contents before attempting to use the **Exec** command.

Text Editor

Starting a Text Editor Session

Each time you start the Text Editor, you can start a new text session, resume the current session (the session that was displayed the last time you used the Text Editor), or open a previous session.

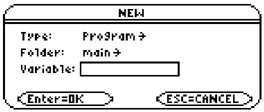
Starting a New Session

1. Press [APPS] and then select the Text Editor icon. Press [ENTER].

2. Select **3:New.**

   The **NEW** dialog box is displayed.

3. Specify a folder and text variable that you want to use to store the new session.



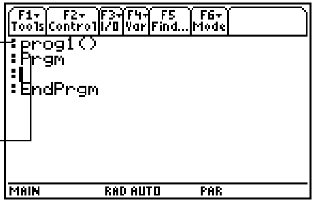
Item	Description
Type	Automatically set as **Text** and cannot be changed.
Folder	Shows the folder in which the text variable will be stored. For information about folders, refer to the *Calculator Home Screen module*. To use a different folder, press ► to display a menu of existing folders. Then select a folder.

Item	Description
Variable	Type a variable name. If you specify a variable that already exists, an error message will be displayed when you press [ENTER]. When you press [ESC] or [ENTER] to acknowledge the error, the **NEW** dialog box is redisplayed.

4. Press [ENTER] (after typing in an input box such as **Variable**, you must press [ENTER] twice) to display an empty Text Editor screen.

A colon marks the beginning of a paragraph.

The blinking cursor shows where typed text will appear.



You can now use the Text Editor as described in the remaining sections of this module.

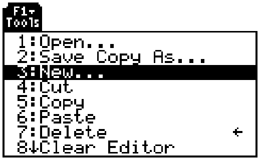
Note: Your session is saved automatically as you type. You do not need to save a session manually before leaving the Text Editor, starting a new session, or opening a previous one.

Resuming the Current Session

You can leave the Text Editor and go to another application at any time. To return to the session that was displayed when you left the Text Editor, launch Text Editor again and select **1:Current**.

Starting a New Session from the Text Editor

To leave the current Text Editor session and start a new one:



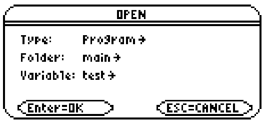
1. Press [F1] and select **3:New**.
2. Specify a folder and text variable for the new session.
3. Press [ENTER] twice.

Opening a Previous Session

You can open a previous Text Editor session at any time.

1. From within the Text Editor, press [F1] and select **1:Open**.
   — or —

   From any application, launch Text Editor again and select **2:Open**.



2. Select the applicable folder and text variable.
3. Press [ENTER].

Note: By default, **Variable** shows the first existing text variable in alphabetic order.

Copying a Session

In some cases, you may want to copy a session so that you can edit the copy while retaining the original.

1. Display the session you want to copy.
2. Press [F1] and select **2:Save Copy As**.
3. Specify the folder and text variable for the copied session.
4. Press [ENTER] twice.

Note about Deleting a Session

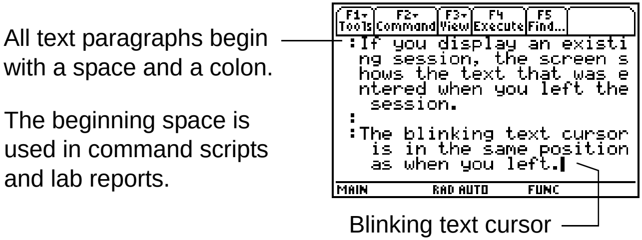
Because all Text Editor sessions are saved automatically, you can accumulate quite a few previous sessions, which take up memory storage space.

To delete a session, use the VAR-LINK screen ([2nd] [VAR-LINK]) to delete that session's text variable. For information about VAR-LINK, refer to *Memory and Variable Management*.

Entering and Editing Text

After beginning a Text Editor session, you can enter and edit text. In general, use the same techniques that you have already used to enter and edit information on the Home screen's entry line.

When you create a new Text Editor session, you see an empty screen. When you open a previous session or return to the current session, you see the existing text for that session.



You do not need to press [ENTER] at the end of each line. At the end of a line, the next character you type wraps to the next line. Press [ENTER] only when you want to start a new paragraph.

As you reach the bottom of the screen, previous lines scroll off the top of the screen.

Using a USB cable and TI Connect™ software with the TI-89 Titanium, you can use the computer keyboard to type a text file and then send that file to the TI-89 Titanium. This is useful if you need to create a lengthy text file.

For information about obtaining cables or updated TI Connect™ software, check the TI web site at education.ti.com, or contact Texas Instruments at TI-Cares™.

Notes:

- Use the cursor pad to scroll through a session or position the text cursor.

- Press [2nd] [up] or [2nd] [down] to scroll up or down one screen at a time, and [♦] [up] or [♦] [down] to go to the top or bottom of the text session.

Typing Alphabetic Characters

To:	Press:
Type a single lowercase alpha character.	[alpha] and then the letter key (status line shows)
Type a single uppercase alpha character.	[↑] and then the letter key (status line shows)
Type a space.	[alpha] [␣] (alpha function of the [(-)] key)
Turn on lowercase alpha-lock.	[2nd] [a-lock] (status line shows)
Turn on uppercase ALPHA-lock.	[↑] [a-lock] (status line shows)
Turn off alpha-lock.	[alpha] (turns off upper- and lowercase lock)

Note: On the TI-89 Titanium, you do not need [alpha] or alpha-lock to type x, y, z, or t. But you must use [↑] or uppercase ALPHA-lock for X, Y, Z, or T. On the TI-89 Titanium, alpha-lock is always turned off when you change applications, such as going from the Text Editor to the Home screen.

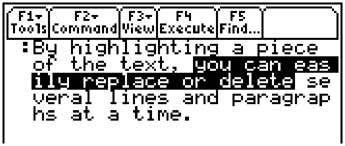
On the TI-89 Titanium, while either type of alpha-lock is on:

- To type a period, comma, or other character that is the primary function of a key, you must turn alpha-lock off.
- To type a second function character such as [2nd] [{], you do not need to turn alpha-lock off. After you type the character, alpha-lock remains on.

Deleting Characters

To delete:	Press:
The character to the left of the cursor	[←] or [F1] **7**
The character to the right of the cursor	[♦] [DEL] (same as [♦] [←])
All characters to the right of the cursor through the end of the paragraph	[CLEAR]
All characters in the paragraph (regardless of the cursor's position in that paragraph)	[CLEAR] [CLEAR]

Note: If there are no characters to the right of the cursor, [CLEAR] erases the entire paragraph.

To:	Do this:
Highlight text	Move the cursor to the beginning or end of the text. Hold [↑] and press: • ◀ or ▶ to highlight characters to the left or right of the cursor, respectively. • ▼ or ▲ to highlight all characters up to the cursor position on the next or previous line, respectively. 

Note: To remove highlighting without replacing or deleting, move the cursor.

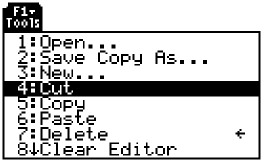
Replacing or Deleting Highlighted Text

To:	Do this:
Replace highlighted text	Type the new text.
Delete highlighted text	Press [←].

Cutting and copying both place highlighted text into the clipboard of the TI-89 Titanium. Cutting deletes the text from its current location (used to move text) and copying leaves the text.

1. Highlight the text you want to move or copy.

2. Press [F1].



3. Select the applicable menu item.

   - To move the text, select **4:Cut**.
     — or —
   - To copy the text, select **5:Copy**.

   Note: You can press:
   [◆] [CUT], [◆] [COPY], [◆] [PASTE]
   to cut, copy, and paste without having to use the [F1] toolbar menu.

4. Move the text cursor to the location where you want to insert the text.

5. Press [F1] and then select **6:Paste**.

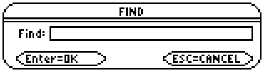
You can use this general procedure to cut, copy, and paste text:

- Within the same text session.
- From one text session to another. After cutting or copying text in one session, open the other session and then paste the text.
- From a text session to a different application. For example, you can paste the text into the Home screen’s entry line.

From the Text Editor:

1. Place the text cursor at any location preceding the text you want to search for. All searches start at the current cursor location.

2. Press [F5].



3. Type the search text.

   The search is not case sensitive. For example: CASE, case, and Case have the same effect.

   Note: The **FIND** dialog box retains the last search text you entered. You can type over it or edit it.

4. Press [ENTER] twice.

If the search text is:	The cursor:
Found	Moves to beginning of the search text.
Not found	Does not move.

Inserting or Overtyping a Character

By default, the TI-89 Titanium is in insert mode. To toggle between insert and overtype mode, press 2nd [INS].

If the TI-89 Titanium is in:	The next character you type:
Insert mode — Thin cursor between characters	Will be inserted at the cursor.
Overtype mode — Cursor highlights a character	Will replace the highlighted character.

Note: Look at the shape of the cursor to see if you're in insert or overtype mode.

Clearing the Text Editor

To erase all existing paragraphs and display an empty text screen, press F1 and then select **8:Clear Editor**.

Entering Special Characters

You can use the CHAR menu to select any special character from a list. You can also type certain commonly used characters from the keyboard. To see which characters are available from the keyboard, you can display a map that shows the characters and their corresponding keys.

1. Press [2nd] [CHAR].
2. Select the applicable category.

   A menu lists the characters in that category.
3. Select a character. You may need to scroll through the menu.

   Note: For accented characters, select International. Commonly used international characters are also available from the default custom menu ([2nd] [CUSTOM]).

↓ indicates that you can scroll.

Displaying the Keyboard Map

The keyboard map shows several shortcuts that let you enter certain special characters from the keyboard. It also shows some shortcuts for other calculator features.

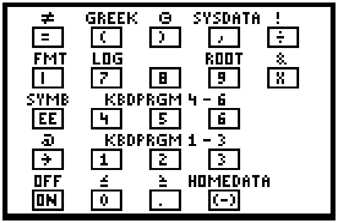
The keyboard map does not display all available shortcuts. Refer to the inside front and the inside back covers of this guidebook for a complete list of shortcut keys.

To access the shortcuts, first press the [2nd] key. Some special characters are marked on the keyboard, but most are not.

On the TI-89 Titanium:

- Press [♦] [EE] to display the keyboard map.

- Press [ESC] to exit the map.



TI-89 Titanium Keyboard map

To access the TI-89 Titanium shortcuts, first press the [♦] key.

TI-89 Titanium keyboard map feature shortcuts:

- GREEK ([♦] [(]) — Accesses the Greek character set (described later in this section).
- SYSDATA ([♦] [,]) — Copies the current graph coordinates to the system variable sysdata.
- FMT ([♦] [|]) — Displays the FORMATS dialog box.
- KBDPRGM1 – 6([♦] 1 through [♦] 6) — If you have user-defined or assembly-language programs named kbdprgm1() through kbdprgm6(), these shortcuts run the corresponding program.
- OFF ([♦] [OFF]) — Similar to [2nd] [OFF] except:
  - You can use [♦] [OFF] if an error message is displayed.
  - When you turn the TI-89 Titanium on again, it will be exactly as you left it.

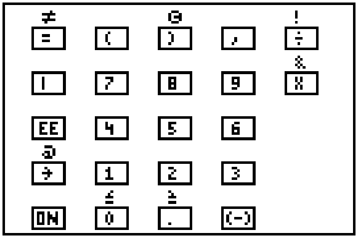
- HOMEDATA ([♦] [(-)]) — Copies the current graph coordinates to the Home screen's history area.

Typing Special Symbols from the Keyboard

Note: To help you find the applicable keys, these maps show only the special symbols.

On the TI-89 Titanium:

► Press [♦] and then the key for the symbol. For example: [♦] [×] (times) displays &.



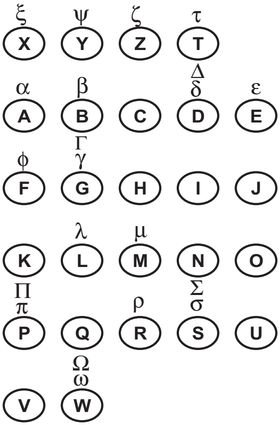
These special symbols are not affected by whether Alpha-Lock is on or off.

Typing Greek Letters from the Keyboard

Press the key combination that accesses the Greek character set on your calculator. Then select the applicable alpha character on the keyboard to enter a Greek letter.

On the TI-89 Titanium:

▶ Press [♦] [(] to access the Greek character set.



Note: If you press a key combination that does not access a Greek letter, you get the normal letter for that key. Your calculator does not display a map of Greek letters; the map shown here is for reference only.

Several keys let you access lowercase and uppercase Greek letters. For example:

On the TI-89 Titanium:

- Press [♦] [(] to access the Greek character set.
- Press [♦] [(] [alpha] + letter to access lowercase Greek letters. Example: [♦] [(] [alpha] [W] displays ω

- Press [◆] [(] [↑] + letter to access uppercase Greek letters. Example: [◆] [(] [↑] [W] displays Ω

The exact keys that you press on the TI-89 Titanium depend on whether alpha-lock is on or off. For example:

On the TI-89 Titanium, if:	Then:
Alpha-lock is off.	[◆] [(] **X** or [◆] [(] [alpha] **X** displays ξ. ([alpha] is not required for X, Y, Z, or T.) [◆] [(] [alpha] **W** displays ω. [◆] [(] [↑] **W** displays Ω. ([↑] is used for uppercase letters.)
Lowercase alpha-lock ([2nd] [a-lock]) is on.	[◆] [(] **X** displays ξ. [◆] [(] **W** displays ω. [◆] [(] [↑] **W** displays Ω.
Uppercase ALPHA-LOCK ([↑] [a-lock]) is on.	[◆] [(] **X** displays ξ. [◆] [(] **W** displays Ω. [◆] [(] [↑] **W** displays Ω.

Important: If you press [alpha] on the TI-89 Titanium to access a Greek letter while alpha-lock is on, it turns alpha-lock off.

For a List of All Special Characters

For a list of all special characters, refer to the *Technical Reference* module.

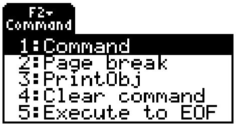
Entering and Executing a Command Script

By using a command script, you can use the Text Editor to type a series of command lines that can be executed at any time on the Home screen. This lets you create interactive example scripts in which you predefine a series of commands and then execute them individually.

Inserting a Command Mark

In the Text Editor:

1. Place the cursor on the line for the command.
2. Press [F2] to display the Command toolbar menu.



3. Select **1:Command**.

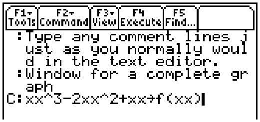
   C is displayed at the beginning of the text line (to the left of the colon).

   Note: This does not insert a new line for the command, it simply marks an existing line as a command line.

4. Type a command just as you would on the Home screen.

   The line can contain only the command, with no additional text.

   Note: You can mark a line as a command either before or after typing the command on that line.



You can type multiple commands on the same line if you type a colon to separate the commands.

Deleting a Command Mark

This deletes only the **C** mark; it does not delete the command text itself.

1. Place the cursor anywhere on the marked line.
2. Press [F2] and select **4:Clear command**.

Executing a Command

To execute a command, you must first mark the line with a **C**. If you execute a line that is not marked with **C**, it will be ignored.

1. Place the cursor anywhere on the command line.
2. Press [F4].

The command is copied to the entry line on the Home screen and executed. The Home screen is displayed temporarily during execution, and then the Text Editor is redisplayed.

After execution, the cursor moves to the next line in the script so that you can continue to execute a series of commands.

Note: To examine the result on the Home screen, use a split screen or press [HOME]

Splitting the Text Editor/ Home Screen

With a split screen, you can view your command script and see the result of an executed command at the same time.

To:	Press:
Split the screen	[F3] and select **1:Script view**. 
Return to a full screen Text Editor	[F3] and select **2:Clear split**.

You can also use [MODE] to set up a split screen manually. However, [F3] sets up a Text Editor/Home screen split much easier than [MODE].

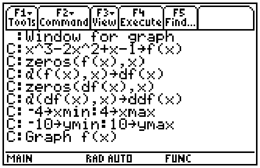
- The active application is indicated by a thick border. (By default, the Text Editor is the active application.)
- To switch between the Text Editor and the Home screen, press [2nd] [⇔] (second function of [APPS]).

Creating a Script from Your Home Screen Entries

From the Home screen, you can save all the entries in the history area to a text variable. The entries are automatically saved in a script format so that you can open the text variable in the Text Editor and execute the entries as commands.

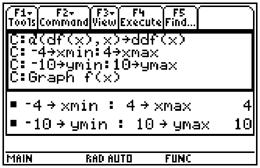
For information, refer to “Saving the Home Screen Entries as a Text Editor Script” in the *Calculator Home Screen* module.

1. Type your script. Press [F2] and select **1:Command** to mark the command lines.
2. Press [F3] and select **1:Script view**.



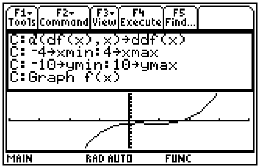
3. Move the cursor to the first command line. Then press [F4] to execute the command.

   Note: Some commands take longer to execute. Wait until the **Busy** indicator disappears before pressing [F4] again.



4. Continue using [F4] to execute each command, but stop just before executing the Graph command.
5. Execute the Graph command.

   Note: In this example, the Graph command displays the Graph screen in place of the Home screen.



6. Press [F3] and select **2:Clear split** to return to a full screen Text Editor.

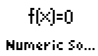
Numeric Solver

Displaying the Solver and Entering an Equation

After you display the Numeric Solver, start by entering the equation that you want to solve.

Displaying the Numeric Solver

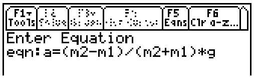
To display the Numeric Solver, press [APPS] and then select the Numeric Solver icon. Press [ENTER].



The Numeric Solver screen shows the last entered equation, if any.

Entering an Equation

On the **eqn:** line, type in your equation.



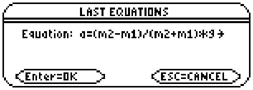
You can:	For example:
Type an equation directly.	**a=(m2−m1)/(m2+m1)∗g** **a+b=c+sin(d)**

You can:	For example:
Refer to a function or equation defined elsewhere. **Notes:** • Do not use system function names (such as **y1(x)** or **r1(θ)**) as simple variables (**y1** or **r1**). • Be careful with implied multiplication. For example, **a(m2+m1)** is treated as a function reference, not as **a∗(m2+m1)**.	Suppose you defined **y1(x)** on either the: • Y= Editor: **y1(x)=1.25x∗cos(x)** – or – • Home screen: **Define y1(x)=1.25x∗cos(x)** In the Numeric Solver, you then would enter: **y1(x)=0** or **y1(t)=0**, etc. The argument does not have to match the one used to define the function or equation.
Type an expression without an = sign. **Note:** When you define the variables, you can either define **exp** or solve for it.	**e+f−ln(g)** After you press [ENTER], the expression is set equal to a system variable called exp and entered as: exp=e+f−ln(g)
Recall a previously entered equation or open a saved equation. **Note:** After you press [ENTER] the current equation is stored automatically to the system variable **eqn**.	Refer to the applicable heading later in this section.

Your most recently entered equations (up to 11 with the default setting) are retained in memory. To recall one of these equations:

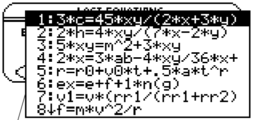
1. From the Numeric Solver screen, press [F5].

   A dialog box displays the most recently entered equation.



2. Select an equation.

   - To select the displayed equation, press [ENTER].
   - To select a different equation, press ▸ to display a list. Then select the one you want.



Only unique equations are listed. If you re-enter the same equation 5 times, it appears only once.

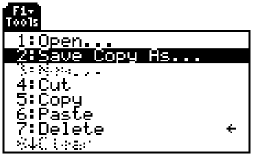
   Note: You can specify how many equations are retained. From the Numeric Solver, press [F1] and select **9:Format** (or use [▦] [◆] [|]). Then select a number from 1 through 11.

3. Press [ENTER].

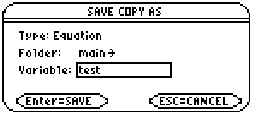
Because the number of equations that you can recall with [F5] **Eqns** is limited, a particular equation may not be retained indefinitely.

To store the current equation for future use, save it to a variable.

1. From the Numeric Solver screen, press [F1] and select **2:Save Copy As.**



2. Specify a folder and a variable name for the equation.
3. Press [ENTER] twice.

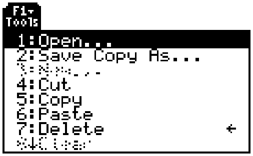


Note: An equation variable has an EXPR data type, as shown on the MEMORY and VAR-LINK screens.

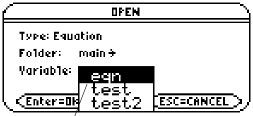
Opening a Saved Equation

To open a previously saved equation variable:

1. From the Numeric Solver screen, press [F1] and select **1:Open**.



2. Select the applicable folder and equation variable.
3. Press [ENTER].



Variable eqn contains the current equation; it always appears alphabetically in the list.

Defining the Known Variables

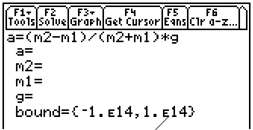
After you type an equation in the Numeric Solver, enter the applicable values for all variables except the unknown variable.

Defining the List of Variables

After typing your equation on the **eqn:** line, press [ENTER] or ⊖.

The screen lists the variables in the order they appear in the equation. If a variable is already defined, its value is shown. You can edit these variable values.

Note: If an existing variable is locked or archived, you cannot edit its value.

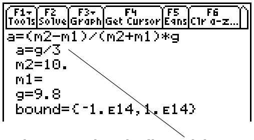


The solution must be within the specified bounds, which you can edit.

Enter a number or expression for all variables except the one you want to solve for.

Notes and Common Errors

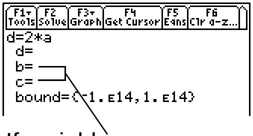
- If you define a variable:
  - In terms of another variable in the equation, that variable must be defined first.
  - In terms of another variable that is not in the equation, that variable must already have a value; it cannot be undefined.
  - As an expression, it is evaluated when you move the cursor off the line. The expression must evaluate to a real number



Since a is defined in terms of g, you must define g before a. When you move the cursor to another line, g/3 is evaluated.

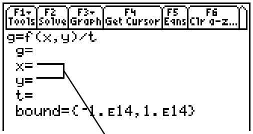
- If the equation contains a variable already defined in terms of other variables, those other variables are listed.

  Note: When you assign a value to a variable in the Numeric Solver, that variable is defined globally. It still exists after you leave the solver.



If variable a was defined previously as b+c→a, then b and c are listed instead of a.

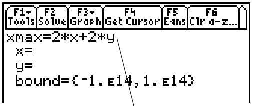
- If you refer to a previously defined function, any variables used as arguments in the function call are listed, not the variables used to define the function.



If f(a,b) was defined previously as √(a^2+b^2) and your equation contains f(x,y), then x and y are listed, not a and b.

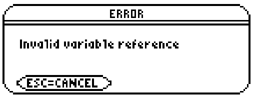
- If the equation contains a system variable (**xmin**, **xmax**, etc.), that variable is not listed. The solver uses the system variable's existing value.

  Note: You cannot solve for a system variable other than exp. Also, if the equation contains a system variable, you cannot use [F3] to graph.



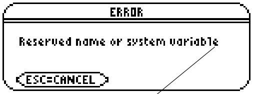
In the standard viewing window, xmax=10.

- Although you can use a system variable in the equation, an error occurs if you use [F3] to graph the solution.



- If you see the error shown to the right, delete the entered variable value. Then edit the equation to use a different variable.

  Note: This error occurs if you use a reserved name incorrectly or refer to an undefined system function as a simple variable without parentheses.



For example, y1(x) is undefined and you use y1.

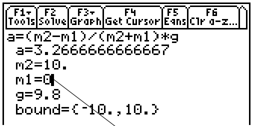
Editing the Equation

In the Numeric Solver, press ↑ until the cursor is on the equation. The screen automatically changes to show only the **eqn:** line. Make your changes, and then press [ENTER] or ↓ to return to the list of variables.

Specifying an Initial Guess and/or Bounds (Optional)

To find a solution more quickly or to find a particular solution (if multiple solutions exist), you can optionally:

- Enter an initial guess for the unknown variable. The guess must be within the specified bounds.
- Enter lower and upper bounds close to the solution.



Initial guess must be within the bounds.

For the bounds, you can also enter variables or expressions that evaluate to appropriate values (**bound={lower,upper}**) or a valid list variable that contains a two-element list (**bound=list**). The bounds must be two floating point elements with the first one less than or equal to the second one.

Note: You can also select an initial guess graphically.

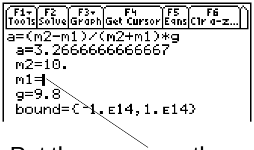
Solving for the Unknown Variable

After you type an equation in the Numeric Solver and enter values for the known variables, you are ready to solve for the unknown variable.

Finding the Solution

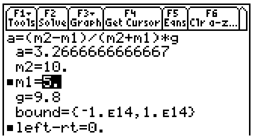
With all known variables defined:

1. Move the cursor to the unknown variable.



Put the cursor on the variable you want to solve for.

2. Press [F2] **Solve**.
3. A ▪ marks the solution and **left−rt.** The ▪ disappears when you edit a value, move the cursor to the equation, or leave the solver.



Note: To stop (break) a calculation, press [ON]. The unknown variable shows the value being tested when the break occurred.

Using the solution and your entered values, the left and right sides of the equation are evaluated separately. **left−rt** shows the difference, which indicates the solution's accuracy. The smaller the value, the more accurate the solution. If the solution is precise, **left−rt=0**.

If you:	**Do this:**
Want to solve for other values	Edit the equation or variable values.
Want to find a different solution for an equation with multiple solutions	Enter an initial guess and/or a new set of bounds close to the other solution.
See the message:  	Press [ESC]. The unknown variable shows the value being tested when the error occurred. • The **left−rt** value may be small enough for you to accept the result. • If not, enter a different set of bounds.

Note: An iterative process is used to solve an equation. If the iterative process cannot converge on a solution, this error occurs.

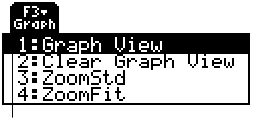
Graphing the Solution

You can graph an equation's solutions any time after defining the known variables, either before or after you solve for the unknown variable. By graphing the solutions, you can see how many solutions exist and use the cursor to select an accurate initial guess and bounds.

Displaying the Graph

In the Numeric Solver, leave the cursor on the unknown variable. Press [F3] and select:

1:Graph View
– or –
3:ZoomStd
– or –
4:ZoomFit



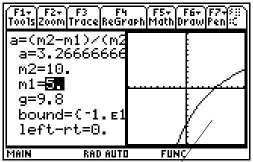
Graph View uses the current Window variable values.

For information about **ZoomStd** and **ZoomFit**, refer to *Basic Function Graphing*.

The graph is shown in a split screen, where:

- The unknown variable is plotted on the x axis.
- **left−rt** is plotted on the y axis.

Solutions for the equation exist at **left−rt=0**, where the graph crosses the x axis.



The current graph format settings are used.

Note: For more information, refer to the *Split Screens* module.

You can explore the graph by using the free-moving cursor, tracing, zooming, etc., as described in *Basic Function Graphing*.

How the Graph Affects Various Settings

When you use the Numeric Solver to display a graph:

- The following modes are changed automatically to these settings:

Mode	Setting
Graph	FUNCTION Any functions selected in the Y= Editor will not be graphed.
Split Screen	LEFT-RIGHT
Number of Graphs	1

  Note: If you were previously using different mode settings, you will need to reselect those settings manually.

- All stat plots are deselected.
- After you leave the Numeric Solver, the Graph screen may continue to display the equation's solution, ignoring any selected Y= functions. If so, display the Y= Editor and then return to the Graph screen. Also, the graph is reset when you change the Graph mode or use **ClrGraph** from the Home screen (F4 **5**) or a program.

Selecting a New Initial Guess from the Graph

To use the graph cursor to select an initial guess:

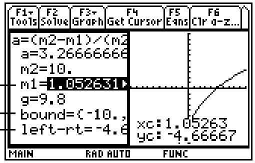
1. Move the cursor (either free-moving or trace) to the point that you want to use as the new guess.

2. Use [2nd] [⊞] to make the Numeric Solver screen active.

3. Make sure the cursor is on the unknown variable, and press [F4].

   Note: Cursor coordinate xc is the unknown variable value, and yc is the **left−rt** value.

4. Press [F2] to re-solve the equation.



[F4] sets the graph cursor's xc value as an initial guess and the yc value as left−rt. The graph's xmin and xmax values are set as the bounds.

Returning to a Full Screen

From the split screen:

- To display the Numeric Solver full screen, use [2nd] [⊞] to make the solver screen active, press [F3], and then select **2:Clear Graph View.**
  – or –
- To display the Home screen, press [2nd] [QUIT] twice.

Clearing Variables Before Leaving the Numeric Solver

When you solve an equation, its variables still exist after you leave the Numeric Solver. If the equation contains single-character variables, their values may inadvertently affect later symbolic calculations. Before leaving the Numeric Solver, you may want to:

1. Press:
   [2nd] [F6]
   to clear all single-character variables in the current folder.
2. Press [ENTER] to confirm the action.

   The screen returns to the solver's **eqn:** line.

Note: Any time you want to clear single-character variables listed in the solver, use:
[2nd] [F6].

Number Bases

Entering and Converting Number Bases

Regardless of the Base mode, you must always use the appropriate prefix when entering a binary or hexadecimal number.

Entering a Binary or Hexadecimal Number

To enter a binary number, use the form:

0b *binaryNumber* (for example: **0b11100110**)

└ Binary number with up to 32 digits
└ Zero, not the letter O, and the letter b

To enter a hexadecimal number, use the form:

0h *hexadecimalNumber* (for example: **0h89F2C**)

└ Hexadecimal number with up to 8 digits
└ Zero, not the letter O, and the letter h

Note: You can type the **b** or **h** in the prefix, as well as hex characters **A – F**, in uppercase or lowercase.

If you enter a number without the **0b** or **0h** prefix, such as 11, it is always treated as a decimal number. If you omit the **0h** prefix on a hexadecimal number containing **A – F**, all or part of the entry is treated as a variable.

Use the ▶ conversion operator

integerExpression ▶ **Bin**
integerExpression ▶ **Dec**
integerExpression ▶ **Hex**

For ▶, press [2nd] [▶]. Also, you can select base conversions from the MATH/Base menu.

For example, to convert 256 from decimal to binary:

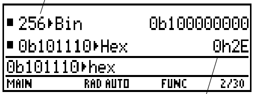
256 ▶ Bin

Note: If your entry is not an integer, a Domain error is displayed.

To convert 101110 from binary to hexadecimal:

0b101110 ▶ Hex

For a binary or hex entry, you must use the 0b or 0h prefix.



Results use the 0b or 0h prefix to identify.

Alternate Method for Conversions

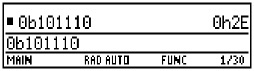
Instead of using ▶, you can:

1. Use [MODE] to set the **Base** mode to the base that you want to convert to.

If Base mode = BIN:

2. From the Home screen, type the number that you want to convert (using the correct prefix) and press [ENTER].

If Base mode = HEX:

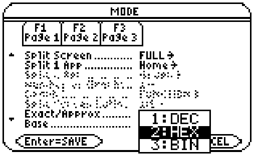


Performing Math Operations with Hex or Bin Numbers

For any operation that uses an integer number, you can enter a hexadecimal or binary number. Results are displayed according to the Base mode. However, results are restricted to certain size limits when Base = HEX or BIN.

Setting the Base Mode for Displayed Results

1. Press [MODE] [F2] to display **Page 2** of the **MODE** screen.
2. Scroll to the **Base** mode, press ⊳, and select the applicable setting.
3. Press [ENTER] to close the **MODE** screen.

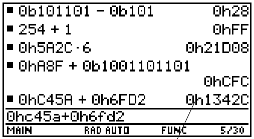


The **Base** mode controls the displayed format of integer results only.

Note: The Base mode affects output only. You must always use the **0h** or **0b** prefix to enter a hex or binary number.

Fractional and floating-point results are always shown in decimal form.

If Base mode = HEX:



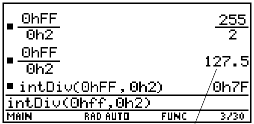
0h prefix in result identifies the base.

Dividing When Base = HEX or BIN

When Base=HEX or BIN, a division result is displayed in hexadecimal or binary form only if the result is an integer.

To ensure that division always produces an integer, use **intDiv( )** instead of [÷].

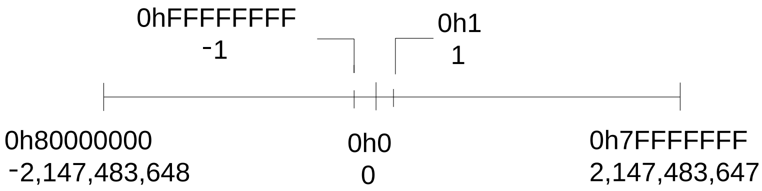
If Base mode = HEX:



Press [♦] [ENTER] to display the result in APPROXIMATE form.

Size Limitations When Base = HEX or BIN

When Base=HEX or BIN, an integer result is stored internally as a signed, 32-bit binary number, which uses the range (shown in hexadecimal and decimal):



If a result’s magnitude is too large to be stored in a signed, 32-bit binary form, a symmetric modulo operation brings the result into the range. Any number greater than 0h7FFFFFFF is affected. For example, 0h80000000 through 0hFFFFFFFF become negative numbers.

Comparing or Manipulating Bits

The following operators and functions let you compare or manipulate bits in a binary number. You can enter an integer in any number base. Your entries are converted to binary automatically for the bitwise operation, and results are displayed according to the Base mode.Boolean Operations

Operator with syntax	Description
not *integer*	Returns the one’s complement, where each bit is flipped.
(-) *integer*	Returns the two’s complement, which is the one’s complement + 1.

Operator with syntax	Description
integer1 **and** *integer2*	In a bit-by-bit **and** comparison, the result is 1 if both bits are 1; otherwise, the result is 0. The returned value represents the bit results.
integer1 **or** *integer2*	In a bit-by-bit **or** comparison, the result is 1 if either bit is 1; the result is 0 only if both bits are 0. The returned value represents the bit results.
integer1 **xor** *integer2*	In a bit-by-bit **xor** comparison, the result is 1 if either bit (but not both) is 1; the result is 0 if both bits are 0 or both bits are 1. The returned value represents the bit results.

Note: You can select these operators from the MATH/Base menu. For an example using each operator, refer to the *Technical Reference* module.

Suppose you enter:

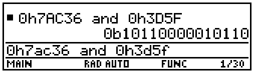
0h7AC36 **and** 0h3D5F

Internally, the hexadecimal integers are converted to a signed, 32-bit binary number.

Then corresponding bits are compared.

If Base mode = HEX:

If Base mode = BIN:



```
0h7AC36 =  0b00000000000001111010110000110110
  and        and
0h3D5F     0b00000000000000000011110101011111
           0b00000000000000000010110000010110 = 0h2C16
              └─Leading zeros are not shown in the result.
```

Note: If you enter an integer that is too large to be stored in a signed, 32-bit binary form, a symmetric modulo operation brings the value into the range.

The result is displayed according to the Base mode.

Rotating and Shifting Bits

Function with syntax	Description
rotate(*integer***)** – or – **rotate(***integer,#ofRotations***)**	If *#ofRotations* is: • omitted — bits rotate once to the right (default is ⁻1). • negative — bits rotate the specified number of times to the right. • positive — bits rotate the specified number of times to the left. In a right rotation, the rightmost bit rotates to the leftmost bit; vice versa for a left rotation.

Function with syntax	Description
shift(*integer***)** – or – **shift(***integer,#ofShifts***)**	If *#ofShifts* is: • omitted — bits shift once to the right (default is -1). • negative — bits shift the specified number of times to the right. • positive — bits shift the specified number of times to the left. In a right shift, the rightmost bit is dropped and 0 or 1 is inserted to match the leftmost bit. In a left shift, the leftmost bit is dropped and 0 is inserted as the rightmost bit.

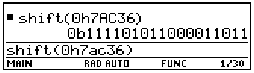
Suppose you enter:

shift(0h7AC36**)**

If Base mode = HEX:

Internally, the hexadecimal integer is converted to a signed, 32-bit binary number.

If Base mode = BIN:



Then the shift is applied to the binary number.

Each bit shifts to the right.

7AC36 = 0b00000000000001111010110000110110

Inserts 0 if leftmost bit is 0,
or 1 if leftmost bit is 1

Dropped

b00000000000000111101011000011011 = 0h3D61B

Leading zeros are not shown in the result.

The result is displayed according to the Base mode.

Note: If you enter an integer that is too large to be stored in a signed, 32-bit binary form, a symmetric modulo operation brings the value into the range.

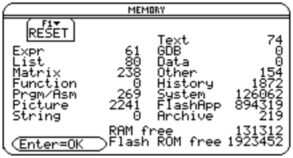
Memory and Variable Management

Checking and Resetting Memory

The **MEMORY** screen shows the amount of memory (in bytes) used by all variables in each data type, regardless of whether the variables are stored in RAM or the user data archive. You can also use this screen to reset the memory.

Displaying the MEMORY Screen

Press [2nd] [MEM]. (The numbers on your **MEMORY** screen may vary from those shown.)



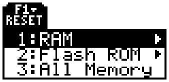
Prgm/Asn: Includes programs written for the TI-89 Titanium as well as any assembly-language programs you have loaded.
History: Size of history pairs saved in the Home screen's history area.
FlashApp: Size of Flash applications.
RAM free: Free space in RAM.
Flash ROM free: Free space in Flash ROM.

Note: To display the size of individual variables and determine if they are in the user data archive, use the **VAR-LINK** screen.

To close the screen, press [ENTER]. To reset the memory, use the following procedure.

From the **MEMORY** screen:

1. Press [F1].
2. Select the applicable item.



Item	Description
RAM	**1:All RAM:** Resetting RAM erases all data and programs from RAM. **2:Default:** Resets all system variables and modes to their original factory settings. This does not affect any user-defined variables, functions, or folders.
Flash ROM	**1:Archive:** Resetting Archive erases all data and programs from Flash ROM. **2:Flash Apps:** Resetting Flash Apps erases all Flash applications from Flash ROM. **3:Both:** Resetting both erases all data, programs, and Flash applications from Flash ROM.
All Memory	Resetting will delete all data, programs, and Flash applications from RAM and Flash ROM.

Important: To delete individual (instead of all) variables, use **VAR-LINK**.

3. When prompted for confirmation, press [ENTER].

   The TI-89 Titanium displays a message when the reset is complete.

   Note: To cancel the reset, press [ESC] instead of [ENTER].

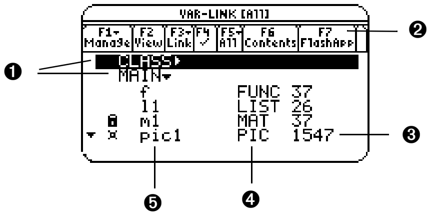
4. Press [ENTER] to acknowledge the message.

Displaying the VAR-LINK Screen

The **VAR-LINK** screen lists the variables and folders that are currently defined. After displaying the screen, you can manipulate the variables and/or folders.

Displaying the VAR-LINK Screen

Press [2nd] [VAR-LINK]. By default, the **VAR-LINK** screen lists all user-defined variables in all folders and with all data types.



❶ Folder names (alphabetically listed)
❷ Shows installed Flash applications
❸ Size in bytes
❹ Data type
❺ Variable names (alphabetically listed)

This...	Indicates this...
▸	Collapsed folder view (to right of folder name).
▾	Expanded folder view (to right of folder name).

This...	Indicates this...
▼	You can scroll for more variables and/or folders (in bottom left corner of screen).
✓	If selected with [F4].
🔒	Locked
¤	Archived

To scroll through the list:

- Press ⊖ or ⊕. (Use [2nd] ⊖ or [2nd] ⊕ to scroll one page at a time.)

  – or –

- Type a letter. If there are any variable names that start with that letter, the cursor moves to highlight the first of those variable names.

Note: Type a letter repeatedly to cycle through the names that start with that letter.

Variable Types as Listed on VAR-LINK

Type	Description
ASM	Assembly-language program
DATA	Data
EXPR	Expression (includes numeric values)
FUNC	Function
GDB	Graph database
LIST	List
MAT	Matrix
PIC	Picture of a graph
PRGM	Program
STR	String
TEXT	Text Editor session

Types not listed above are miscellaneous data types used by software applications.

Closing the VAR-LINK Screen

To close the **VAR-LINK** screen and return to the current application, use [ENTER] or [ESC] as described below.

Press:	To:
[ENTER]	Paste the highlighted variable or folder name to the cursor location in the current application.

Press:	To:
ESC	Return to the current application without pasting the highlighted name.

Displaying Information about Variables on the Home Screen

From the Home screen, you can display information about variables without opening the VAR-LINK screen.

- To determine if a variable with a given name exists in the system table, Enter the **IsVar()** function on the Home screen.

  IsVar (*var_name*)

  IsVar is a function, which requires you to enclose the variable name in parentheses.

- To determine if a variable is archived, use the **IsArchiv()** function.

  IsArchiv *(var_name)*

- To determine if a variable is locked, use the **IsLocked()** function.

  IsLocked *(var_name)*

Manipulating Variables and Folders with VAR-LINK

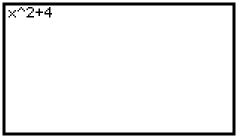
On the **VAR-LINK** screen, you can show the contents of a variable. You can also select one or more listed items and manipulate them by using the operations in this section.

Showing the Contents of a Variable

You can show all variable types except **ASM**, **DATA**, **GDB, and variables created by Flash Apps.** For example, you must open a **DATA** variable in the Data/Matrix Editor.

1. On **VAR-LINK**, move the cursor to highlight the variable.

2. Press:
   [2nd] [F6]

   If you highlight a folder, the screen shows the number of variables in that folder.

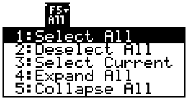
3. To return to **VAR-LINK**, press any key.

Note: You cannot edit the contents from this screen.

Selecting Items from the List

For other operations, select one or more variables and/or folders.

To select:	Do this:
A single variable or folder	Move the cursor to highlight the item, then press [F4].
A group of variables or folders	Highlight each item and press [F4]. A ✓ is displayed to the left of each selected item. (If you select a folder, all variables in that folder are selected.) Use [F4] to select or deselect an item.

To select:	Do this:
All folders and all variables  	Press ⓡ to expand the folder, then press [F5] **All** and select **1:Select All**. Choosing **3:Select Current** selects the last set of items transmitted to your unit during the current **VAR-LINK** session. Choosing **4:Expand All** or **5:Collapse All** expands or collapses your folders or Flash applications.

Note: Press either ⓛ or ⓡ to toggle between expanded or collapsed view when you have a folder highlighted.

Folders and Variables

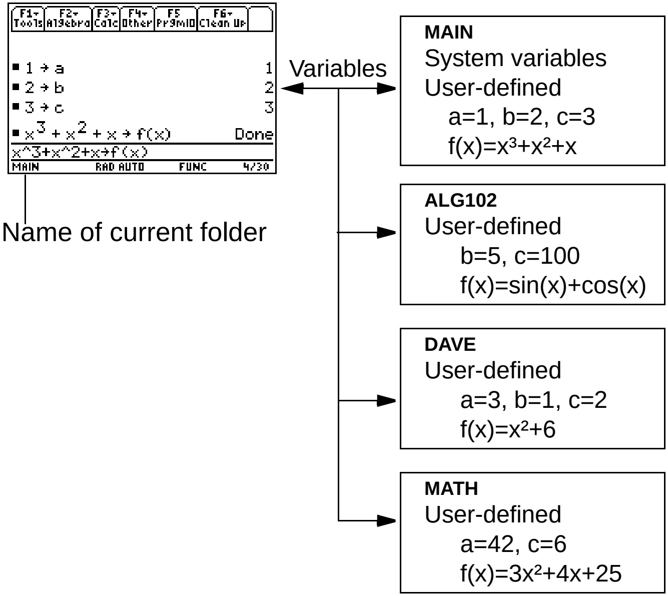
Folders give you a convenient way to manage variables by organizing them into related groups.

The TI-89 Titanium has one built-in folder named **MAIN.** Unless you create other folders and designate a user-created folder as the current folder, all variables are stored in the **MAIN** folder by default. A system variable or a variable with a reserved name can be stored in the **MAIN** folder only.

Example of variables that can be stored in MAIN only
Window variables (**xmin**, **xmax**, etc.) Table setup variables (**TblStart**, **∆Tbl**, etc.) Y= Editor functions (**y1(x)**, etc.)

By creating additional folders, you can store independent sets of user-defined variables (including user-defined functions). For example, you can create separate folders for different TI-89 Titanium applications (Math, Text Editor, etc.) or classes. You can store a user-defined variable in any existing folder.

The user-defined variables in one folder are independent of the variables in any other folder. Therefore, folders can store separate sets of variables with the same names but different values.



You cannot create a folder within another folder.

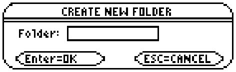
The system variables in the **MAIN** folder are always directly accessible, regardless of the current folder.

Note: User-defined variables are stored in the “current folder” unless you specify otherwise.

Creating a Folder from the VAR-LINK Screen

1. Press [2nd] [VAR-LINK].

2. Press [F1] **Manage** and select **5:Create Folder.**



3. Type a unique folder name up to eight characters, and press [ENTER] twice.

After you create a new folder from **VAR-LINK**, that folder is not automatically set as the current folder.

Creating a Folder from the Home Screen

Enter the **NewFold** command on the Home screen.

NewFold *folderName*

Folder name to create. This new folder is set automatically as the current folder.

Setting the Current Folder from the Home Screen

Enter the **setFold** function on the Home screen.

setFold (*folderName*)

setFold is a function, which requires you to enclose the folder name in parentheses.

When you execute **setFold**, it returns the name of the folder that was previously set as the current folder.

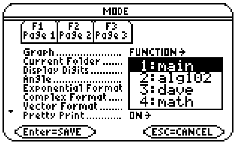
Setting the Current Folder from the MODE Dialog Box

1. Press [MODE].

2. Highlight the **Current Folder** setting.

3. Press ► to display a menu of existing folders.

   Note: To cancel the menu or exit the dialog box without saving any changes, press [ESC].



4. Select the applicable folder. Either:

   - Highlight the folder name and press [ENTER].

     – or –

   - Press the corresponding number or letter for that folder.

5. Press [ENTER] to save your changes and close the dialog box.

Renaming Variables or Folders

Remember, if you use [F4] to select a folder, the variables in that folder are selected automatically. As necessary, use [F4] to deselect individual variables.

1. On **VAR-LINK**, select the variables and/or folders.
2. Press [F1] **Manage** and select **3:Rename.**
3. Type a unique name, and press [ENTER] twice.

   If you selected multiple items, you are prompted to enter a new name for each one.

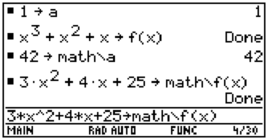
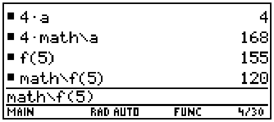
Using Variables in Different Folders

You can access a user-defined variable or function that is not in the current folder. Specify the complete pathname instead of only the variable name.

A pathname has the form:

folderName \ *variableName*
– or –
folderName \ *functionName*

For example:

<table>
<tr><th>If Current Folder = MAIN</th><th>Folders and Variables</th></tr>
<tr><td>
</td><td>MAIN
a=1
f(x)=x^3+x^2+x</td></tr>
<tr><td>
</td><td>MATH
a=42
f(x)=$3x^2+4x+25$</td></tr>
</table>

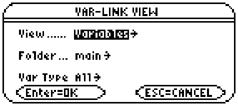
To see a list of existing folders and variables, press [2nd] [VAR-LINK]. On the **VAR-LINK** screen, you can highlight a variable and press [ENTER] to paste that variable name to the open application's entry line. If you paste a variable name that is not in the current folder, the pathname (*folderName\variableName*) is pasted.

Listing Only a Specified Folder and/or Variable Type, or Flash application

If you have a lot of variables, folders, or Flash applications, it may be difficult to locate a particular variable. By changing **VAR-LINK's** view, you can specify the information you want to see.

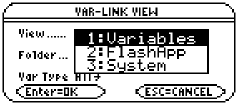
From the **VAR-LINK** screen:

1. Press [F2] **View.**



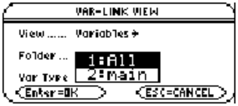
2. Highlight the setting you want to change, and press ▶. This displays a menu of valid choices. (To cancel a menu, press [ESC].)

   View — Allows you to choose variables, Flash applications, or system variables to view.



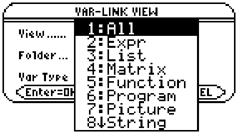
   Note: To list system variables (window variables, etc.), select **3:System.**

   Folder — Always lists **1:All** and **2:main**, but lists other folders only if you have created them.



   Var Type — Lists the valid variable types.

   ↓ — indicates that you can scroll for additional variable types.



3. Select the new setting.

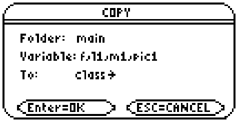
4. When you are back on the **VAR-LINK VIEW** screen, press [ENTER].

The **VAR-LINK** screen is updated to show only the specified folder, variable type, or Flash application.

Copying or Moving Variables from One Folder to Another

You must have at least one folder other than **MAIN**. You cannot use **VAR-LINK** to copy variables within the same folder.

1. On **VAR-LINK**, select the variables.
2. Press [F1] **Manage** and select **2:Copy** or **4:Move**.
3. Select the destination folder.



4. Press [ENTER]. The copied or moved variables retain their original names.

   Note: To copy a variable to a different name in the same folder, use [STO▸] (such as a1→a2) or the CopyVar command from the Home screen.

Locking or Unlocking Variables Folders, or Flash Applications

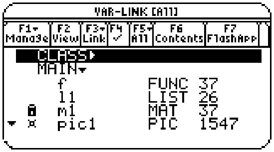
When a variable is locked, you cannot delete, rename, or store to it. However, you can copy, move, or display its contents. When a folder is locked, you can manipulate the variables in the folder (assuming the variables are not locked), but you cannot delete the folder. When a Flash application is locked, you cannot delete it.

1. On **VAR-LINK**, select the variables, folders, or Flash application.

2. Press [F1] **Manage** and select **6:Lock** or **7:UnLock.**

🔒 indicates a locked variable or folder in RAM.

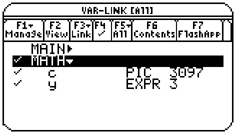
¤ indicates an archived variable, which is locked automatically.



Deleting a Folder from the VAR-LINK Screen

When you delete a folder from the **VAR-LINK** screen, all of the variables in that folder are also deleted. You cannot delete the **MAIN** folder.

1. Press [2nd] [VAR-LINK].
2. Press [F4] to select the folder(s) to delete. (The folder's variables become selected automatically.)



3. Press [F1] **1:Delete** or [←].

4. Press [ENTER] to confirm the deletion of the folder and all its variables.

Deleting a Variable or a Folder from the Home Screen

Before deleting a folder from the Home screen, you must first delete all the variables stored in that folder.

- To delete a variable, enter the **DelVar** command on the calculator Home screen.

  DelVar *var1* [, *var2*] [, *var3*] ...

- To delete all variables of a specific type, enter the **DelType** command on the calculator Home screen.

  DelType *var_type* where *var_type* is the variable type.

  Note: The **DelType** command deletes all variables of the specified type in all folders.

- To delete an empty folder, enter the **DelFold** command on the calculator Home screen.

  DelFold *folder1* [, *folder2*] [, *folder3*] ...

  Note: You cannot delete the **MAIN** folder.

Pasting a Variable Name to an Application

Suppose you are typing an expression on the Home screen and can't remember which variable to use. You can display the **VAR-LINK** screen, select a variable from the list, and paste that variable name directly onto the Home screen's entry line.

Which Applications Can You Use?

From the following applications, you can paste a variable name to the current cursor location.

- Home screen, Y= Editor, Table Editor, or Data/Matrix Editor — The cursor must be on the entry line.
- Text Editor, Window Editor, Numeric Solver, or Program Editor — The cursor can be anywhere on the screen.

You can also paste a variable name to the current cursor location in many Flash applications.

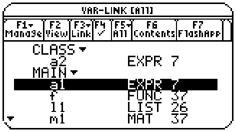
Procedure

Starting from an application listed above:

1. Position the cursor where you want to insert the variable name.

   sin(|

2. Press 2nd [VAR-LINK].

3. Highlight the applicable variable.

   Note: You can also highlight and paste folder names.

4. Press ENTER to paste the variable name.

   sin(a1|

   Note: This pastes the variable's name, not its contents. Use 2nd [RCL], instead of 2nd [VAR-LINK], to recall a variable's contents.

5. Finish typing the expression.

   sin(a1)|

If you paste a variable name that is not in the current folder, the variable's pathname is pasted.

```
sin(class\a2
```

Assuming that CLASS is *not* the current folder, this is pasted if you highlight the a2 variable in CLASS.

Archiving and Unarchiving a Variable

To archive or unarchive one or more variables interactively, use the **VAR-LINK** screen. You can also perform these operations from the Home screen or a program.

Why Would You Want to Archive a Variable?

The user data archive lets you:

- Store data, programs, or any other variables to a safe location where they cannot be edited or deleted inadvertently.
- Create additional free RAM by archiving variables. For example:
  - You can archive variables that you need to access but do not need to edit or change, or variables that you are not using currently but need to retain for future use.

    Note: You cannot archive variables with reserved names or system variables.
  - If you acquire additional programs for your TI-89 Titanium, particularly if they are large, you may need to create additional free RAM before you can install those programs.

Additional free RAM can improve performance times for certain types of calculations.

From the VAR-LINK Screen

To archive or unarchive:

1. Press [2nd] [VAR-LINK] to display the **VAR-LINK** screen.
2. Select one or more variables, which can be in different folders. (You can select an entire folder by selecting the folder name.)

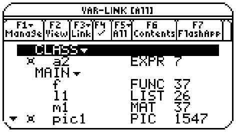
   Note: To select a single variable, highlight it. To select multiple variables, highlight each variable and press [F4] ✓.

3. Press [F1] and select either:

   8:Archive Variable
   – or –
   9:Unarchive Variable

If you select **8:Archive Variable**, the variables are moved to the user data archive.

¤ = archived variables



You can access an archived variable just as you would any locked variable. For all purposes, an archived variable is still in its original folder; it is simply stored in the user data archive instead of RAM.

Note: An archived variable is locked automatically. You can access the variable, but you cannot edit or delete it.

From the Home Screen or a Program

Use the **Archive** and **Unarchiv** commands:

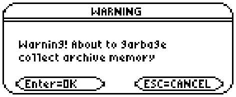
> **Archive** *variable1*, *variable2*, …
>
> **Unarchiv** *variable1*, *variable2*, …

If a Garbage Collection Message Is Displayed

If you use the user data archive extensively, you may see a Garbage Collection message. This occurs if you try to archive a variable when there is not enough free archive memory. However, the TI-89 Titanium will attempt to rearrange the archived variables to make additional room.

Responding to the Garbage Collection Message

When you see the message to the right:



- To continue archiving, press [ENTER].

  – or –

- To cancel, press [ESC].

After garbage collection, depending on how much additional space is freed, the variable may or may not be archived. If not, you can unarchive some variables and try again.

Why not Perform Garbage Collection Automatically, without a Message?

The message:

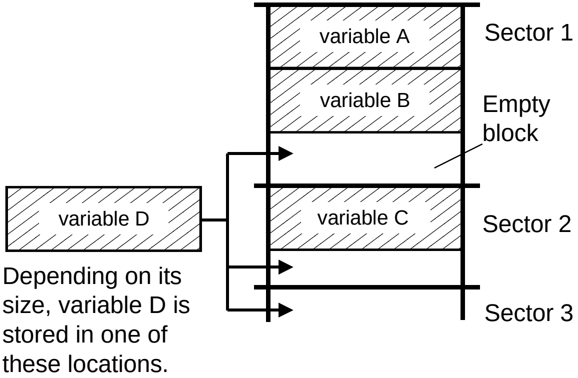
- Lets you know why an archive will take longer than usual. It also alerts you that the archive may fail if there is not enough memory.
- Can alert you when a program is caught in a loop that repetitively fills the user data archive. Cancel the archive and investigate the reason.

Why Is Garbage Collection Necessary?

The user data archive is divided into sectors. When you first begin archiving, variables are stored consecutively in sector 1. This continues to the end of the sector. If there is not enough space left in the sector, the next variable is stored at the beginning of the next sector. Typically, this leaves an empty block at the end of the previous sector.

Each variable that you archive is stored in the first empty block large enough to hold it.

Note: An archived variable is stored in a continuous block within a single sector; it cannot cross a sector boundary.

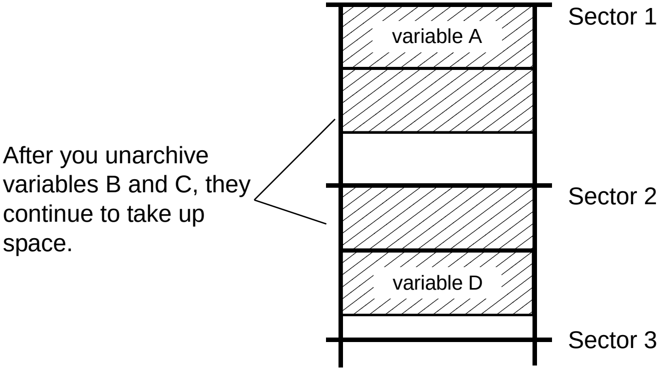


This process continues to the end of the last sector. Depending on the size of individual variables, the empty blocks may account for a significant amount of space.

Note: Garbage collection occurs when the variable you are archiving is larger than any empty block.

How Unarchiving a Variable Affects the Process

When you unarchive a variable, it is copied to RAM but it is not actually deleted from user data archive memory.

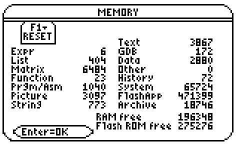


Unarchived variables are “marked for deletion,” meaning they will be deleted during the next garbage collection.

If the MEMORY Screen Shows Enough Free Space

Even if the **MEMORY** screen shows enough free space to archive a variable, you may still get a Garbage Collection message.

This TI-89 Titanium memory screen shows free space that will be available after all “marked for deletion” variables are deleted.

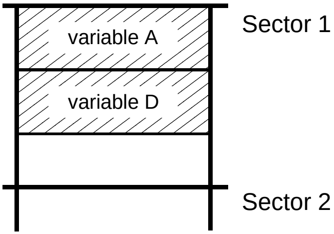


When you unarchive a variable, the Flash ROM free amount increases immediately, but the space is not actually available until after the next garbage collection.

The Garbage Collection Process

The garbage collection process:

- Deletes unarchived variables from the user data archive.
- Rearranges the remaining variables into consecutive blocks.



Memory Error When Accessing an Archived Variable

An archived variable is treated the same as a locked variable. You can access the variable, but you cannot edit or delete it. In some cases, however, you may get a **Memory Error** when you try to access an archived variable.

What Causes the Memory Error?

The **Memory Error** message is displayed if there is not enough free RAM to access the archived variable. This may cause you to ask, “If the variable is in the user data archive, why does it matter how much RAM is available?” The answer is that the following operations can be performed only if a variable is in RAM.

- Opening a text variable in the Text Editor.
- Opening a data variable, list, or matrix in the Data/Matrix Editor.
- Opening a program or function in the Program Editor.
- Running a program or referring to a function.

Note: A temporary copy lets you open or execute an archived variable. However, you cannot save any changes to the variable.

So that you don't have to unarchive variables unnecessarily, the TI-89 Titanium performs a "behind-the scenes" copy. For example, if you run a program that is in the user data archive, the TI-89 Titanium:

1. Copies the program to RAM.
2. Runs the program.
3. Deletes the copy from RAM when the program is finished.

The error message is displayed if there is not enough free RAM for the temporary copy.

Note: Except for programs and functions, referring to an archived variable does not copy it. If variable ab is archived, it is not copied if you perform **6∗ab**.

Correcting the Error

To free up enough RAM to access the variable:

1. Use the **VAR-LINK** screen (2nd [VAR-LINK]) to determine the size of the archived variable that you want to access.
2. Use the **MEMORY** screen (2nd [MEM]) to check the RAM free size.

3. Free up the needed amount of memory by:
    - Deleting unnecessary variables from RAM.
    - Archiving large variables or programs (moving them from RAM to the user data archive).

    Note: Typically, the RAM free size must be larger than the archived variable.

Connectivity

Connecting Two Units

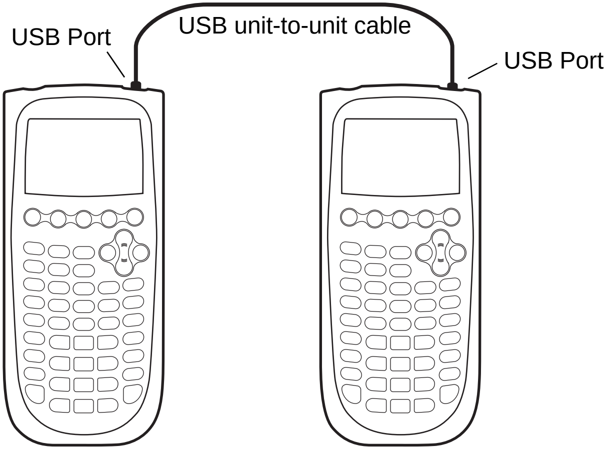
The TI-89 Titanium comes with a cable that lets you connect two units. Once connected, you can transmit information between two units. A USB unit-to-unit cable is included with the TI-89 Titanium; use the calculator’s USB port with this cable.

Note: The TI-89 Titanium features both a USB port and an I/O port, so you can connect TI graphing calculators with either type of link port. However, using the I/O port requires the I/O unit-to-unit cable (sold separately) or the USB Silver Edition cable (also sold separately), which is used to connect to a computer.

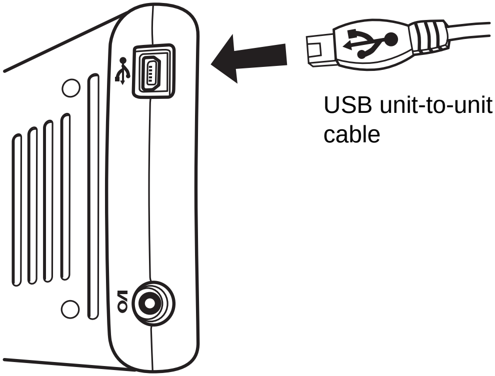
Connecting before Sending or Receiving

Using firm pressure, insert one end of the cable into the link port of each unit. Either unit can send or receive, depending on how you set them up from the **VAR-LINK** screen.

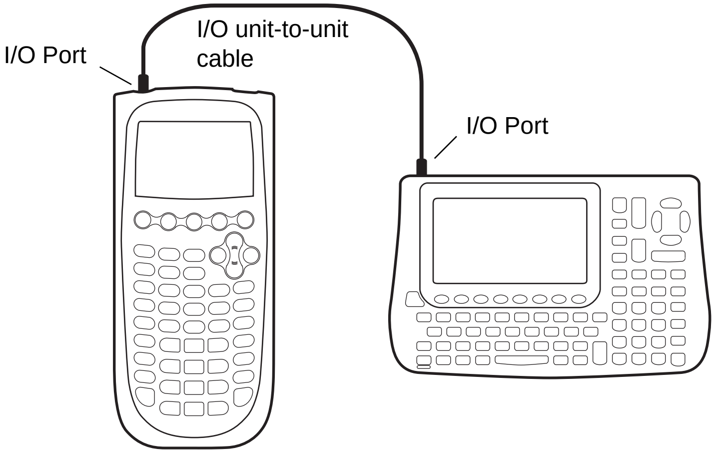
You can link a TI-89 Titanium or Voyage™ 200 to another TI-89 Titanium, Voyage™ 200, TI-89, or TI-92 Plus.



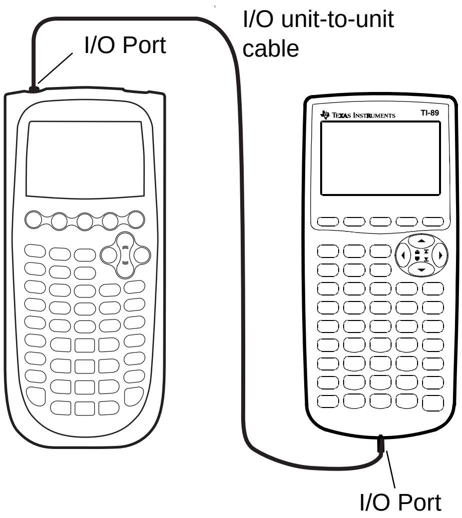
Two TI-89 Titanium calculators linked together



Position so that the USB symbols face each other; then insert the connector.



A TI-89 Titanium and a Voyage™ 200 linked together



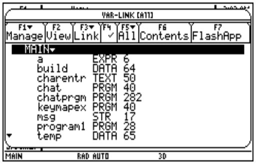
A TI-89 Titanium and a TI-89 linked together

Transmitting Variables, Flash Applications, and Folders

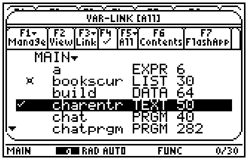
Transmitting variables is a convenient way to share any variable listed on the **VAR-LINK** screen — functions, programs, etc. You can also transmit Flash applications (Apps) and folders.

Flash applications will transfer only between certain units. For example, you can transfer an App from a TI-89 Titanium to another TI-89 Titanium, or from a TI-89 Titanium to a TI-89.

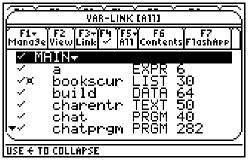
1. Connect two graphing calculators using the appropriate cable.
2. On the *sending* unit, press [2nd] [VAR-LINK] to display the **VAR-LINK** screen.



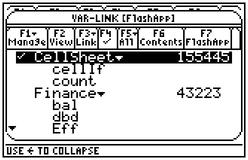
3. On the *sending* unit, select the variables, folders, or Flash applications you want to send.
   - To select a single variable, Flash application, or folder, move the cursor to highlight it and press [F4] to place a checkmark (✓) beside it.



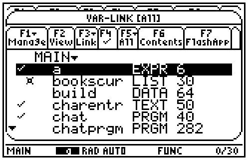
     - If on the default **VAR-LINK** screen, this selects the folder and its contents. Collapsed folders become expanded when selected.



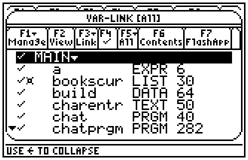
- If selecting a Flash App (from the F7 tab), this selects the App folder and its contents. A checkmark appears beside the folder, but not beside the contents. Collapsed Flash App folders do not automatically become expanded.



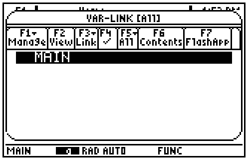
- To select multiple variables, Flash applications, or folders, highlight each one and press [F4] to place a checkmark (✓) beside it. Use [F4] again to deselect any that you do not want to transmit.



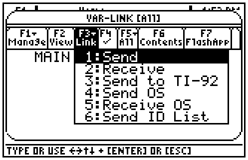
- To select all variables, Flash applications, or folders use [F5] **All 1:Select All**.



4. On the *receiving* unit, press [2nd] [VAR-LINK] to display the **VAR-LINK** screen. (The sending unit remains on the **VAR-LINK** screen.)



5. On both the *receiving* and the sending unit, press [F3] **Link** to display the menu options.



6. On the *receiving* unit, select **2:Receive**.

   The message **VAR-LINK: WAITING TO RECEIVE** and the **BUSY** indicator are displayed in the status line of the receiving unit.

7. On the *sending* unit, select **1:Send**

   This starts the transmission.

During transmission, a progress bar is displayed in the status line of the receiving unit. When transmission is complete, the **VAR-LINK** screen is updated on the receiving unit.

Note: Before transferring a purchased App, the receiving unit must have the appropriate certificate, if required. A certificate is a file that is generated by TI. Free and concept Apps do not require a certificate.

Rules for Transmitting Variables, Flash Applications, or Folders

Unlocked and unarchived variables that have the same name on both the sending and receiving units will be overwritten from the sending unit.

Locked variables that have the same name on both the sending and receiving units must be unlocked on the receiving unit before they can be overwritten from the sending unit. If archived variables have the same names on both the sending and receiving units, a message asks you to confirm that you will allow the variables to be overwritten.

If you select:	What happens:
Unlocked variable	The variable is transmitted to the current folder and it remains unlocked on the receiving unit.
Locked variable	The variable is transmitted to the current folder and it remains locked on the receiving unit.
Archived variable	The variable is transmitted to the current folder and it remains archived on the receiving unit.
Unlocked Flash application	If the receiving unit has the correct certification, the Flash application is transmitted. It remains unlocked on the receiving unit.
Locked Flash application	If the receiving unit has the correct certification, the Flash application is transmitted. It remains locked on the receiving unit.
Unlocked Folder	The folder and its selected contents are transmitted. The folder remains unlocked on the receiving unit.
Locked Folder	The folder and its selected contents are transmitted. The folder becomes unlocked on the receiving unit.

Canceling a Transmission

From either the sending or receiving unit:

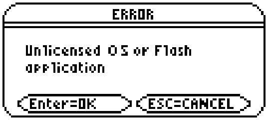
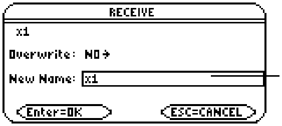
1. Press ON.

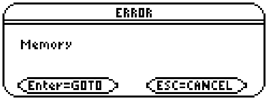
   An error message is displayed.

2. Press ESC or ENTER.

Common Error and Notification Messages

Shown on:	Message and Description:
Sending unit	 This is displayed after several seconds if: • A cable is not attached to the sending unit's link port. – or – • A receiving unit is not attached to the other end of the cable. – or – • The receiving unit is not set up to receive. Press ESC or ENTER to cancel the transmission. **Note:** The sending unit may not always display this message. Instead, it may remain **BUSY** until you cancel the transmission.

Shown on:	Message and Description:
Sending unit	 The receiving unit does not have the correct certification for the operating system (OS) or Flash application being sent.
Receiving unit	 New Name is active only if you change Overwrite to NO. The receiving unit has a variable with the same name as the specified variable being sent. • To overwrite the existing variable, press [ENTER]. (By default, **Overwrite = YES**.) • To store the variable to a different name, set **Overwrite = NO**. In the **New Name** input box, type a variable name that does not exist in the receiving unit. Then press [ENTER] twice. • To skip this variable and continue with the next one, set **Overwrite = SKIP** and press [ENTER]. • To cancel the transmission, press [ESC].

Shown on:	Message and Description:
Receiving unit	 The receiving unit does not have enough memory for what is being sent. Press [ESC] or [ENTER] to cancel the transmission.

Deleting Variables, Flash Applications, or Folders

1. Press [2nd] [VAR-LINK] to display the **VAR-LINK** screen.
2. Select the variables, folders, or Flash applications to delete.
   - To select a single variable, Flash application, or folder, move the cursor to highlight it and press [F4] to place a checkmark (✓) beside it.
     - If on the default **VAR-LINK** screen, this selects the folder and its contents. Collapsed folders become expanded when selected.
     - If selecting a Flash App (from the F7 tab), this selects the App folder and its contents. A checkmark appears beside the folder, but not beside the contents. Collapsed Flash App folders do not automatically become expanded.

     Note: You cannot delete the **Main** folder.
   - To select multiple variables, Flash applications, or folders highlight each one and press [F4] to place a checkmark (✓) beside it. Use [F4] again to deselect any that you do not want to transmit.
   - To select all variables, Flash applications, or folders use [F5] **All 1:Select All.**

3. Press F1 and choose **1:Delete**.
   – or –
   Press ←. A confirmation message appears.
4. Press ENTER to confirm the deletion.

Where to Get Flash Applications (Apps)

For up-to-date information about available Flash applications, check the Texas Instruments Web site at education.ti.com.

Many Apps no longer require a certificate. If you try to transfer an App from one unit to another and receive an **Unlicensed OS or Flash application** message, try downloading the App again from the Texas Instruments Web site at education.ti.com.

You can download a Flash application and/or certificate from the Texas Instruments Web site to a computer, and use a TI Connectivity Cable USB to install the application or certificate on your TI-89 Titanium.

For Flash App installation instructions, see education.ti.com/guides.

Transmitting Variables under Program Control

You can use a program containing **GetCalc** and **SendCalc** to transmit a variable from one device to another.

SendCalc sends a variable to the link port, where a linked device can receive the variable. The linked device must be on the Home screen or must execute **GetCalc** from a program.

You can use optional parameters with the SendCalc or GetCalc command to specify either the USB port or I/O port. (See Appendix A for details.) If you do not include these parameters, the TI-89 Titanium communicates through the USB port.

The “Chat” Program

The following program uses **GetCalc** and **SendCalc.** The program sets up two loops that let the linked devices take turns sending and receiving/displaying a variable named **msg.** **InputStr** lets each user enter a message in the **msg** variable.

```
:Chat()
:Prgm
:ClrIO
:Disp "On first unit to send,","
 enter 1;","On first to receive,"
:InputStr " enter 0",msg
:If msg="0" Then
:   While true
:      GetCalc msg
:      Disp msg
:      InputStr msg
:      SendCalc msg
:   EndWhile
:Else
:   While true
:     InputStr msg
:     SendCalc msg
:     GetCalc msg
:     Disp msg
:  EndWhile
:EndIf
:EndPrgm
```

Notes:

❶ Sets up this unit to receive and display the variable msg.

❷ Then lets this user enter a message in msg and send it.

❸ Loop executed by the unit that receives the first message.

❹ Lets this user enter a message in msg and send it.

❺ Then sets up this unit to receive and display msg.

❻ Loop executed by the unit that sends the first message.

To synchronize **GetCalc** and **SendCalc**, the loops are arranged so that the receiving unit executes **GetCalc** while the sending unit is waiting for the user to enter a message.

Running the Program

This procedure assumes that:

- The two devices are linked with the connecting cable.
- The Chat program is loaded on both devices.
  - Use each device's Program Editor to enter the program.
    – or –
  - Enter the program on one device and then use **VAR-LINK** to transmit the program variable to the other device.

To run the program on both devices:

1. On the Home screen of each device, enter **chat( ).**
2. When each device displays its initial prompt, respond as shown below.

On the:	Type:
Device that will send the first message.	**1** and press [ENTER].
Device that will receive the first message.	**0** and press [ENTER].

3. Take turns typing a message and pressing [ENTER] to send the variable **msg** to the other device.

Stopping the Program

Because the **Chat** program sets up an infinite loop on both devices, press [ON] (on both devices) to break the program. If you press [ESC] to acknowledge the error message, the program stops on the Program I/O screen. Press [F5] or [ESC] to return to the Home screen.

Upgrading the Operating System (OS)

You can upgrade the OS on your TI-89 Titanium using your computer. You can also transfer the OS from one unit to another identical model (for example, from a TI-89 Titanium to a TI-89 Titanium or from a Voyage™ 200 to a Voyage™ 200).

Installing OS software resets all device memory to the original factory settings. This means that all user-defined variables (in both RAM and the user data archive), functions, programs, lists, and folders (except the Main folder) will be deleted. It is possible that Flash applications could also be deleted. You should use TI Connect software to back up your data to your computer before installing a new OS on your calculator.

See the important information concerning batteries before performing an OS upgrade.

Important Operating System Download Information

New batteries should be installed before beginning an OS download.

If you are operating your TI-89 Titanium in a language other than English, you should ensure that you have the most current localizer application installed when you upgrade the OS software. If you do not have the most current localizer installed, prompts, error messages and status information related to new functionality in the OS may not display correctly.

When in OS download mode, the Automatic Power Down™ (APD™) feature does not function. If you leave your device in download mode for an extended time before you actually start the downloading process, your batteries may become depleted. You will then need to replace the depleted batteries with new batteries before downloading.

If you accidentally interrupt the transfer before it is complete, you will need to reinstall the OS. Again, remember to install new batteries before downloading.

Backing Up Your Unit Before an Operating System Installation

When you install an OS upgrade, the installation process:

- Deletes all user-defined variables (in both RAM and the user data archive), functions, programs, and folders.
- Could delete all Flash applications.
- Resets all system variables and modes to their original factory settings. This is equivalent to using the **MEMORY** screen to reset all memory.

To retain any existing variables or Flash applications, do the following before installing the upgrade:

- **Important:** Install new batteries.
- Transmit the variables or Flash applications to another device.
  – or –

- Use a USB cable or TI Connectivity Cable USB and TI Connect™ software (education.ti.com/downloadticonnect) to send the variables and/or Flash applications to a computer.

Where to Get Operating System Upgrades

For up-to-date information about available OS upgrades, check the Texas Instruments Web site at education.ti.com/downloadticonnect.

You can download an OS upgrade, Localizer or Flash application from the Texas Instruments Web site to a computer, and use a USB computer cable to install the OS or application on your TI-89 Titanium.

For complete information, refer to the instructions on the web.

Transferring the Operating System

OS software will transfer only from a TI-89 Titanium to a TI-89 Titanium, TI-89 to a TI-89, from a Voyage™ 200 to a Voyage™ 200, or from a TI-92 Plus to a TI-92 Plus.

To transfer the Operating System (OS) from unit to unit:

1. Link two like units together, for example, a TI-89 Titanium to a TI-89 Titanium; or a Voyage™ 200 to a Voyage™ 200.
2. On the receiving and the sending unit, press [2nd] [VAR-LINK] to display the **VAR-LINK** screen.
3. On the receiving and the sending unit, press [F3] **Link** to display the menu options.

4. On the receiving unit, select **5:Receive OS**.

   A warning message displays. Press [ESC] to halt the process, or press [ENTER] to proceed. Pressing [ENTER], displays **VAR-LINK: WAITING TO RECEIVE** and **BUSY** in the status line of the receiving unit.

5. On the sending unit, select **4:Send OS**.

   A warning message displays. Press [ESC] to halt the process, or press [ENTER] to start the transmission.

Important:

- For each receiving unit, remember to back up information as necessary and install new batteries.
- Be sure both the sending and receiving units are in the **VAR-LINK** screen.

During the transfer, the receiving unit shows how the transfer is progressing. When the transfer is complete:

- The sending unit returns to the **VAR-LINK** screen.
- The receiving unit returns to either the Apps desktop or the Home screen. You may need to use [♦] [-] (lighten) or [♦] [+] (darken) to adjust the contrast.

Do Not Attempt to Cancel an Operating System Transfer

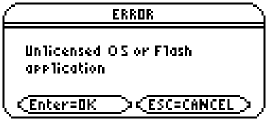
After the transfer starts, the receiving unit’s existing OS is effectively deleted. If you interrupt the transfer before it is complete, the receiving unit will not operate properly. You will then need to reinstall the OS upgrade.

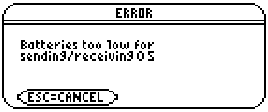
If You are Upgrading the Operating System on Multiple Units

To perform an OS upgrade on multiple units, download and install the OS into one unit and then transfer the OS upgrade from one unit to another. This method is faster than installing it on each unit via a computer. OS upgrades are released free of charge and you do not need to obtain a certificate before you download or install them.

Error Messages

Most error messages are displayed on the sending unit. Depending on when the error occurs during the transfer process, you may see an error message on the receiving unit.

Error Message	Description
	The sending and receiving units are not connected properly, or the receiving unit is not set up to receive.
	The certificate on the receiving unit is not valid for the operating system (OS) or App on the sending unit. You must obtain and install a valid certificate. If the App no longer requires a certificate, you can download it again from the Texas Instruments Web site at education.ti.com and then install the App again on your calculator.
	An error occurred during the transfer. The current OS in the receiving unit is corrupted. You must reinstall the product software from a computer.

Error Message	Description
	Replace the batteries on the unit displaying this message.

Collecting and Transmitting ID Lists

The **VAR-LINK** screen F3 **6:Send ID List** menu option allows collection of electronic ID numbers from individual TI-89 Titanium, TI-89, Voyage™ 200, or TI-92 Plus devices.

ID Lists and Group Certificates

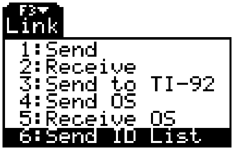
The ID list feature provides a convenient way to collect device IDs for group purchase of commercial applications. After the IDs are collected, transmit them to Texas Instruments so a group certificate can be issued.

A group certificate allows distribution of purchased software to multiple TI-89 Titanium, TI-89, Voyage™ 200, or TI-92 Plus units. The software can be loaded, deleted from, and reloaded to the devices as often as needed for as long as the software remains listed in the group certificate. You may add new ID numbers and/or new commercial applications to a group certificate.

Collecting ID Lists

You can use one device to collect all of the IDs, or use several collection units and then consolidate their ID lists onto one device.

To send an ID number from one device to another, first connect two units by using a USB unit-to-unit cable or I/O unit-to-unit cable.

Step:	On the:	Do this:
1.	Collecting unit (Receiving unit)	Display the Home screen. Press: [HOME]
2.	Sending unit	a. Press [2nd] [VAR-LINK] to display the **VAR-LINK** screen.
		b. Press [F3] **Link** and select **6:Send ID List**. 
		The sending unit adds a copy of its unique ID number to the collection unit's ID list. The sending unit always retains its own ID number, which cannot be deleted from the device.
3.	Additional units	Repeat steps 1 and 2 until all the IDs are collected onto one device. Depending on available memory in the collection device, it is possible to collect over 4,000 IDs.

Notes:

- You cannot view the ID list on the sending or collecting units.

- Each time an ID list is successfully sent from one device to another, the ID list is automatically deleted from the sending unit.

- If an ID is collected from a device twice, the duplicate ID is automatically deleted from the list.

Clearing the ID List

The ID list remains on the collection device after it is uploaded to the computer. You can then use the collection device to upload the list to other computers.

To clear the ID list from the collection unit:

1. Press 2nd [VAR-LINK] to display the **VAR-LINK** screen.

2. Press F1 **Manage** and select **A:Clear ID List.**

Compatibility among the TI-89 Titanium, Voyage™ 200, TI-89, and TI-92 Plus

In general, TI-89 Titanium, TI-89, Voyage™ 200, and TI-92 Plus data and programs are compatible with each other, with a few exceptions.

Most functions of the TI-89 Titanium are compatible with the TI-89, Voyage™ 200, and TI-92 Plus. The TI-89 Titanium and the TI-89 are similar, except that the TI-89 Titanium has more memory (more room for Apps and user archive) and the TI-89 Titanium has a USB port. The Voyage™ 200 is the same as the TI-92 Plus except it has more memory, and thus more room for applications (Apps).

All data is compatible among the TI-89 Titanium, TI-89, Voyage™ 200, and TI-92 Plus, but some programs written for one may not run or may not run the same on the other because of differences in the device’s screen sizes and keyboards and the USB port on the TI-89 Titanium.

Other incompatibilites can occur because of different version the operating system. To download the latest version of the operating system, visit the Texas Instruments Web site at education.ti.com/downloadticonnect.

To ➞ From ➘	TI-89 Titanium	TI-89	Voyage™ 200	TI-92 Plus
TI-89 Titanium	OS Apps Variables	Apps Variables	Variables	Variables
TI-89	Apps Variables	OS Apps Variables	Variables	Variables
Voyage™ 200	Variables	Variables	OS Apps Variables	Apps Variables
TI-92 Plus	Variables	Variables	Apps Variables	OS Apps Variables

Activities

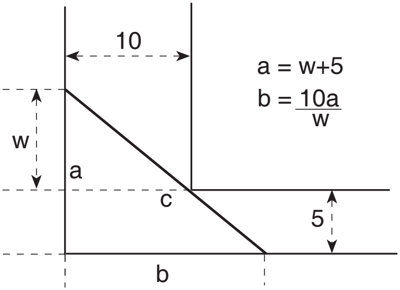
Analyzing the Pole-Corner Problem

A ten-foot-wide hallway meets a five-foot-wide hallway in the corner of a building. Find the maximum length pole that can be moved around the corner without tilting the pole.

Maximum Length of Pole in Hallway

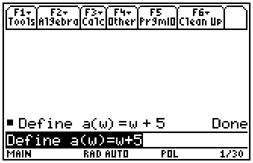
The maximum length of a pole **c** is the shortest line segment touching the interior corner and opposite sides of the two hallways as shown in the diagram below.

Use proportional sides and the Pythagorean theorem to find the length **c** with respect to **w.** Then find the zeros of the first derivative of **c(w)**. The minimum value of **c(w)** is the maximum length of the pole.

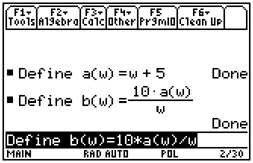


1. Define the expression for side **a** in terms of **w** and store it in **a(w)**.

   Note: When you want to define a function, use multiple character names as you build the definition.

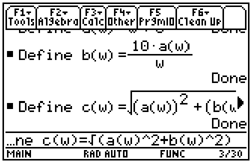


2. Define the expression for side **b** in terms of **w** and store it in **b(w).**



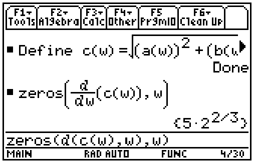
3. Define the expression for side **c** in terms of **w** and store it in **c(w).**

   Enter: **Define c(w)= √(a(w)^2+b(w)^2)**



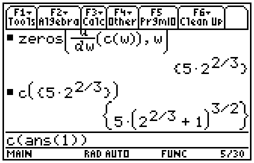
4. Use the **zeros( )** function to compute the zeros of the first derivative of **c(w)** to find the minimum value of **c(w)**.

   Note: The maximum length of the pole is the minimum value of **c(w)**.



5. Compute the exact maximum length of the pole.

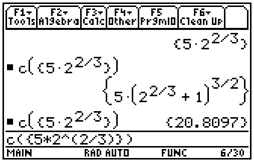
   Enter: **c** ([2nd] [ANS])



6. Compute the approximate maximum length of the pole.

   Result: Approximately 20.8097 feet.

   Note: Use the auto-paste feature to copy the result from step 4 to the entry line inside the parentheses of c( ) and press [♦] [ENTER].



Deriving the Quadratic Formula

This activity shows you how to derive the quadratic formula:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Detailed information about using the functions in this example can be found in *Symbolic Manipulation*.

Performing Computations to Derive the Quadratic Formula

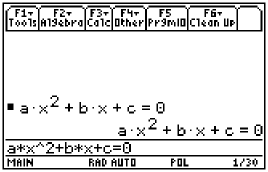
Perform the following steps to derive the quadratic formula by completing the square of the generalized quadratic equation.

1. Clear all one-character variables in the current folder.

   [2nd] [F6]

   Choose **1:Clear a-z** and press [ENTER] to confirm.

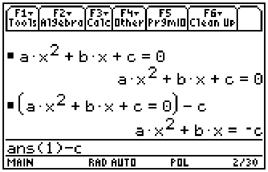
2. On the Home screen, enter the generalized quadratic equation: **ax^2+bx+c=0**.



3. Subtract c from both sides of the equation.

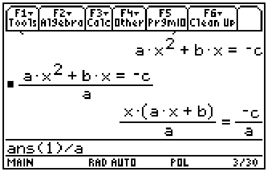
   [2nd] [ANS] [−] [alpha] **C**

   Note: This example uses the result of the last answer to perform computations on the TI-89 Titanium. This feature reduces keystroking and chances for error.

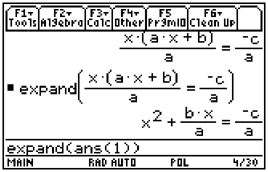


4. Divide both sides of the equation by the leading coefficient **a.**

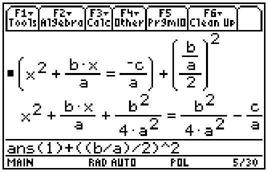
   Note: Continue to use the last answer ([2nd] [ANS]) as in step 3 in steps 4 through 9.



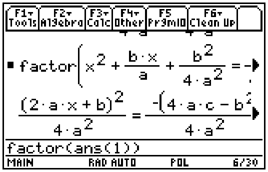
5. Use the **expand( )** function to expand the result of the last answer.



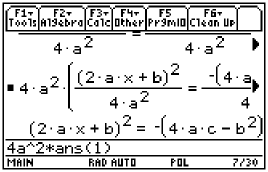
6. Complete the square by adding **((b/a)/2)^2** to both sides of the equation.



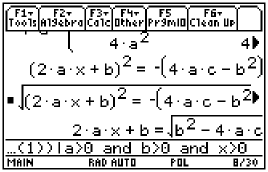
7. Factor the result using the **factor( )** function.



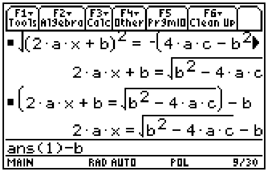
8. Multiply both sides of the equation by **4a^2**.



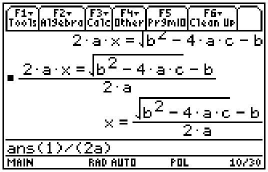
9. Take the square root of both sides of the equation with the constraint that **a>0** and **b>0** and **x>0**.



10. Solve for **x** by subtracting **b** from both sides and then dividing by **2a**.



Note: This is only one of the two general quadratic solutions due to the constraint in step 9.

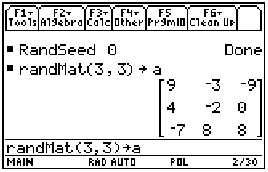


Exploring a Matrix

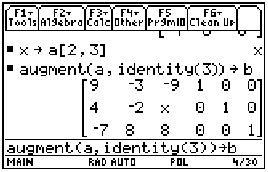
This activity shows you how to perform several matrix operations.

Perform these steps to generate a random matrix, augment and find the identity matrix, and then solve to find an invalid value of the inverse.

1. On the Home screen, use **RandSeed** to set the random number generator seed to the factory default, and then use **randMat( )** to create a random 3x3 matrix and store it in **a**.



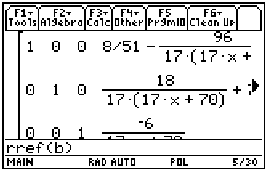
2. Replace the **[2,3]** element of the matrix with the variable **x**, and then use the **augment( )** function, to augment the 3x3 identity to **a** and store the result in **b.**



3. Use **rref( ) to** “row reduce” matrix **b**:

   The result will have the identity matrix in the first three columns and **a^-1** in the last three columns.

   Note: Use the cursor in the history area to scroll the result.

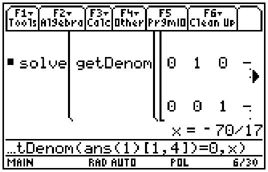


4. Solve for the value of **x** that will cause the inverse of the matrix to be invalid.

   Enter:
   solve(getDenom([2nd] [ANS] **[1,4])=0,x)**

   Result: **x=** $^{-}$**70/17**

   Note: Use the cursor in the history area to scroll the result.



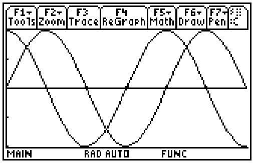
Exploring cos(x) = sin(x)

This activity uses two methods to find where **cos(x) = sin(x)** for the values of **x** between 0 and 3π.

Method 1: Graph Plot

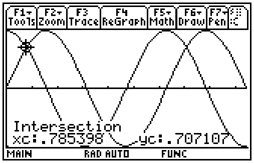
Perform the following steps to observe where the graphs of the functions **y1(x)=cos(x)** and **y2(x)=sin(x)** intersect.

1. In the **Y= Editor**, set **y1(x)=cos(x)** and **y2(x)=sin(x).**
2. In the **Window Editor**, set **xmin=0** and **xmax=3π.**
3. Press [F2] and select **A:ZoomFit.**



4. Find the intersection point of the two functions.

   Note: Press [F5] and select **5:Intersection.** Respond to the screen prompts to select the two curves, and the lower and upper bounds for intersection **A.**



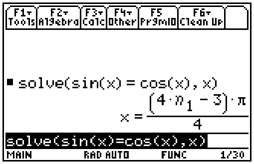
5. Note the **x** and **y** coordinates. (Repeat steps 4 and 5 to find the other intersections.)

Method 2: Symbolic Manipulation

Perform the following steps to solve the equation **sin(x)=cos(x)** with respect to **x.**

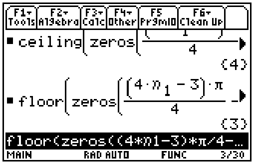
1. On the Home screen, enter **solve(sin(x)= cos(x),x).**

   The solution for **x** is where **@n1** is any integer.



2. Using the **ceiling( )** and **floor( )** functions, find the ceiling and floor values for the intersection points as shown.

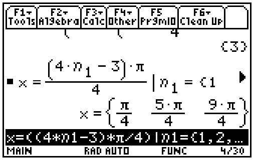
   Note: Move the cursor into the history area to highlight the last answer. Press [ENTER] to copy the result of the general solution.



3. Enter the general solution for x and apply the constraint for **@n1** as shown.

   Compare the result with Method 1.

   Note: To get the ***with*** operator, press: |



Finding Minimum Surface Area of a Parallelepiped

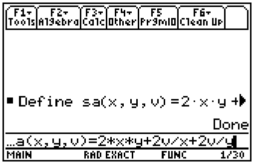
This activity shows you how to find the minimum surface area of a parallelepiped having a constant volume **V**. Detailed information about the steps used in this example can be found in *Symbolic Manipulation* and *3D Graphing*.

Exploring a 3D Graph of the Surface Area of a Parallelepiped

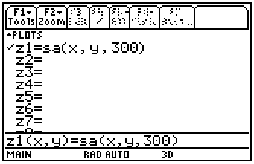
Perform the following steps to define a function for the surface area of a parallelepiped, draw a 3D graph, and use the **Trace** tool to find a point close to the minimum surface area.

1. On the Home screen, define the function **sa(x,y,v)** for the surface area of a parallelepiped.

   Enter: **define sa(x,y,v)=2∗x∗y + 2v/x+2v/y**

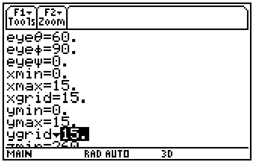


2. Select the 3D Graph mode. Then enter the function for **z1(x,y)** as shown in this example with volume **v=300**.

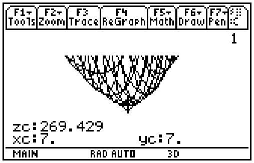


3. Set the Window variables to:

   eye= [60,90,0]
   x= [0,15,15]
   y= [0,15,15]
   z= [260,300]
   ncontour= [5]



4. Graph the function and use **Trace** to go to the point close to the minimum value of the surface area function.

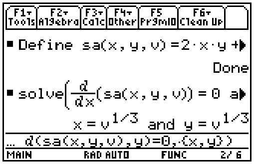


Finding the Minimum Surface Area Analytically

Perform the following steps to solve the problem analytically on the Home screen.

1. Solve for **x** and **y** in terms of **v**.

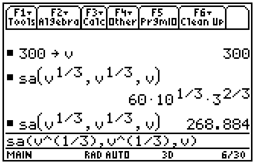
   Enter: **solve(d(sa(x,y,v),x)=0** and **d(sa(x,y,v),y)=0,{x,y})**



2. Find the minimum surface area when the value of **v** equals 300.

   Enter: **300→v**
   Enter: **sa(v^(1/3), v^(1/3),v)**

   Note: Press [ENTER] to obtain the exact result in symbolic form. Press [♦] [ENTER] to obtain the approximate result in decimal form.



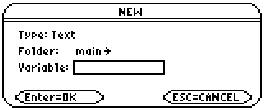
Running a Tutorial Script Using the Text Editor

This activity shows you how to use the **Text Editor** to run a tutorial script.

Running a Tutorial Script

Perform the following steps to write a script using the **Text Editor,** test each line, and observe the results in the history area on the Home screen.

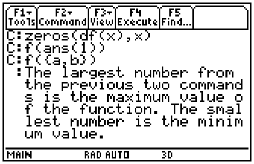
1. Open the **Text Editor**, and create a new variable named **demo1.**



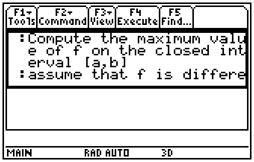
   Note: The command symbol **C** is accessed from the [F2] **1:Command** toolbar menu.

2. Type the following lines into the **Text Editor**.

```
  : Compute the maximum value of f on the closed interval [a,b]
  : assume that f is differentiable on [a,b]
C : define f(x)=x^3−2x^2+x−7
C : 1→a:3.22→b
C : d(f(x),x)→df(x)
C : zeros(df(x),x)
C : f(ans(1))
C : f({a,b})
  : The largest number from the previous two commands is the
  maximum value of the function. The smallest number is the
  minimum value.
```

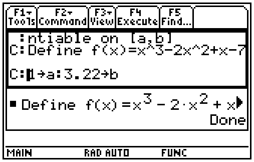


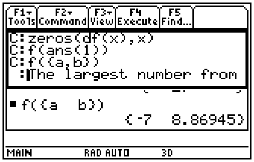
3. Press [F3] and select **1:Script view** to show the **Text Editor** and the Home screen on a split-screen. Move the cursor to the first line in the **Text Editor**.



4. Press [F4] repeatedly to execute each line in the script one at a time.

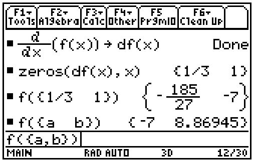
   Note: Press [F4] and select **2:Clear split** to go back to a full-sized **Text Editor** screen.





5. To see the results of the script on a full-sized screen, go to the Home screen.

   Note: Press [2nd] [QUIT] twice to display the Home screen.



Decomposing a Rational Function

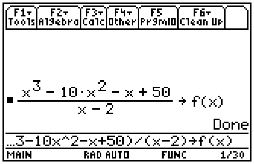
This activity examines what happens when a rational function is decomposed into a quotient and remainder. Detailed information about the steps used in this example can be found in *Basic Function Graphing* and *Symbolic Manipulation*.

To examine the decomposition of the rational function **f(x)=(x^3−10x^2−x+50)/(x−2)** on a graph:

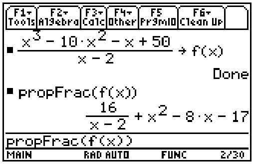
1. On the Home screen, enter the rational function as shown below and store it in a function **f(x)**.

   Enter: **(x^3−10x^2−x+50)/(x−2)→f(x)**

   Note: Actual entries are displayed in reverse type in the example screens.

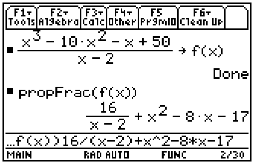


2. Use the proper fraction function (**propFrac**) to split the function into a quotient and remainder.



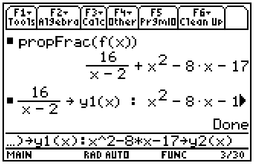
3. Copy the last answer to the entry line.
   –or–
   Enter: **16/(x−2)+x^2−8∗x−17**

   Note: Move the cursor into the history area to highlight the last answer. Press [ENTER] to copy it to the entry line.

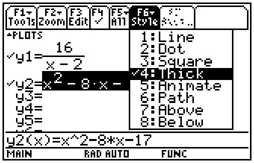


4. Edit the last answer in the entry line. Store the remainder to **y1(x)** and the quotient to **y2(x)** as shown.

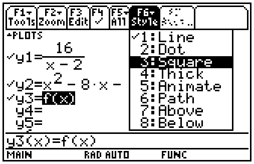
   Enter: **16/(x−2)→y1(x): x^2−8∗x−17→y2(x)**



5. In the **Y= Editor**, select the thick graphing style for **y2(x)**.



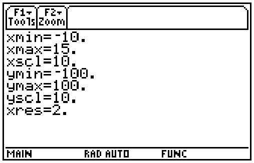
6. Add the original function **f(x)** to **y3(x)** and select the square graphing style.



7. In the **Window Editor**, set the window variables to:

   x= [$^{-}$10,15,10]

   y= [$^{-}$100,100,10]

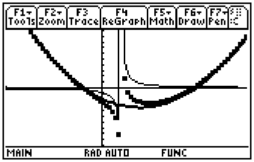


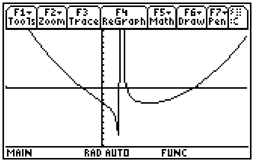
8. Draw the graph.

   Note: Be sure the Graph mode is set to Function.

Observe that the global behavior of the **f(x)** function is basically represented by the quadratic quotient **y2(x).** The rational expression is basically a quadratic function as **x** gets very large in both the positive and negative directions.

The lower graph is **y3(x)=f(x)** graphed separately using the line style.





Studying Statistics: Filtering Data by Categories

This activity provides a statistical study of the weights of high school students using categories to filter the data.

Each student is placed into one of eight categories depending on the student's sex and academic year (freshman, sophomore, junior, or senior). The data (weight in pounds) and respective categories are entered in the **Data/Matrix Editor**.

Table 1: Category vs. Description

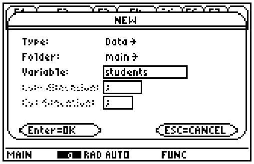
Category (C2)	Academic Year and Sex
1	Freshman boys
2	Freshman girls
3	Sophomore boys
4	Sophomore girls
5	Junior boys
6	Junior girls
7	Senior boys
8	Senior girls

Table 2: C1 (weight of each student in pounds) vs. C2 (category)

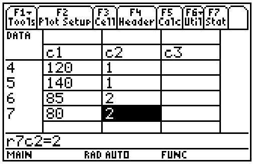
C1	C2	C1	C2	C1	C2	C1	C2
110	1	115	3	130	5	145	7
125	1	135	3	145	5	160	7
105	1	110	3	140	5	165	7
120	1	130	3	145	5	170	7
140	1	150	3	165	5	190	7
85	2	90	4	100	6	110	8
80	2	95	4	105	6	115	8
90	2	85	4	115	6	125	8
80	2	100	4	110	6	120	8
95	2	95	4	120	6	125	8

Perform the following steps to compare the weight of high school students to their year in school.

1. Start the **Data/Matrix Editor**, and create a new Data variable named **students.**

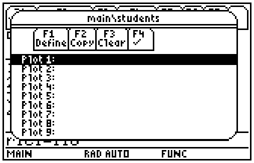


2. Enter the data and categories from Table 2 into columns **c1** and **c2**, respectively.

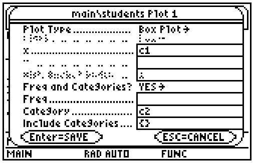


3. Open the [F2] **Plot Setup** toolbar menu.

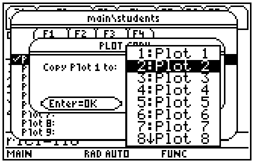
   Note: Set up several box plots to compare different subsets of the entire data set.



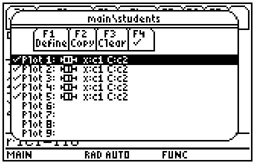
4. Define the plot and filter parameters for **Plot 1** as shown in this screen.



5. Copy **Plot 1** to **Plot 2**.

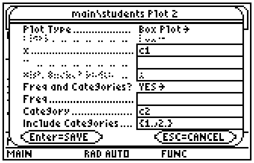


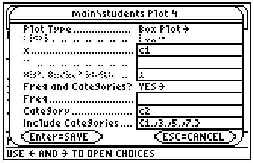
6. Repeat step 5 and copy **Plot 1** to **Plot 3**, **Plot 4**, and **Plot 5**.



7. Press [F1], and modify the **Include Categories** item for **Plot 2** through **Plot 5** to the following:

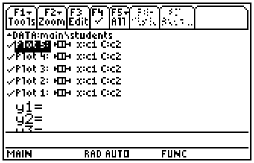
   Plot 2: {1,2}
   (freshman boys, girls)
   Plot 3: {7,8}
   (senior boys, girls)
   Plot 4: {1,3,5,7}
   (all boys)
   Plot 5: {2,4,6,8}
   (all girls)



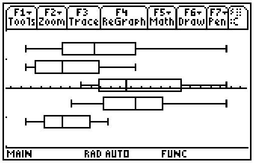


8. In the **Y= Editor**, deselect any functions that may be selected from a previous activity.

   Note: Only **Plot 1** through **Plot 5** should be selected.

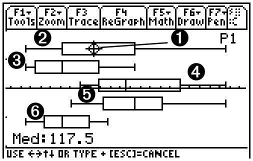


9. Display the plots by pressing [F2] and selecting **9:Zoomdata.**



10. Use the **Trace** tool to compare the median student weights for different subsets.

    ❶ median, all students
    ❷ all students
    ❸ all freshmen
    ❹ all seniors
    ❺ all boys
    ❻ all girls



CBL 2™ Program for the TI-89 Titanium

This activity provides a program that can be used when the TI-89 Titanium is connected to a Calculator-Based Laboratory™ (CBL 2™) unit. This program works with the "Newton's Law of Cooling" experiment. You can use your computer keyboard to type lengthy text and then use TI Connect™ software to send it to the calculator. More CBL 2™ programs are available from the TI Web site at educaton.ti.com.

Program Instruction	Description
:cooltemp( )	Program name
:Prgm	
:Local i	Declare local variable; exists only at run time.

Program Instruction	Description
:setMode("Graph","FUNCTION")	Set up the TI-89 Titanium for function graphing.
:PlotsOff	Turn off any previous plots.
:FnOff	Turn off any previous functions.
:ClrDraw	Clear any items previously drawn on graph screens.
:ClrGraph	Clear any previous graphs.
:ClrIO	Clear the TI-89 Titanium Program IO (input/output) screen.
:⁻10→xmin:99→xmax:10→xscl	Set up the Window variables.
:⁻20→ymin:100→ymax:10→yscl	
:{0}→data	Create and/or clear a list named data.
:{0}→time	Create and/or clear a list named time.
:Send{1,0}	Send a command to clear the CBL 2™ unit.
:Send{1,2,1}	Set up Chan. 2 of the CBL 2™ to AutoID to record temp.
:Disp "Press ENTER to start"	Prompt the user to press [ENTER].
:Disp "graphingTemperature."	
:Pause	Wait until the user is ready to start.
:PtText "TEMP(C)",2,99	Label the y axis of the graph.
:PtText "T(S)",80,⁻5	Label the x axis of the graph.

Program Instruction	Description
:Send{3,1,⁻1,0}	Send the Trigger command to the CBL 2™; collect data in real-time.
:For i,1,99	Repeat next two instructions for 99 temperature readings.
:Get data[i]	Get a temperature from the CBL 2™ and store it in a list.
:PtOn i,data[i]	Plot the temperature data on a graph.
:EndFor	
:seq(i,i,1,99,1)→time	Create a list to represent time or data sample number.
:NewPlot 1,1,time,data,,,,4	Plot time and data using NewPlot and the Trace tool.
:DispG	Display the graph.
:PtText "TEMP(C)",2,99	Re-label the axes.
:PtText "T(S)",80,⁻5	
:EndPrgm	Stop the program.

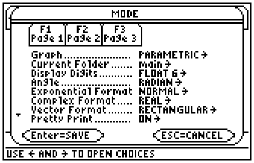
You can also use the Calculator-Based Ranger™ system (CBR™) to explore the mathematical and scientific relationships between distance, velocity, acceleration, and time using data collected from activities you perform.

Studying the Flight of a Hit Baseball

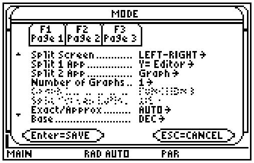
This activity uses the split screen settings to show a parametric graph and a table at the same time to study the flight of a hit baseball.

Perform the following steps to study the flight of a hit baseball that has an initial velocity of 95 feet per second and an initial angle of 32 degrees.

1. Set the modes for **Page 1** as shown in this screen.



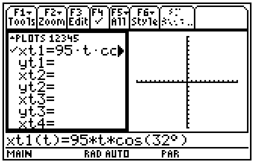
2. Set the modes for **Page 2** as shown in this screen.



3. In the **Y= Editor** on the left side, enter the equation for the distance of the ball at time **t** for **xt1(t)**.

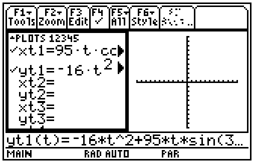
   xt1(t)=95∗t∗cos(32°)

   Note: Press [2nd] [°] to obtain the degree symbol.



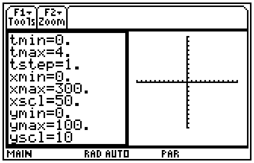
4. In the **Y= Editor**, enter the equation for the height of the ball at time t for **yt1(t).**

   yt1(t)=⁻16∗t^2+95∗t∗sin(32°)



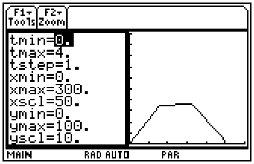
5. Set the Window variables to:

   t values= [0,4,1]
   x values= [0,300,50]
   y values= [0,100,10]



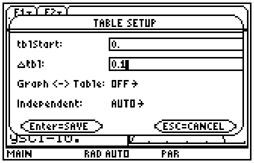
6. Switch to the right side and display the graph.

   Note: Press [2nd] [⊞].



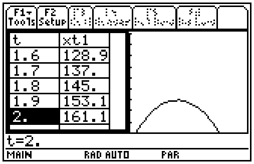
7. Display the **TABLE SETUP** dialog box, and change **tblStart** to **0** and **∆tbl** to **0.1.**

   Note: Press [◆] [TBLSET].



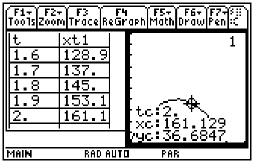
8. Display the table in the left side and press [↓] to highlight **t=2.**

   Note: Press [◆] [TABLE].



9. Switch to the right side. Press [F3], and trace the graph to show the values of **xc** and **yc** when **tc=2.**

   Note: As you move the trace cursor from **tc=0.0** to **tc=3.1**, you will see the position of the ball at time **tc.**



Optional Exercise

Assuming the same initial velocity of 95 feet per second, find the angle that the ball should be hit to achieve the greatest distance.

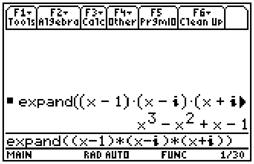
Visualizing Complex Zeros of a Cubic Polynomial

This activity describes graphing the complex zeros of a cubic polynomial.

Visualizing Complex Roots

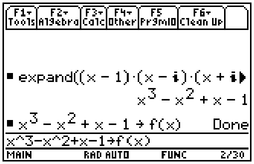
Perform the following steps to expand the cubic polynomial **$(x-1)(x-i)(x+i)$**, find the absolute value of the function, graph the modulus surface, and use the **Trace** tool to explore the modulus surface.

1. On the Home screen, use the **expand( )** function to expand the cubic expression **$(x-1)(x-i)(x+i)$** and see the first polynomial.



2. Copy and paste the last answer to the entry line and store it in the function **f(x)**.

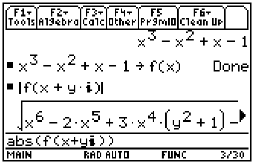
   Note: Move the cursor into the history area to highlight the last answer and press [ENTER], to copy it to the entry line.



3. Use the **abs( )** function to find the absolute value of **f(x+y*i*)**.

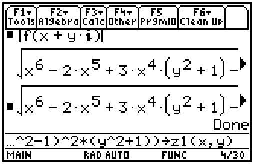
   (This calculation may take about 2 minutes.)

   Note: The absolute value of a function forces any roots to visually just touch rather than cross the **x** axis. Likewise, the absolute value of a function of two variables will force any roots to visually just touch the **xy** plane.



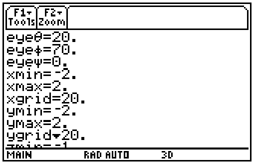
4. Copy and paste the last answer to the entry line and store it in the function **z1(x,y)**.

   Note: The graph of **z1(x,y)** will be the modulus surface.



5. Set the unit to 3D graph mode, turn on the axes for graph format, and set the Window variables to:

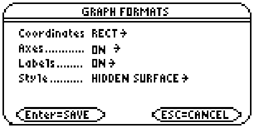
   eye= [20,70,0]
   x= [−2,2,20]
   y= [−2,2,20]
   z= [−1,2]
   ncontour= [5]



6. In the **Y=Editor**, press:
   [♦] [I]
   and set the Graph Format variables to:

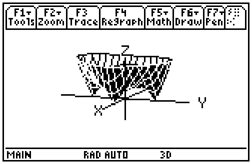
   Axes= **ON**
   Labels= **ON**
   Style= **HIDDEN SURFACE**

   Note: Calculating and drawing the graph takes about three minutes.

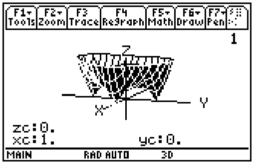


7. Graph the modulus surface.

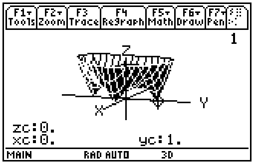
   The 3D graph is used to visually display a picture of the roots where the surface touches the **xy** plane.



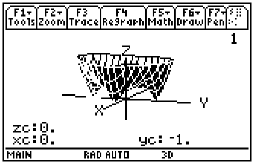
8. Use the Trace tool to explore the function values at **x=1** and **y=0.**



9. Use the Trace tool to explore the function values at **x=0** and **y=1.**



10. Use the Trace tool to explore the function values at **x=0** and **y=⁻1**.



Summary

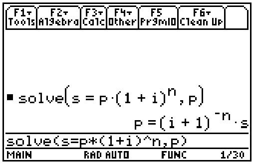
Note that **zc** is zero for each of the function values in steps 7–9. Thus, the complex zeros 1,⁻i, i of the polynomial x^3-x^2+x-1 can be visualized with the three points where the graph of the modulus surface touches the **xy** plane.

Solving a Standard Annuity Problem

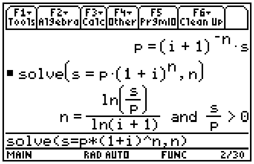
This activity can be used to find the interest rate, starting principal, number of compounding periods, and future value of an annuity.

Perform the following steps to find the interest rate (**i**) of an annuity where the starting principal (**p**) is 1,000, number of compounding periods (**n**) is 6, and the future value (**s**) is 2,000.

1. On the Home screen, enter the equation to solve for **p**.



2. Enter the equation to solve for **n**.



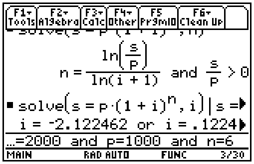
3. Enter the equation to solve for i using the ***with*** operator.

   solve(s=p∗(1+i)^n,i) | s=2000 and **p=1000** and **n=6**

   Result: The interest rate is 12.246%.

   Note:

   - To enter the “*with*” (|) operator: [|]
   - Press [◆] [ENTER] to obtain a floating-point result.



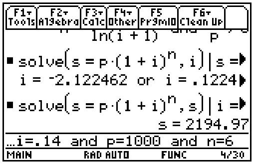
Finding the Future Value of an Annuity

Find the future value of an annuity using the values from the previous example where the interest rate is 14%.

Enter the equation to solve for **s.**

solve(s=p∗(1+i)^n,s) | i=.14 and **p=1000** and **n=6**

Result: The future value at 14% interest is 2,194.97.



Computing the Time-Value-of-Money

This activity creates a function that can be used to find the cost of financing an item. Detailed information about the steps used in this example can be found in the electronic chapter *Programming*, which is available from the TI Web site at education.ti.com and on the CD in this package.

Time-Value-of- Money Function

In the Program Editor, define the following Time-Value-of-Money (**tvm**) function where **temp1** = number of payments, **temp2** = annual interest rate, **temp3** = present value, **temp4** = monthly payment, **temp5** = future value, and **temp6** = begin- or end-of-payment period

(1 = beginning of month, 0 = end of month).

```
:tvm(temp1,temp2,temp3,temp4,temp5,temp6)
:Func
:Local tempi,tempfunc,tempstr1
:⁻temp3+(1+temp2/1200 temp6) temp4 ((1−(1+temp2/1200)^
    (⁻temp1))/(temp2/1200))−temp5 (1+temp2/1200)^(⁻temp1)
    →tempfunc
:For tempi,1,5,1
:"temp"&exact(string(tempi))→tempstr1
:If when(#tempstr1=0,false,false,true) Then
:If tempi=2
:Return approx(nsolve(tempfunc=0,#tempstr1) | #tempstr1>0 and
    #tempstr1<100)
:Return approx(nsolve(tempfunc=0,#tempstr1))
:EndIf
:EndFor
:Return "parameter error"
:EndFunc
```

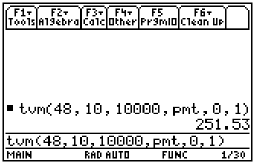
Note: You can use your computer keyboard to type lengthy text and then use TI Connect™ software to send it to the TI-89 Titanium.

Finding the Monthly Payment

Find the monthly payment on 10,000 if you make 48 payments at 10% interest per year.

On the Home screen, enter the **tvm** values to find **pmt.**

Result: The monthly payment is 251.53.

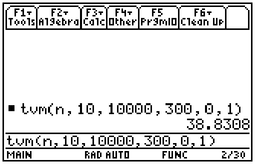


Finding the Number of Payments

Find the number of payments it will take to pay off the loan if you could make a 300 payment each month.

On the Home screen, enter the **tvm** values to find **n.**

Result: The number of payments is 38.8308.

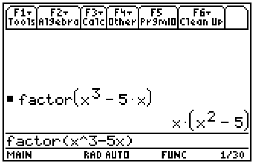


Finding Rational, Real, and Complex Factors

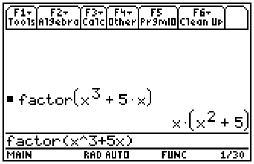
This activity shows how to find rational, real, or complex factors of expressions. Detailed information about the steps used in this example can be found in *Symbolic Manipulation*.

Enter the expressions shown below on the Home screen.

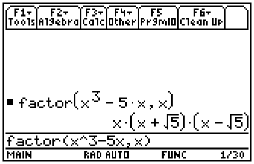
1. **factor(x^3−5x)** [ENTER] displays a rational result.



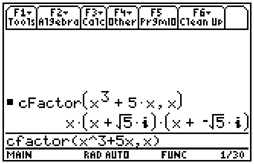
2. **factor(x^3+5x)** [ENTER] displays a rational result.



3. **factor(x^3−5x,x)** [ENTER] displays a real result.



4. **cfactor(x^3+5x,x)** [ENTER] displays a complex result.



Simulation of Sampling without Replacement

This activity simulates drawing different colored balls from an urn without replacing them. Detailed information about the steps used in this example can be found in the electronic chapter *Programming*.

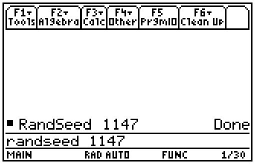
Sampling-without- Replacement Function

In the **Program Editor**, define **drawball( )** as a function that can be called with two parameters. The first parameter is a list where each element is the number of balls of a certain color. The second parameter is the number of balls to select. This function returns a list where each element is the number of balls of each color that were selected.

```
:drawball(urnlist,drawnum)
:Func
:Local templist,drawlist,colordim,
     numballs,i,pick,urncum,j
:If drawnum>sum(urnlist)
:Return “too few balls”
:dim(urnlist)→colordim
:urnlist→templist
:newlist(colordim)→drawlist
:For i,1,drawnum,1
:sum(templist)→numballs
:rand(numballs)→pick
(continued in next column)
:For j,1,colordim,1
:cumSum(templist)→urncum
:If pick ≤ urncum[j] Then
:drawlist[j]+1→drawlist[j]
:templist[j]−1→templist[j]
:Exit
:EndIf
:EndFor
:EndFor
:Return drawlist
:EndFunc
```

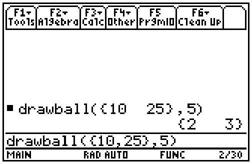
Suppose an urn contains **n1** balls of a color, **n2** balls of a second color, **n3** balls of a third color, etc. Simulate drawing balls without replacing them.

1. Enter a random seed using the **RandSeed** command.



2. Assuming the urn contains 10 red balls and 25 white balls, simulate picking 5 balls at random from the urn without replacement. Enter **drawball({10,25},5)**.

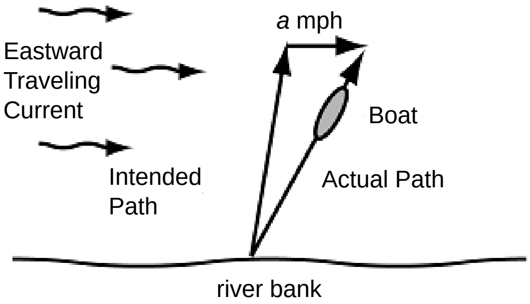
   Result: 2 red balls and 3 white balls.



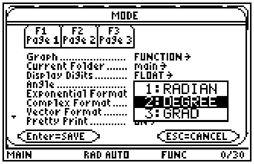
Using Vectors to Determine Velocity

A small fishing boat leaves from the south bank of the Allegheny River and heads at an 80° angle with an engine speed of 20 knots. However, the eastward force of the current carries the boat along so it actually travels at a 60° angle with the shore.

How fast is the current, and how fast does the boat actually travel?

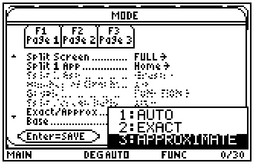


1. Set the modes for **Page 1** as shown in this screen. (Show angles in degrees and display all digits with a floating decimal point.)

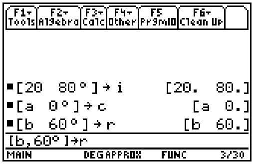
   Press: [MODE] ↓ ↓ ↓. On the Angle option, select **2:DEGREE**. On the Display Digits option, select **E:FLOAT**.

2. Set the modes for **Page 2** as shown in this screen. (Display answers in decimal form.)

   Press: [MODE] [F2] ↓ ↓. On the Exact/Approx option, select **3:APPROXIMATE**.

3. Enter vectors describing the initial path of the boat, water current, and resultant path of the boat.



   Store these vectors as **i**, **c**, and **r**. Use the value **a** for the unknown speed of the current. Use the value **b** for the speed of the boat.

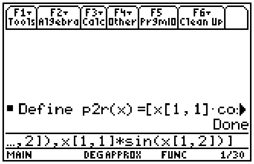
   Enter:

   [20,80°]→i

   [a,0°]→c

   [b,60°]→r

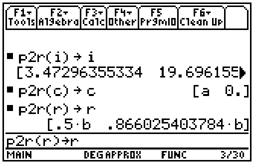
Vectors are commonly written in either polar or rectangular form, so it is useful to convert polar vectors into rectangular form.



4. Define function **p2r**.

   Enter: **Define p2r(x)=[x[1,1]*cos(x[1,2]), x[1,1]*sin(x[1,2])]**

When converted to rectangular form, the sum of vectors **i** and **c** equals the resultant vector **r.**



5. Using function **p2r**, convert vectors **i**, **c**, and **r** to rectangular form.

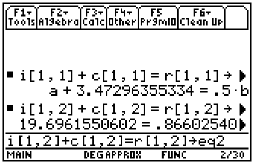
   Enter:

   p2r(i)→i

   p2r(c)→c

   p2r(r)→r

Because the vectors are equal, the x-coordinate of **i**+**c** must equal the x-coordinate of the resultant vector **r**. Likewise, the y-coordinate of **i**+**c** must equal the y-coordinate of resultant vector **r.**



6. Set up two equations involving vectors **i**+**c** and **r.**

   - Equation 1 sets the x-coordinates equal to each other.
   - Equation 2 sets the y-coordinates equal.

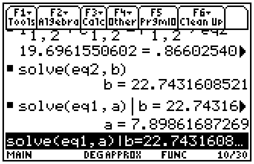
   Store these equations into **eq1** and **eq2**, respectively. Enter:

   i[1,1]+c[1,1]=r[1,1]→eq1

   i[1,2]+c[1,2]=r[1,2]→eq2

7. Solve **eq2** for **b** to calculate the actual speed of the boat.

   solve(eq2,b)



8. Substitute the known value of **b** into **eq1**, and solve **eq1** for **a** to determine **a**, the speed of the eastward traveling current.

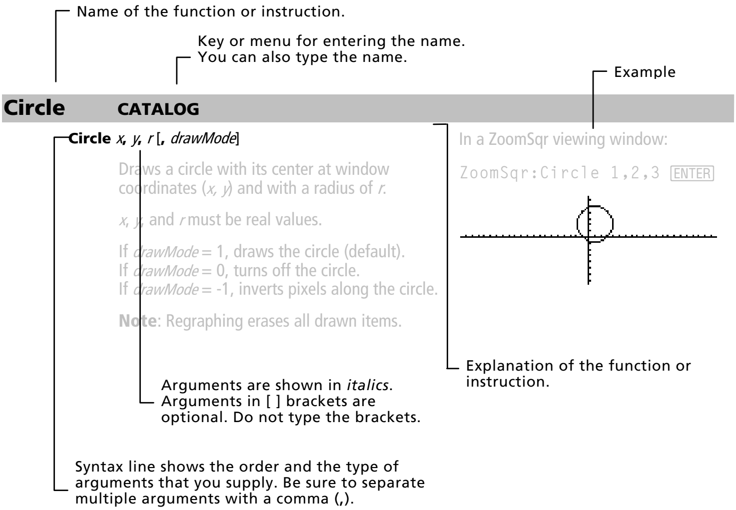
   solve(eq1,a) | b

The boat travels at a speed of 22.7 knots, and the water current is approximately 7.9 knots.

A

Appendix A: Categorical Listing of Operations

This section describes the syntax and action of each TI-89 Titanium function and instruction that is included in the operating system (OS). See modules relating to calculator software applications (Apps) for functions and instructions specific to those Apps.



Quick-Find Locator

This section lists the TI-89 Titanium functions and instructions in functional groups along with the page numbers where they are described.

Algebra

Calculus

Graphics

Lists

Math

Matrices

Programming

=	900	**≠**	900	**<**	901
≤	901	**>**	901	**≥**	901
# (indirection)	904	→ (store)	908	© (comment)	908
and	784	**ans()**	786	**Archive**	786
checkTmr()	789	**ClockOff**	790	**ClockOn**	790
ClrErr	790	**ClrGraph**	791	**ClrHome**	791
ClrIO	791	**ClrTable**	791	**CopyVar**	793
CustmOff	799	**CustmOn**	799	**Custom**	800
Cycle	800	**dayOfWk()**	803	**Define**	804
DelFold	804	**DelVar**	805	**Dialog**	807
Disp	808	**DispG**	808	**DispHome**	808
DispTbl	809	**DropDown**	810	**Else**	812
ElseIf	812	**EndCustm**	812	**EndDlog**	813
EndFor	813	**EndFunc**	813	**EndIf**	813
EndLoop	813	**EndPrgm**	813	**EndTBar**	813
EndTry	813	**EndWhile**	813	**entry()**	813
Exec	814	**Exit**	814	**For**	819
format()	819	**Func**	820	**Get**	820
GetCalc	821	**getConfg()**	821	**getDate()**	822
getDtFmt()	822	**getDtStr()**	822	**getFold()**	822
getKey()	823	**getMode()**	823	**getTime()**	823
getTmFmt()	823	**getTmStr()**	824	**getTmZn()**	824
getType()	824	**getUnits()**	825	**Goto**	825
If	826	**Input**	827	**InputStr**	828
isClkOn()	829	**Item**	829	**Lbl**	829
left()	830	**Local**	834	**Lock**	834
Loop	836	**MoveVar**	839	**NewFold**	841
NewProb	842	**not**	843	**or**	845
Output	845	**part()**	846	**PassErr**	848
Pause	848	**PopUp**	850	**Prgm**	850
Prompt	851	**Rename**	859	**Request**	859
Return	859	**right()**	859	**Send**	863
SendCalc	863	**SendChat**	863	**setDate()**	864
setDtFmt()	864	**setFold()**	864	**setGraph()**	864
setMode()	865	**setTable()**	866	**setTime()**	866
setTmFmt()	867	**setTmZn()**	867	**startTmr()**	877
setUnits()	867	**Stop**	878	**Style**	878
switch()	879	**Table**	880	**Text**	883
Then	883	**timeCnv()**	883	**Title**	884
Toolbar	885	**Try**	886	**Unarchiv**	886
Unlock	887	**when()**	887	**While**	888
xor	889				

Statistics

! (factorial)	902	**BldData**	788	**CubicReg**	799
cumSum()	799	**ExpReg**	816	**LinReg**	832
LnReg	834	**Logistic**	835	**mean()**	837
median()	837	**MedMed**	838	**nCr()**	840
NewData	840	**NewPlot**	842	**nPr()**	843
OneVar	844	**PlotsOff**	849	**PlotsOn**	849
PowerReg	850	**QuadReg**	855	**QuartReg**	855
rand()	856	**randNorm()**	856	**RandSeed**	857
ShowStat	870	**SinReg**	873	**SortA**	876
SortD	876	**stdDev()**	877	**TwoVar**	886
variance()	887				

Strings

& (append)	902	# (indirection)	904	**char()**	789
dim()	808	**expr()**	815	**format()**	819
inString()	828	**left()**	830	**mid()**	838
ord()	845	**right()**	859	**rotate()**	860
shift()	869	**string()**	878		

Alphabetical Listing of Operations

Operations whose names are not alphabetic (such as +, !, and >) are listed at the end of this appendix. Unless otherwise specified, all examples in this section were performed in the default reset mode, and all variables are assumed to be undefined. Additionally, due to formatting restraints, approximate results are truncated at three decimal places (3.14159265359 is shown as 3.141...).

abs() — MATH/Number menu

abs(*expression1***)** ⇒ *expression*
abs(*list1***)** ⇒ *list*
abs(*matrix1***)** ⇒ *matrix*

Returns the absolute value of the argument.

If the argument is a complex number, returns the number's modulus.

Note: All undefined variables are treated as real variables.

```
abs({π/2,⁻π/3}) [ENTER]        {π/2  π/3}
abs(2−3i) [ENTER]              √13
abs(z) [ENTER]                 |z|
abs(x+yi) [ENTER]              √(x²+y²)
```

and — MATH/Test and MATH/Base menus

Boolean expression1 **and** *expression2* ⇒ *Boolean expression*
Boolean list1 **and** *list2* ⇒ *Boolean list*
Boolean matrix1 **and** *matrix2* ⇒ *Boolean matrix*

Returns true or false or a simplified form of the original entry.

```
x≥3 and x≥4 [ENTER]                      x≥4
{x≥3,x≤0} and {x≥4,x≤⁻2} [ENTER]
                                  {x ≥ 4   x ≤ ⁻2}
```

---

integer1 **and** *integer2* ⇒ *integer*

Compares two real integers bit-by-bit using an **and** operation. Internally, both integers are converted to signed, 32-bit binary numbers. When corresponding bits are compared, the result is 1 if both bits are 1; otherwise, the result is 0. The returned value represents the bit results, and is displayed according to the Base mode.

You can enter the integers in any number base. For a binary or hexadecimal entry, you must use the 0b or 0h prefix, respectively. Without a prefix, integers are treated as decimal (base 10).

If you enter a decimal integer that is too large for a signed, 32-bit binary form, a symmetric modulo operation is used to bring the value into the appropriate range.

In Hex base mode:

```
0h7AC36 and 0h3D5F [ENTER]   0h2C16
```

Important: Zero, not the letter O.

In Bin base mode:

```
0b100101 and 0b100 [ENTER]    0b100
```

In Dec base mode:

```
37 and 0b100 [ENTER]              4
```

Note: A binary entry can have up to 32 digits (not counting the 0b prefix). A hexadecimal entry can have up to 8 digits.

AndPic CATALOG

AndPic *picVar*[, *row, column*]

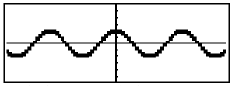
Displays the Graph screen and logically "ANDS" the picture stored in *picVar* and the current graph screen at pixel coordinates *(row, column)*.

picVar must be a picture type.

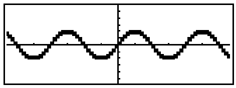
Default coordinates are (0,0), which is the upper left corner of the screen.

In function graphing mode and Y= Editor:

```
y1(x) = cos(x) ⊖
```
[2nd][F6] Style = 3:Square
[F6] Style = 3:Square
[F2] Zoom = 7:ZoomTrig
[F1] = 2:Save Copy As...
Type = Picture, Variable = PIC1

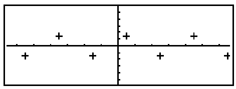


```
y2(x) = sin(x)
```
[2nd][F6] Style = 3:Square
[F6] Style = 3:Square
y1 = no checkmark (F4 to deselect)
[F2] Zoom = 7:ZoomTrig



[HOME]
[◆][CALC HOME]

AndPic PIC1 [ENTER] Done



angle() MATH/Complex menu

angle(*expression1***)** ⇒ *expression*

Returns the angle of *expression1*, interpreting *expression1* as a complex number.

Note: All undefined variables are treated as real variables.

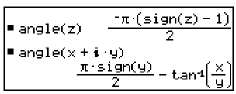
In Degree angle mode:

angle(0+2*i*) [ENTER] 90

In Radian angle mode:

angle(1+*i*) [ENTER] $\frac{\pi}{4}$

angle(z) [ENTER]
angle(x+ *i*y) [ENTER]



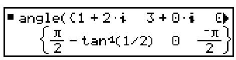
---

angle(*list1***)** ⇒ *list*
angle(*matrix1***)** ⇒ *matrix*

Returns a list or matrix of angles of the elements in *list1* or *matrix1*, interpreting each element as a complex number that represents a two-dimensional rectangular coordinate point.

In Radian angle mode:

angle({1+2*i*,3+0*i*,0−4*i*}) [ENTER]



ans() [2nd] [ANS] key

ans() ⇒ *value*
ans(*integer***)** ⇒ *value*

Returns a previous answer from the Home screen history area.

integer, if included, specifies which previous answer to recall. Valid range for *integer* is from 1 to 99 and cannot be an expression. Default is 1, the most recent answer.

To use **ans()** to generate the Fibonacci sequence on the Home screen, press:

```
1 [ENTER]                                  1
1 [ENTER]                                  1
[2nd] [ANS] [+] [2nd] [ANS] [(] [-] 2 [ENTER]   2
[ENTER]                                    3
[ENTER]                                    5
```

approx() MATH/Algebra menu

approx(*expression***)** ⇒ *value*

Returns the evaluation of *expression* as a decimal value, when possible, regardless of the current Exact/Approx mode.

This is equivalent to entering *expression* and pressing [◆] [ENTER] on the Home screen.

```
approx(π) [ENTER]        3.141...
```

---

approx(*list1***)** ⇒ *list*
approx(*matrix1***)** ⇒ *matrix*

Returns a list or matrix where each element has been evaluated to a decimal value, when possible.

```
approx({sin(π),cos(π)}) [ENTER]
                    {0.  ⁻1.}

approx([√(2),√(3)]) [ENTER]
              [1.414...  1.732...]
```

Archive CATALOG

Archive *var1* [, *var2*] [, *var3*] …

Moves the specified variables from RAM to the user data archive memory.

You can access an archived variable the same as you would a variable in RAM. However, you cannot delete, rename, or store to an archived variable because it is locked automatically.

To unarchive variables, use **Unarchiv**.

```
10→arctest [ENTER]              10
Archive arctest [ENTER]         Done
5*arctest [ENTER]               50
15→arctest [ENTER]
```


```
[ESC]
Unarchiv arctest [ENTER]        Done
15→arctest [ENTER]              15
```

arcLen() MATH/Calculus menu

arcLen(*expression1*,*var*,*start*,*end***)** ⇒ *expression*

Returns the arc length of *expression1* from *start* to *end* with respect to variable *var*.

Regardless of the graphing mode, arc length is calculated as an integral assuming a function mode definition.

```
arcLen(cos(x),x,0,π) [ENTER] 3.820...

arcLen(f(x),x,a,b) [ENTER]
```

$$\int_a^b \sqrt{\left(\frac{d}{dx}(f(x))\right)^2+1}\ dx$$

---

arcLen(*list1*,*var*,*start*,*end***)** ⇒ *list*

Returns a list of the arc lengths of each element of *list1* from *start* to *end* with respect to *var*.

```
arcLen({sin(x),cos(x)},x,0,π)
                 {3.820...  3.820...}
```

augment() MATH/Matrix menu

augment(*list1*, *list2***)** ⇒ *list*

Returns a new list that is *list2* appended to the end of *list1*.

```
augment({1,⁻3,2},{5,4}) [ENTER]
                    {1 ⁻3 2 5 4}
```

augment(*matrix1***,** *matrix2***)** ⇒ *matrix*
augment(*matrix1***;** *matrix2***)** ⇒ *matrix*

Returns a new matrix that is *matrix2* appended to *matrix1*. When the "," character is used, the matrices must have equal row dimensions, and *matrix2* is appended to *matrix1* as new columns. When the ";" character is used, the matrices must have equal column dimensions, and *matrix2* is appended to *matrix1* as new rows. Does not alter *matrix1* or *matrix2*.

`[1,2;3,4]→M1` ENTER	$\begin{bmatrix}1 & 2\\3 & 4\end{bmatrix}$
`[5;6]→M2` ENTER	$\begin{bmatrix}5\\6\end{bmatrix}$
`augment(M1,M2)` ENTER	$\begin{bmatrix}1 & 2 & 5\\3 & 4 & 6\end{bmatrix}$
`[5,6]→M2` ENTER	$\begin{bmatrix}5 & 6\end{bmatrix}$
`augment(M1;M2)` ENTER	$\begin{bmatrix}1 & 2\\3 & 4\\5 & 6\end{bmatrix}$

avgRC() CATALOG

avgRC(*expression1***,** *var* [**,** *h*]**)** ⇒ *expression*

Returns the forward-difference quotient (average rate of change).

expression1 can be a user-defined function name (see **Func**).

h is the step value. If *h* is omitted, it defaults to 0.001.

Note that the similar function **nDeriv()** uses the central-difference quotient.

`avgRC(f(x),x,h)` ENTER	$\frac{f(x+h) - f(x)}{h}$
`avgRC(sin(x),x,h)|x=2` ENTER	$\frac{\sin(h+2) - \sin(2)}{h}$
`avgRC(x^2-x+2,x)` ENTER	`2.·(x-.4995)`
`avgRC(x^2-x+2,x,.1)` ENTER	`2.·(x-.45)`
`avgRC(x^2-x+2,x,3)` ENTER	`2·(x+1)`

▶Bin MATH/Base menu

integer1 **▶Bin** ⇒ *integer*

Converts *integer1* to a binary number. Binary or hexadecimal numbers always have a 0b or 0h prefix, respectively.

Zero, not the letter O, followed by b or h.

0b *binaryNumber*
0h *hexadecimalNumber*

A binary number can have up to 32 digits. A hexadecimal number can have up to 8.

Without a prefix, *integer1* is treated as decimal (base 10). The result is displayed in binary, regardless of the Base mode.

If you enter a decimal integer that is too large for a signed, 32-bit binary form, a symmetric modulo operation is used to bring the value into the appropriate range.

`256▶Bin` ENTER	`0b100000000`
`0h1F▶Bin` ENTER	`0b11111`

BldData CATALOG

BldData [*dataVar*]

Creates data variable *dataVar* based on the information used to plot the current graph. **BldData** is valid in all graphing modes.

If *dataVar* is omitted, the data is stored in the system variable sysData.

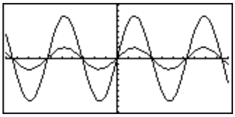
Note: The first time you start the Data/Matrix Editor after using **BldData**, *dataVar* or sysData (depending on the argument you used with **BldData**) is set as the current data variable.

The incremental values used for any independent variables (x in the example to the right) are calculated according to the Window variable values.

For information about the increments used to evaluate a graph, refer to the module that describes that graphing mode.

In function graphing mode and Radian angle mode:

```
8*sin(x)→y1(x) [ENTER]          Done
2*sin(x)→y2(x) [ENTER]          Done
ZoomStd [ENTER]
```



[HOME]
[◆] [CALC HOME]

```
BldData [ENTER]                 Done
[APPS] 6 [ENTER]
```

DATA	x	y1	y2
	c1	c2	c3
1	-10.	4.3522	1.088
2	-9.832	3.168	.792
3	-9.664	1.8945	.47363
4	-9.496	.56769	.14192

3D graphing mode has two independent variables. In the sample data to the right, notice that x remains constant as y increments through its range of values.

Then, x increments to its next value and y again increments through its range. This pattern continues until x has incremented through its range.

Note: The following sample data is from a 3D graph.

DATA	x	y	z1
	c1	c2	c3
1	-10.	-10.	0.
2	-10.	-8.571	5.8309
3	-10.	-7.143	8.9706
4	-10.	-5.714	9.8677

ceiling() MATH/Number menu

ceiling(*expression1***)** ⇒ *integer*

Returns the nearest integer that is ≥ the argument.

The argument can be a real or a complex number.

Note: See also **floor()**.

```
ceiling(0.456) [ENTER]          1.
```

---

ceiling(*list1***)** ⇒ *list*
ceiling(*matrix1***)** ⇒ *matrix*

Returns a list or matrix of the ceiling of each element.

```
ceiling({-3.1,1,2.5}) [ENTER]
                         {-3. 1 3.}
ceiling([0,-3.2i;1.3,4]) [ENTER]
                         [0   -3.·i]
                         [2.   4   ]
```

cFactor() MATH/Algebra/Complex menu

cFactor(*expression1*[**,** *var*]**)** ⇒ *expression*
cFactor(*list1*[**,***var*]**)** ⇒ *list*
cFactor(*matrix1*[**,***var*]**)** ⇒ *matrix*

cFactor(*expression1***)** returns *expression1* factored with respect to all of its variables over a common denominator.

expression1 is factored as much as possible toward linear rational factors even if this introduces new non-real numbers. This alternative is appropriate if you want factorization with respect to more than one variable.

```
cFactor(a^3*x^2+a*x^2+a^3+a)
[ENTER]
  a·(a + ⁻i)·(a + i)·(x + ⁻i)·(x + i)

cFactor(x^2+4/9) [ENTER]
  (3·x + ⁻2·i)·(3·x + 2·i)
  ────────────────────────
             9

cFactor(x^2+3) [ENTER]        x² + 3

cFactor(x^2+a) [ENTER]        x² + a
```

cFactor(*expression1,var***)** returns *expression1* factored with respect to variable *var*.

expression1 is factored as much as possible toward factors that are linear in *var*, with perhaps non-real constants, even if it introduces irrational constants or subexpressions that are irrational in other variables.

The factors and their terms are sorted with *var* as the main variable. Similar powers of *var* are collected in each factor. Include *var* if factorization is needed with respect to only that variable and you are willing to accept irrational expressions in any other variables to increase factorization with respect to *var*. There might be some incidental factoring with respect to other variables.

```
cFactor(a^3*x^2+a*x^2+a^3+a,x)
[ENTER]
  a·(a² + 1)·(x + ⁻i)·(x + i)

cFactor(x^2+3,x) [ENTER]
  (x + √3·i)·(x + ⁻√3·i)

cFactor(x^2+a,x) [ENTER]
  (x + √a·⁻i)·(x + √a·i)
```

For the AUTO setting of the Exact/Approx mode, including *var* also permits approximation with floating-point coefficients where irrational coefficients cannot be explicitly expressed concisely in terms of the built-in functions. Even when there is only one variable, including *var* might yield more complete factorization.

Note: See also **factor()**.

```
cFactor(x^5+4x^4+5x^3-6x-3)
[ENTER]
  x⁵ + 4·x⁴ + 5·x³ -6·x-3

cFactor(ans(1),x) [ENTER]
  (x -.965)·(x +.612)·(x + 2.13)·
  (x + 1.11 - 1.07·i)·
  (x + 1.11 + 1.07·i)
```

char() MATH/String menu

char(*integer***)** ⇒ *character*

Returns a character string containing the character numbered *integer* from the TI-89 Titanium/Voyage™ 200 character set. See Appendix B for a complete listing of character codes. The valid range for *integer* is 0–255.

```
char(38) [ENTER]        "&"

char(65) [ENTER]        "A"
```

checkTmr() CATALOG

checkTmr(*starttime***)** ⇒ *integer*

Returns an integer representing the number of seconds that have elapsed since a timer was started. *starttime* is an integer returned from the **startTmr()** function.

You can also use a list or matrix of *starttime* integers. Valid *starttime* integers must fall between 0 and the current time of the clock. You can run multiple timers simultaneously.

Note: See also **startTmr()** and **timeCnv()**.

```
startTmr() [ENTER]         148083315

checkTmr(148083315)        34

startTmr()→Timer1
⋮
startTmr()→Timer2
⋮
checkTmr(Timer1)→Timer1Value
⋮
checkTmr(Timer2)→Timer2Value
```

Circle CATALOG

Circle *x*, *y*, *r* [, *drawMode*]

Draws a circle with its center at window coordinates (*x, y*) and with a radius of *r*.

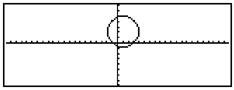
x, *y*, and *r* must be real values.

If *drawMode* = 1, draws the circle (default).
If *drawMode* = 0, turns off the circle.
If *drawMode* = -1, inverts pixels along the circle.

Note: Regraphing erases all drawn items. See also **PxlCrcl**.

In a ZoomSqr viewing window:

```
ZoomSqr:Circle 1,2,3 [ENTER]
```



ClockOff CATALOG

ClockOff

Turns the clock OFF.

ClockOn CATALOG

ClockOn

Turns the clock ON.

ClrDraw CATALOG

ClrDraw

Clears the Graph screen and resets the Smart Graph feature so that the next time the Graph screen is displayed, the graph will be redrawn.

While viewing the Graph screen, you can clear all drawn items (such as lines and points) by pressing [F4] (ReGraph) or pressing:
[2nd] [F6]
[F6]
and selecting 1:ClrDraw.

ClrErr CATALOG

ClrErr

Clears the error status. It sets errornum to zero and clears the internal error context variables.

The **Else** clause of the **Try...EndTry** in the program should use **ClrErr** or **PassErr**. If the error is to be processed or ignored, use **ClrErr**. If what to do with the error is not known, use **PassErr** to send it to the next error handler. If there are no more pending **Try...EndTry** error handlers, the error dialog box will be displayed as normal.

Note: See also **PassErr** and **Try**.

Program listing:

```
:clearerr()
:Prgm
:PlotsOff:FnOff:ZoomStd
:For i,0,238
:Δx*i+xmin→xcord
: Try
:  PtOn xcord,ln(xcord)
: Else
:  If errornum=800 or
     errornum=260 Then
:   ClrErr © clear the error
:  Else
:   PassErr © pass on any other
     error
:  EndIf
: EndTry
:EndFor
:EndPrgm
```

ClrGraph CATALOG

ClrGraph

Clears any functions or expressions that were graphed with the **Graph** command or were created with the **Table** command. (See **Graph** or **Table**.)

Any previously selected Y= functions will be graphed the next time that the graph is displayed.

ClrHome CATALOG

ClrHome

Clears all items stored in the **entry()** and **ans()** Home screen history area. Does not clear the current entry line.

While viewing the Home screen, you can clear the history area by pressing [F1] and selecting 8:Clear Home.

For functions such as **solve()** that return arbitrary constants or integers (@1, @2, etc.), **ClrHome** resets the suffix to 1.

ClrIO CATALOG

ClrIO

Clears the Program I/O screen.

ClrTable CATALOG

ClrTable

Clears all table values. Applies only to the ASK setting on the Table Setup dialog box.

While viewing the Table screen in Ask mode, you can clear the values by pressing [F1] and selecting 8:Clear Table.

colDim() MATH/Matrix/Dimensions menu

colDim(*matrix***)** ⇒ *expression*

Returns the number of columns contained in *matrix*.

Note: See also **rowDim()**.

```
colDim([0,1,2;3,4,5]) [ENTER]        3
```

colNorm() MATH/Matrix/Norms menu

colNorm(*matrix***)** ⇒ *expression*

Returns the maximum of the sums of the absolute values of the elements in the columns in *matrix*.

Note: Undefined matrix elements are not allowed. See also **rowNorm()**.

```
[1,⁻2,3;4,5,⁻6]→mat [ENTER]
                        [1  ⁻2   3]
                        [4   5  ⁻6]
colNorm(mat) [ENTER]                 9
```

comDenom() MATH/Algebra menu

comDenom(*expression1*[,*var*]**)** ⇒ *expression*
comDenom(*list1*[,*var*]**)** ⇒ *list*
comDenom(*matrix1*[,*var*]**)** ⇒ *matrix*

comDenom(*expression1***)** returns a reduced ratio of a fully expanded numerator over a fully expanded denominator.

```
comDenom((y^2+y)/(x+1)^2+y^2+y) [ENTER]
```

$$\blacksquare\ \mathrm{comDenom}\left(\frac{y^2+y}{(x+1)^2}+y^2+y\right)$$
$$\frac{x^2\cdot y^2+x^2\cdot y+2\cdot x\cdot y^2+2\cdot\blacktriangleright}{x^2+2\cdot x+1}$$

comDenom(*expression1,var***)** returns a reduced ratio of numerator and denominator expanded with respect to *var*. The terms and their factors are sorted with *var* as the main variable. Similar powers of *var* are collected. There might be some incidental factoring of the collected coefficients. Compared to omitting *var*, this often saves time, memory, and screen space, while making the expression more comprehensible. It also makes subsequent operations on the result faster and less likely to exhaust memory.

```
comDenom((y^2+y)/(x+1)
^2+y^2+y,x) [ENTER]
```

$$\blacksquare\ \mathrm{comDenom}\left(\frac{y^2+y}{(x+1)^2}+y^2+y,\blacktriangleright\right.$$
$$\frac{x^2\cdot y\cdot(y+1)+2\cdot x\cdot y\cdot(y+1)\blacktriangleright}{x^2+2\cdot x+1}$$

```
comDenom((y^2+y)/(x+1)
^2+y^2+y,y) [ENTER]
```

$$\blacksquare\ \mathrm{comDenom}\left(\frac{y^2+y}{(x+1)^2}+y^2+y,\blacktriangleright\right.$$
$$\frac{y^2\cdot(x^2+2\cdot x+2)+y\cdot(x^2+\blacktriangleright}{x^2+2\cdot x+1}$$

If *var* does not occur in *expression1*, **comDenom(***expression1,var***)** returns a reduced ratio of an unexpanded numerator over an unexpanded denominator. Such results usually save even more time, memory, and screen space. Such partially factored results also make subsequent operations on the result much faster and much less likely to exhaust memory.

```
comDenom(exprn,abc)→comden
(exprn) [ENTER]                 Done
comden((y^2+y)/(x+1)^2+y^2+y)
[ENTER]
```

$$\blacksquare\ \mathrm{comden}\left(\frac{y^2+y}{(x+1)^2}+y^2+y\right)$$
$$\frac{(x^2+2\cdot x+2)\cdot y\cdot(y+1)}{(x+1)^2}$$

Even when there is no denominator, the **comden** function is often a fast way to achieve partial factorization if **factor()** is too slow or if it exhausts memory.

```
comden(1234x^2*(y^3-y)+2468x
*(y^2-1)) [ENTER]
      1234·x·(x·y + 2)·(y² - 1)
```

Hint: Enter this **comden()** function definition and routinely try it as an alternative to **comDenom()** and **factor()**.

conj() MATH/Complex menu

conj(*expression1***)** ⇒ *expression*
conj(*list1***)** ⇒ *list*
conj(*matrix1***)** ⇒ *matrix*

Returns the complex conjugate of the argument.

Note: All undefined variables are treated as real variables.

```
conj(1+2i) [ENTER]                    1 - 2·i
conj([2,1-3i;⁻i,⁻7]) [ENTER]
```

$$\begin{bmatrix}2 & 1+3\cdot i\\ i & ^-7\end{bmatrix}$$

```
conj(z)                               z
conj(x+iy)                       x + ⁻i·y
```

CopyVar CATALOG

CopyVar *var1*, *var2*

Copies the contents of variable *var1* to *var2*. If *var2* does not exist, **CopyVar** creates it.

Note: **CopyVar** is similar to the store instruction (→) when you are copying an expression, list, matrix, or character string except that no simplification takes place when using **CopyVar**. You must use **CopyVar** with non-algebraic variable types such as Pic and GDB variables.

x+y→a [ENTER]	x + y
10→x [ENTER]	10
CopyVar a,b [ENTER]	Done
a→c [ENTER]	y + 10
DelVar x [ENTER]	Done
b [ENTER]	x + y
c [ENTER]	y + 10

cos() [2nd] [COS] key [COS] key

cos(*expression1***)** ⇒ *expression*
cos(*list1***)** ⇒ *list*

cos(*expression1***)** returns the cosine of the argument as an expression.

cos(*list1***)** returns a list of the cosines of all elements in *list1*.

Note: The argument is interpreted as either a degree or radian angle, according to the current angle mode setting. You can use ° or ʳ to override the angle mode temporarily.

In Degree angle mode:

cos((π/4)ʳ) [ENTER]	$\frac{\sqrt{2}}{2}$
cos(45) [ENTER]	$\frac{\sqrt{2}}{2}$
cos({0,60,90}) [ENTER]	{1 1/2 0}

In Radian angle mode:

cos(π/4) [ENTER]	$\frac{\sqrt{2}}{2}$
cos(45°) [ENTER]	$\frac{\sqrt{2}}{2}$

cos(*squareMatrix1***)** ⇒ *squareMatrix*

Returns the matrix cosine of *squareMatrix1*. This is *not* the same as calculating the cosine of each element.

When a scalar function f(A) operates on *squareMatrix1* (A), the result is calculated by the algorithm:

1. Compute the eigenvalues (λ_i) and eigenvectors (V_i) of A.

   squareMatrix1 must be diagonalizable. Also, it cannot have symbolic variables that have not been assigned a value.

2. Form the matrices:

$$B = \begin{bmatrix} \lambda_1 & 0 & \dots & 0 \\ 0 & \lambda_2 & \dots & 0 \\ 0 & 0 & \dots & 0 \\ 0 & 0 & \dots & \lambda_n \end{bmatrix} \text{ and } X = [V_1, V_2, \dots, V_n]$$

3. Then $A = X B X^{-1}$ and $f(A) = X f(B) X^{-1}$. For example, $\cos(A) = X \cos(B) X^{-1}$ where:

cos (B) =

All computations are performed using floating-point arithmetic.

In Radian angle mode:

cos([1,5,3;4,2,1;6,⁻2,1]) [ENTER]

$$\begin{bmatrix} .212\ldots & .205\ldots & .121\ldots \\ .160\ldots & .259\ldots & .037\ldots \\ .248\ldots & ^{-}.090\ldots & .218\ldots \end{bmatrix}$$

$\cos^{-1}()$ — [◆] [COS⁻¹] key — [2nd] [COS⁻¹] key

cos⁻¹ **(***expression1***)** ⇒ *expression*
cos⁻¹ **(***list1***)** ⇒ *list*

cos⁻¹ **(***expression1***)** returns the angle whose cosine is *expression1* as an expression.

cos⁻¹ **(***list1***)** returns a list of the inverse cosines of each element of *list1*.

Note: The result is returned as either a degree or radian angle, according to the current angle mode setting.

In Degree angle mode:

```
cos⁻¹(1) [ENTER]                    0
```

In Radian angle mode:

```
cos⁻¹({0,.2,.5}) [ENTER]
                 {π/2  1.369...  1.047...}
```

---

cos⁻¹**(***squareMatrix1***)** ⇒ *squareMatrix*

Returns the matrix inverse cosine of *squareMatrix1*. This is *not* the same as calculating the inverse cosine of each element. For information about the calculation method, refer to **cos()**.

squareMatrix1 must be diagonalizable. The result always contains floating-point numbers.

In Radian angle mode and Rectangular complex format mode:

```
cos⁻¹([1,5,3;4,2,1;6,⁻2,1])
[ENTER]
[1.734...+.064...•i    ⁻1.490...+2.105...•i  ...]
[⁻.725...+1.515...•i   .623...+.778...•i     ...]
[⁻2.083...+2.632...•i  1.790...−1.271...•i   ...]
```

cosh() — MATH/Hyperbolic menu

cosh(*expression1***)** ⇒ *expression*
cosh(*list1***)** ⇒ *list*

cosh (*expression1***)** returns the hyperbolic cosine of the argument as an expression.

cosh (*list1***)** returns a list of the hyperbolic cosines of each element of *list1*.

```
cosh(1.2) [ENTER]                1.810...
cosh({0,1.2}) [ENTER]       {1   1.810...}
```

---

cosh(*squareMatrix1***)** ⇒ *squareMatrix*

Returns the matrix hyperbolic cosine of *squareMatrix1*. This is *not* the same as calculating the hyperbolic cosine of each element. For information about the calculation method, refer to **cos()**.

squareMatrix1 must be diagonalizable. The result always contains floating-point numbers.

In Radian angle mode:

```
cosh([1,5,3;4,2,1;6,⁻2,1])
[ENTER]
[421.255  253.909  216.905]
[327.635  255.301  202.958]
[226.297  216.623  167.628]
```

$\cosh^{-1}()$ — MATH/Hyperbolic menu

cosh⁻¹ **(***expression1***)** ⇒ *expression*
cosh⁻¹ **(***list1***)** ⇒ *list*

cosh⁻¹ **(***expression1***)** returns the inverse hyperbolic cosine of the argument as an expression.

cosh⁻¹ **(***list1***)** returns a list of the inverse hyperbolic cosines of each element of *list1*.

```
cosh⁻¹(1) [ENTER]                       0
cosh⁻¹({1,2.1,3}) [ENTER]
              {0  1.372...  cosh⁻¹(3)}
```

---

cosh⁻¹**(***squareMatrix1***)** ⇒ *squareMatrix*

Returns the matrix inverse hyperbolic cosine of *squareMatrix1*. This is *not* the same as calculating the inverse hyperbolic cosine of each element. For information about the calculation method, refer to **cos()**.

squareMatrix1 must be diagonalizable. The result always contains floating-point numbers.

In Radian angle mode and Rectangular complex format mode:

```
cosh⁻¹([1,5,3;4,2,1;6,⁻2,1])
[ENTER]
[2.525...+1.734...•i    ⁻.009...−1.490...•i  ...]
[.486...−.725...•i      1.662...+.623...•i   ...]
[⁻.322...−2.083...•i    1.267...+1.790...•i  ...]
```

cot() — MATH/Trig menu

cot(*expression1***)** ⇒ *expression*
cot(*list1***)** ⇒ *list*

Returns the cotangent of *expression1* or returns a list of the cotangents of all elements in *list1*.

Note: The argument is interpreted as either a degree or radian angle, according to the current angle mode.

```
In Degree angle mode:

cot(45) [ENTER]                         1

In Radian angle mode:

cot({1,2.1,3}) [ENTER]
              {1/tan(1)  -.584…  1/tan(3)}
```

cot^{-1}() — MATH/Trig menu

cot^{-1}(*expression1***)** ⇒ *expression*
cot^{-1}(*list1***)** ⇒ *list*

Returns the angle whose cotangent is *expression1* or returns a list containing the inverse cotangents of each element of *list1*.

Note: The result is returned as either a degree or radian angle, according to the current angle mode.

```
In Degree angle mode:

cot^-1(1) [ENTER]                       45

In Radian angle mode:

cot^-1(1) [ENTER]                       π/4
```

coth() — MATH/Hyperbolic menu

coth(*expression1***)** ⇒ *expression*
cot(*list1***)** ⇒ *list*

Returns the hyperbolic cotangent of *expression1* or returns a list of the hyperbolic cotangents of all elements of *list1*.

```
coth(1.2) [ENTER]                       1.199…

coth({1,3.2}) [ENTER]
                    {1/tanh(1)  1.003…}
```

coth^{-1}() — MATH/Hyperbolic menu

coth^{-1}(*expression1***)** ⇒ *expression*
coth^{-1}(*list1***)** ⇒ *list*

Returns the inverse hyperbolic cotangent of *expression1* or returns a list containing the inverse hyperbolic cotangents of each element of *list1*.

```
coth^-1(3.5) [ENTER]                    .293…

coth^-1({-2,2.1,6}) [ENTER]
          {-ln(3)/2  .518…  ln(7/5)/2}
```

crossP() — MATH/Matrix/Vector ops menu

crossP(*list1*, *list2***)** ⇒ *list*

Returns the cross product of *list1* and *list2* as a list.

list1 and *list2* must have equal dimension, and the dimension must be either 2 or 3.

```
crossP({a1,b1},{a2,b2}) [ENTER]
                 {0  0  a1·b2-a2·b1}

crossP({0.1,2.2,-5},{1,-.5,0})
[ENTER]
                  {-2.5  -5.  -2.25}
```

---

crossP(*vector1*, *vector2***)** ⇒ *vector*

Returns a row or column vector (depending on the arguments) that is the cross product of *vector1* and *vector2*.

Both *vector1* and *vector2* must be row vectors, or both must be column vectors. Both vectors must have equal dimension, and the dimension must be either 2 or 3.

```
crossP([1,2,3],[4,5,6]) [ENTER]
                          [-3  6  -3]

crossP([1,2],[3,4]) [ENTER]
                           [0  0  -2]
```

csc() — MATH/Trig menu

csc(*expression1***)** ⇒ *expression*
csc(*list1***)** ⇒ *list*

Returns the cosecant of *expression1* or returns a list containing the cosecants of all elements in *list1*.

In Degree angle mode:

csc(π/4) [ENTER] $\frac{1}{\sin(\frac{\pi}{4})}$

In Radian angle mode:

csc({1,π/2,π/3}) [ENTER] $\{\frac{1}{\sin(1)}\ 1\ \frac{2\cdot\sqrt{3}}{3}\}$

csc⁻¹() — MATH/Trig menu

csc^{-1}(*expression1***)** ⇒ *expression*
csc^{-1}(*list1***)** ⇒ *list*

Returns the angle whose cosecant is *expression1* or returns a list containing the inverse cosecants of each element of *list1*.

Note: The result is returned as either a degree or radian angle, according to the current angle mode.

In Degree angle mode:

csc^{-1}(1) [ENTER] 90

In Radian angle mode:

csc^{-1}({1,4,6}) [ENTER]

$\{\frac{\pi}{2}\ \sin^{-1}(1/4)\ \sin^{-1}(1/6)\}$

csch() — MATH/Hyperbolic menu

csch(*expression1***)** ⇒ *expression*
csch(*list1***)** ⇒ *list*

Returns the hyperbolic cosecant of *expression1* or returns a list of the hyperbolic cosecants of all elements of *list1*.

csch(3) [ENTER] $\frac{1}{\sinh(3)}$

csch({1,2.1,4}) [ENTER]

$\{\frac{1}{\sinh(1)}\ .248\ldots\ \frac{1}{\sinh(4)}\}$

csch⁻¹() — MATH/Hyperbolic menu

csch^{-1}(*expression1***)** ⇒ *expression*
csch^{-1}(*list1***)** ⇒ *list*

Returns the inverse hyperbolic cosecant of *expression1* or returns a list containing the inverse hyperbolic cosecants of each element of *list1*.

csch^{-1}(1) [ENTER] $\sinh^{-1}(1)$

csch^{-1}({1,2.1,3}) [ENTER]

$\{\sinh^{-1}(1)\ .459\ldots\ \sinh^{-1}(1/3)\}$

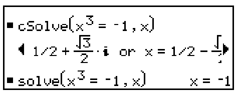
cSolve() — MATH/Algebra/Complex menu

cSolve(*equation***,** *var***)** ⇒ *Boolean expression*

Returns candidate complex solutions of an equation for *var*. The goal is to produce candidates for all real and non-real solutions. Even if *equation* is real, **cSolve()** allows non-real results in real mode.

Although the TI-89 Titanium/Voyage™ 200 processes all undefined variables that do not end with an underscore (_) as if they were real, **cSolve()** can solve polynomial equations for complex solutions.

cSolve(x^3=⁻1,x) [ENTER]
solve(x^3=⁻1,x) [ENTER]



cSolve() temporarily sets the domain to complex during the solution even if the current domain is real. In the complex domain, fractional powers having odd denominators use the principal rather than the real branch. Consequently, solutions from **solve()** to equations involving such fractional powers are not necessarily a subset of those from **cSolve()**.

```
cSolve(x^(1/3)=⁻1,x) [ENTER]  false

solve(x^(1/3)=⁻1,x) [ENTER]   x = ⁻1
```

cSolve() starts with exact symbolic methods. Except in EXACT mode, **cSolve()** also uses iterative approximate complex polynomial factoring, if necessary.

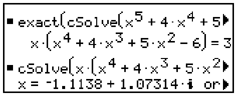
Note: See also **cZeros()**, **solve()**, and **zeros()**.

Note: If *equation* is non-polynomial with functions such as **abs()**, **angle()**, **conj()**, **real()**, or **imag()**, you should place an underscore _ [◆] [_] [2nd] [_]) at the end of *var*. By default, a variable is treated as a real value.
If you use *var_*, the variable is treated as complex.

You should also use *var_* for any other variables in *equation* that might have unreal values. Otherwise, you may receive unexpected results.

Display Digits mode in Fix 2:

```
exact(cSolve(x^5+4x^4+5x
  ^3-6x-3=0,x)) [ENTER]
cSolve(ans(1),x) [ENTER]
```



z is treated as real:

```
cSolve(conj(z)=1+i,z) [ENTER]
                         z=1+i
```

z_ is treated as complex:

```
cSolve(conj(z_)=1+i,z_) [ENTER]
                         z_=1−i
```

cSolve(*equation1* **and** *equation2* [**and** …], **{***varOrGuess1***,** *varOrGuess2* [**,** …]**})** ⇒ *Boolean expression*

Returns candidate complex solutions to the simultaneous algebraic equations, where each *varOrGuess* specifies a variable that you want to solve for.

Optionally, you can specify an initial guess for a variable. Each *varOrGuess* must have the form:

variable
– or –
variable = *real or non-real number*

For example, x is valid and so is x=3+*i*.

If all of the equations are polynomials and if you do NOT specify any initial guesses, **cSolve()** uses the lexical Gröbner/Buchberger elimination method to attempt to determine **all** complex solutions.

Complex solutions can include both real and non-real solutions, as in the example to the right.

Note: The following examples use an underscore _ [◆] [_] [2nd] [_] so that the variables will be treated as complex.

```
cSolve(u_∗v_−u_=v_ and
v_^2=⁻u_,{u_,v_}) [ENTER]
```

$$u\_=1/2 + \frac{\sqrt{3}}{2}\cdot i \text{ and } v\_=1/2 - \frac{\sqrt{3}}{2}\cdot i$$
$$\text{or } u\_=1/2 - \frac{\sqrt{3}}{2}\cdot i \text{ and } v\_=1/2 + \frac{\sqrt{3}}{2}\cdot i$$
$$\text{or } u\_=0 \text{ and } v\_=0$$

Simultaneous *polynomial* equations can have extra variables that have no values, but represent given numeric values that could be substituted later.

```
cSolve(u_*v_-u_=c_*v_ and
v_^2=-u_,{u_,v_}) [ENTER]
```

$$u\_=\frac{^-(\sqrt{1-4\cdot c\_}+1)^2}{4} \text{ and } v\_=\frac{\sqrt{1-4\cdot c\_}+1}{2}$$

or

$$u\_=\frac{^-(\sqrt{1-4\cdot c\_}-1)^2}{4} \text{ and } v\_=\frac{^-(\sqrt{1-4\cdot c\_}-1)}{2}$$

or u_=0 and v_=0

You can also include solution variables that do not appear in the equations. These solutions show how families of solutions might contain arbitrary constants of the form @*k*, where *k* is an integer suffix from 1 through 255. The suffix resets to 1 when you use **ClrHome** or [F1] 8:Clear Home.

For polynomial systems, computation time or memory exhaustion may depend strongly on the order in which you list solution variables. If your initial choice exhausts memory or your patience, try rearranging the variables in the equations and/or *varOrGuess* list.

```
cSolve(u_*v_-u_=v_ and
v_^2=-u_,{u_,v_,w_}) [ENTER]
```

$$u\_=1/2+\frac{\sqrt{3}}{2}\cdot i \text{ and } v\_=1/2-\frac{\sqrt{3}}{2}\cdot i$$

and w_=@1

or

$$u\_=1/2-\frac{\sqrt{3}}{2}\cdot i \text{ and } v\_=1/2+\frac{\sqrt{3}}{2}\cdot i$$

and w_=@1

or u_=0 and v_=0 and w_=@1

If you do not include any guesses and if any equation is non-polynomial in any variable but all equations are linear in all solution variables, **cSolve()** uses Gaussian elimination to attempt to determine all solutions.

```
cSolve(u_+v_=e^(w_) and u_-v_=
i, {u_,v_}) [ENTER]
```

$$u\_=\frac{e^{w\_}}{2}+1/2\cdot i \text{ and } v\_=\frac{e^{w\_}-i}{2}$$

If a system is neither polynomial in all of its variables nor linear in its solution variables, **cSolve()** determines at most one solution using an approximate iterative method. To do so, the number of solution variables must equal the number of equations, and all other variables in the equations must simplify to numbers.

```
cSolve(e^(z_)=w_ and w_=z_^2,
{w_,z_}) [ENTER]
         w_=.494… and z_=-.703…
```

A non-real guess is often necessary to determine a non-real solution. For convergence, a guess might have to be rather close to a solution.

```
cSolve(e^(z_)=w_ and w_=z_^2,
{w_,z_=1+i}) [ENTER]
         w_=.149… + 4.891…•i and
            z_=1.588… + 1.540…•i
```

CubicReg MATH/Statistics/Regressions menu

CubicReg *list1*, *list2*[, [*list3*] [, *list4*, *list5*]]

Calculates the cubic polynomial regression and updates all the statistics variables.

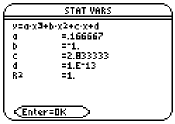
All the lists must have equal dimensions except for *list5*.

list1 represents xlist.
list2 represents ylist.
list3 represents frequency.
list4 represents category codes.
list5 represents category include list.

Note: *list1* through *list4* must be a variable name or c1–c99 (columns in the last data variable shown in the Data/Matrix Editor). *list5* does not have to be a variable name and cannot be c1–c99 .

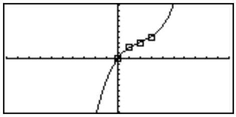
In function graphing mode.

```
{0,1,2,3}→L1 [ENTER]          {0 1 2 3}
{0,2,3,4}→L2 [ENTER]          {0 2 3 4}
CubicReg L1,L2 [ENTER]             Done
ShowStat [ENTER]
```




```
[ENTER]
regeq(x)→y1(x) [ENTER]             Done
NewPlot 1,1,L1,L2 [ENTER]          Done
```

[◆] [GRAPH]



cumSum() MATH/List menu

cumSum(*list1***)** ⇒ *list*

Returns a list of the cumulative sums of the elements in *list1*, starting at element 1.

```
cumSum({1,2,3,4}) [ENTER]
                              {1 3 6 10}
```

---

cumSum(*matrix1***)** ⇒ *matrix*

Returns a matrix of the cumulative sums of the elements in *matrix1*. Each element is the cumulative sum of the column from top to bottom.

[1,2;3,4;5,6]→m1 [ENTER] $\begin{bmatrix}1 & 2\\3 & 4\\5 & 6\end{bmatrix}$

cumSum(m1) [ENTER] $\begin{bmatrix}1 & 2\\4 & 6\\9 & 12\end{bmatrix}$

CustmOff CATALOG

CustmOff

Removes a custom toolbar.

CustmOn and **CustmOff** enable a program to control a custom toolbar. Manually, you can press [2nd] [CUSTOM] to toggle a custom toolbar on and off. Also, a custom toolbar is removed automatically when you change applications.

See **Custom** program listing example.

CustmOn CATALOG

CustmOn

Activates a custom toolbar that has already been set up in a **Custom...EndCustm** block.

CustmOn and **CustmOff** enable a program to control a custom toolbar. Manually, you can press [2nd] [CUSTOM] to toggle a custom toolbar on and off.

See **Custom** program listing example.

Custom [2nd] [CUSTOM] key

Custom
block
EndCustm

Sets up a toolbar that is activated when you press [2nd] [CUSTOM]. It is very similar to the **ToolBar** instruction except that Title and Item statements cannot have labels.

block can be either a single statement or a series of statements separated with the ":" character.

Note: [2nd] [CUSTOM] acts as a toggle. The first instance invokes the menu, and the second instance removes the menu. The menu is removed also when you change applications.

Program listing:

```
:Test()
:Prgm
:Custom
:Title  "Lists"
:Item   "List1"
:Item   "Scores"
:Item   "L3"
:Title  "Fractions"
:Item   "f(x)"
:Item   "h(x)"
:Title  "Graph"
:EndCustm
:EndPrgm
```

Cycle CATALOG

Cycle

Transfers program control immediately to the next iteration of the current loop (**For**, **While**, or **Loop**).

Cycle is not allowed outside the three looping structures (**For**, **While**, or **Loop**).

Program listing:

```
:© Sum the integers from 1 to
  100 skipping 50.
:0→temp
:For i,1,100,1
:If i=50
:Cycle
:temp+i→temp
:EndFor
:Disp temp
```

Contents of temp after execution: 5000

CyclePic CATALOG

CyclePic *picNameString*, *n* [, [*wait*] , [*cycles*], [*direction*]]

Displays all the PIC variables specified and at the specified interval. The user has optional control over the time between pictures, the number of times to cycle through the pictures, and the direction to go, circular or forward and backwards.

direction is 1 for circular or ⁻1 for forward and backwards. Default = 1.

1. Save three pics named pic1, pic2, and pic3.
2. Enter: CyclePic "pic",3,.5,4,⁻1
3. 3.The three pictures (3) will be displayed au

▸Cylind MATH/Matrix/Vector ops menu

vector▸**Cylind**

Displays the row or column vector in cylindrical form [r∠θ, z].

vector must have exactly three elements. It can be either a row or a column.

[2,2,3] ▸Cylind [ENTER]
$[2\cdot\sqrt{2}\ \angle\frac{\pi}{4}\quad 3]$

cZeros() MATH/Algebra/Complex menu

cZeros(*expression*, *var***)** ⇒ *list*

Returns a list of candidate real and non-real values of *var* that make *expression*=0. **cZeros()** does this by computing **exp▶list(cSolve(***expression*=0,*var***)**,*var***)**. Otherwise, **cZeros()** is similar to **zeros()**.

Note: See also **cSolve()**, **solve()**, and **zeros()**.

Note: If *expression* is non-polynomial with functions such as **abs()**, **angle()**, **conj()**, **real()**, or **imag()**, you should place an underscore _ ([◆] [_], [2nd] [_]) at the end of *var*. By default, a variable is treated as a real value. If you use *var_*, the variable is treated as complex.

You should also use *var_* for any other variables in *expression* that might have unreal values. Otherwise, you may receive unexpected results.

Display Digits mode in Fix 3:

```
cZeros(x^5+4x^4+5x^3-6x-3,x) [ENTER]
    {⁻2.125  ⁻.612  .965
     ⁻1.114 - 1.073·i
     ⁻1.114 + 1.073·i}
```

z is treated as real:

```
cZeros(conj(z)-1-i,z) [ENTER]
                      {1+i}
```

z_ is treated as complex:

```
cZeros(conj(z_)-1-i,z_) [ENTER]
                      {1-i}
```

---

cZeros({*expression1*, *expression2* [, …] **}**, **{***varOrGuess1*,*varOrGuess2* [, …] **})** ⇒ *matrix*

Returns candidate positions where the expressions are zero simultaneously. Each *varOrGuess* specifies an unknown whose value you seek.

Optionally, you can specify an initial guess for a variable. Each *varOrGuess* must have the form:

variable
– or –
variable = *real or non-real number*

For example, x is valid and so is x=3+*i*.

If all of the expressions are polynomials and you do NOT specify any initial guesses, **cZeros()** uses the lexical Gröbner/Buchberger elimination method to attempt to determine **all** complex zeros.

Note: The following examples use an underscore _ ([◆] [_], [2nd] [_]) so that the variables will be treated as complex.

Complex zeros can include both real and non-real zeros, as in the example to the right.

Each row of the resulting matrix represents an alternate zero, with the components ordered the same as the *varOrGuess* list. To extract a row, index the matrix by [*row*].

```
cZeros({u_*v_-u_-v_,v_^2+u_},
{u_,v_}) [ENTER]
```

$$\begin{bmatrix} 1/2 - \frac{\sqrt{3}}{2}\cdot i & 1/2 + \frac{\sqrt{3}}{2}\cdot i \\ 1/2 + \frac{\sqrt{3}}{2}\cdot i & 1/2 - \frac{\sqrt{3}}{2}\cdot i \\ 0 & 0 \end{bmatrix}$$

Extract row 2:

```
ans(1)[2] [ENTER]
```

$$\begin{bmatrix} 1/2 + \frac{\sqrt{3}}{2}\cdot i & 1/2 - \frac{\sqrt{3}}{2}\cdot i \end{bmatrix}$$

Simultaneous *polynomials* can have extra variables that have no values, but represent given numeric values that could be substituted later.

```
cZeros({u_*v_-u_-(c_*v_),
v_^2+u_},{u_,v_}) [ENTER]
```

$$\begin{bmatrix} \frac{^-(\sqrt{1-4\cdot c\_}+1)^2}{4} & \frac{\sqrt{1-4\cdot c\_}+1}{2} \\ \frac{^-(\sqrt{1-4\cdot c\_}-1)^2}{4} & \frac{^-(\sqrt{1-4\cdot c\_}-1)}{2} \\ 0 & 0 \end{bmatrix}$$

You can also include unknown variables that do not appear in the expressions. These zeros show how families of zeros might contain arbitrary constants of the form @*k*, where *k* is an integer suffix from 1 through 255. The suffix resets to 1 when you use **ClrHome** or [F1] 8:Clear Home.

```
cZeros({u_*v_-u_-v_,v_^2+u_},
{u_,v_,w_}) [ENTER]
```

$$\begin{bmatrix} 1/2-\frac{\sqrt{3}}{2}\cdot i & 1/2+\frac{\sqrt{3}}{2}\cdot i & @1 \\ 1/2+\frac{\sqrt{3}}{2}\cdot i & 1/2-\frac{\sqrt{3}}{2}\cdot i & @1 \\ 0 & 0 & @1 \end{bmatrix}$$

For polynomial systems, computation time or memory exhaustion may depend strongly on the order in which you list unknowns. If your initial choice exhausts memory or your patience, try rearranging the variables in the expressions and/or *varOrGuess* list.

If you do not include any guesses and if any expression is non-polynomial in any variable but all expressions are linear in all unknowns, **cZeros()** uses Gaussian elimination to attempt to determine all zeros.

```
cZeros({u_+v_-e^(w_),u_-v_-i},
{u_,v_}) [ENTER]
```

$$\begin{bmatrix} \frac{e^{w\_}}{2}+1/2\cdot i & \frac{e^{w\_}-i}{2} \end{bmatrix}$$

If a system is neither polynomial in all of its variables nor linear in its unknowns, **cZeros()** determines at most one zero using an approximate iterative method. To do so, the number of unknowns must equal the number of expressions, and all other variables in the expressions must simplify to numbers.

```
cZeros({e^(z_)-w_,w_-z_^2},
{w_,z_}) [ENTER]
```

[.494… ⁻.703…]

A non-real guess is often necessary to determine a non-real zero. For convergence, a guess might have to be rather close to a zero.

```
cZeros({e^(z_)-w_,w_-z_^2},
{w_,z_=1+i}) [ENTER]
```

[.149…+4.89…·*i* 1.588…+1.540…·*i*]

d() [2nd] [*d*] key or MATH/Calculus menu

d*(expression1*, *var* [,*order*]**)** ⇒ *expression*
d*(list1*,*var* [,*order*]**)** ⇒ *list*
d*(matrix1*,*var* [,*order*]**)** ⇒ *matrix*

Returns the first derivative of *expression1* with respect to variable *var*. *expression1* can be a list or a matrix.

order, if included, must be an integer. If the order is less than zero, the result will be an anti-derivative.

***d*()** does not follow the normal evaluation mechanism of fully simplifying its arguments and then applying the function definition to these fully simplified arguments. Instead, ***d*()** performs the following steps:

1. Simplify the second argument only to the extent that it does not lead to a non-variable.
2. Simplify the first argument only to the extent that it does recall any stored value for the variable determined by step 1.
3. Determine the symbolic derivative of the result of step 2 with respect to the variable from step 1.
4. If the variable from step 1 has a stored value or a value specified by a "with" (|) operator, substitute that value into the result from step 3.

d(3x^3-x+7,x) [ENTER] $9x^2-1$

d(3x^3-x+7,x,2) [ENTER] $18\cdot x$

d(f(x)*g(x),x) [ENTER]

$$\frac{d}{dx}(f(x))\cdot g(x)+\frac{d}{dx}(g(x))\cdot f(x)$$

d(sin(f(x)),x) [ENTER]

$$\cos(f(x))\frac{d}{dx}(f(x))$$

d(x^3,x)|x=5 [ENTER] 75

d(d(x^2*y^3,x),y) [ENTER] $6\cdot y^2\cdot x$

d(x^2,x,⁻1) [ENTER] $\frac{x^3}{3}$

d({x^2,x^3,x^4},x) [ENTER]
$\{2\cdot x \quad 3\cdot x^2 \quad 4\cdot x^3\}$

dayOfWk() CATALOG

dayOfWk(*year,month,day***)** ⇒ *integer*

Returns an integer from 1 to 7, with each integer representing a day of the week. Use **dayOfWk()** to determine on which day of the week a particular date would occur.
Note: May not give accurate results for years prior to 1583 (pre-Gregorian calendar).

Enter the year as a four-digit integer. The month and day can be either one- or two-digit integers.

`dayOfWk(1948,9,6)` 2

Integer values:

```
1 = Sunday
2 = Monday
3 = Tuesday
4 = Wednesday
5 = Thursday
6 = Friday
7 = Saturday
```

▸DD MATH/Angle menu

number **▸DD** ⇒ *value*
list1 **▸DD** ⇒ *list*
matrix1 **▸DD** ⇒ *matrix*

Returns the decimal equivalent of the argument. The argument is a number, list, or matrix that is interpreted by the Mode setting in radians or degrees.

Note: ▸DD can also accept input in radians.

In Degree angle mode:

`1.5°▸DD` [ENTER] 1.5°

`45°22'14.3"▸DD` [ENTER] 45.370...°

`{45°22'14.3",60°0'0"}▸DD` [ENTER]
{45.370... 60}°

In Radian angle mode:

`1.5▸DD` [ENTER] 85.9°

▸Dec MATH/Base menu

integer1 **▸Dec** ⇒ *integer*

Converts *integer1* to a decimal (base 10) number. A binary or hexadecimal entry must always have a 0b or 0h prefix, respectively.

Zero, not the letter O, followed by b or h.
0b *binaryNumber*
0h *hexadecimalNumber*
A binary number can have up to 32 digits. A hexadecimal number can have up to 8.

Without a prefix, *integer1* is treated as decimal. The result is displayed in decimal, regardless of the Base mode.

`0b10011▸Dec` [ENTER] 19

`0h1F▸Dec` [ENTER] 31

Define CATALOG

Define *funcName***(***arg1Name, arg2Name, ...***) =** *expression*

Creates *funcName* as a user-defined function. You then can use *funcName***()**, just as you use built-in functions. The function evaluates *expression* using the supplied arguments and returns the result.

funcName cannot be the name of a system variable or built-in function.

The argument names are placeholders; you should not use those same names as arguments when you use the function.

Note: This form of **Define** is equivalent to executing the expression: *expression*→*funcName*(*arg1Name,arg2Name*). This command also can be used to define simple variables; for example, Define a=3.

```
Define g(x,y)=2x-3y [ENTER]              Done
g(1,2) [ENTER]                              ⁻4
1→a:2→b:g(a,b) [ENTER]                      ⁻4

Define h(x)=when(x<2,2x-3,
⁻2x+3) [ENTER]                            Done

h(⁻3) [ENTER]                               ⁻9
h(4) [ENTER]                                ⁻5

Define eigenvl(a)=
  cZeros(det(identity(dim(a)
  [1])-x*a),x) [ENTER]                    Done
eigenvl([⁻1,2;4,3]) [ENTER]
```

$$\left\{\frac{2\cdot\sqrt{3}-1}{11} \quad \frac{^-(2\cdot\sqrt{3}+1)}{11}\right\}$$

Define *funcName***(***arg1Name, arg2Name, ...***) = Func**
block
EndFunc

Is identical to the previous form of **Define**, except that in this form, the user-defined function *funcName***()** can execute a block of multiple statements.

block can be either a single statement or a series of statements separated with the ":" character. *block* also can include expressions and instructions (such as **If**, **Then**, **Else**, and **For**). This allows the function *funcName***()** to use the **Return** instruction to return a specific result.

Note: It is usually easier to author and edit this form of Function in the program editor rather than on the entry line.

```
Define g(x,y)=Func:If x>y Then
:Return x:Else:Return y:EndIf
:EndFunc [ENTER]                          Done

g(3,⁻7) [ENTER]                              3
```

Define *progName***(***arg1Name, arg2Name, ...***) = Prgm**
block
EndPrgm

Creates *progName* as a program or subprogram, but cannot return a result using **Return**. Can execute a block of multiple statements.

block can be either a single statement or a series of statements separated with the ":" character. *block* also can include expressions and instructions (such as **If**, **Then**, **Else**, and **For**) without restrictions.

Note: It is usually easier to author and edit a program block in the Program Editor rather than on the entry line.

```
Define listinpt()=prgm:Local
  n,i,str1,num:InputStr "Enter
  name of list",str1:Input "No.
  of elements",n:For
  i,1,n,1:Input "element
  "&string(i),num:
  num→#str1[i]:EndFor:EndPrgm
  [ENTER]
                                          Done

listinpt() [ENTER]Enter name of list
```

DelFold CATALOG

DelFold *folderName1*[**,** *folderName2*] [**,** *folderName3*] **...**

Deletes user-defined folders with the names *folderName1, folderName2,* etc. An error message is displayed if the folders contain any variables.

Note: You cannot delete the main folder.

```
NewFold games [ENTER]                     Done
```
(creates the folder games)

```
DelFold games [ENTER]                     Done
```
(deletes the folder games)

DelVar CATALOG

DelVar *var1*[**,** *var2*] [**,** *var3*] **...**

Deletes the specified variables from memory.

```
2→a [ENTER]                2
(a+2)^2 [ENTER]           16
DelVar a [ENTER]        Done
(a+2)^2 [ENTER]      (a + 2)²
```

deSolve() MATH/Calculus menu

deSolve(*1stOr2ndOrderOde***,** *independentVar***,** *dependentVar***)** ⇒ *a general solution*

Returns an equation that explicitly or implicitly specifies a general solution to the 1st- or 2nd-order ordinary differential equation (ODE). In the ODE:

- Use a prime symbol (', press [2nd] [′]) to denote the 1st derivative of the dependent variable with respect to the independent variable.
- Use two prime symbols to denote the corresponding second derivative.

The ' symbol is used for derivatives within **deSolve()** only. In other cases, use ***d*()**.

The general solution of a 1st-order equation contains an arbitrary constant of the form @*k*, where *k* is an integer suffix from 1 through 255. The suffix resets to 1 when you use **ClrHome** or [F1] 8: Clear Home. The solution of a 2nd-order equation contains two such constants.

Note: To type a prime symbol ('), press [2nd] [′].

```
deSolve(y''+2y'+y=x^2,x,y)[ENTER]
        y=(@1·x+@2)·e^-x+x²−4·x+6

right(ans(1))→temp [ENTER]
          (@1·x+@2)·e^-x+x²−4·x+6

d(temp,x,2)+2*d(temp,x)+temp-x^
2 [ENTER]                         0

DelVar temp [ENTER]            Done
```

Apply **solve()** to an implicit solution if you want to try to convert it to one or more equivalent explicit solutions.

```
deSolve(y'=(cos(y))^2*x,x,y) [ENTER]
```

$$\tan(y)=\frac{x^2}{2}+@3$$

When comparing your results with textbook or manual solutions, be aware that different methods introduce arbitrary constants at different points in the calculation, which may produce different general solutions.

```
solve(ans(1),y) [ENTER]
```

$$y=\tan^{-1}\left(\frac{x^2+2\cdot @3}{2}\right)+@n1\cdot\pi$$

Note: To type an @ symbol, press:

TI-89: [◆] [STO▸]
Voyage 200: [2nd] R

```
ans(1)|@3=c-1 and @n1=0 [ENTER]
```

$$y=\tan^{-1}\left(\frac{x^2+2\cdot(c-1)}{2}\right)$$

deSolve(*1stOrderOde* **and** *initialCondition***,** *independentVar***,** *dependentVar***)** ⇒ *a particular solution*

Returns a particular solution that satisfies *1stOrderOde* and *initialCondition*. This is usually easier than determining a general solution, substituting initial values, solving for the arbitrary constant, and then substituting that value into the general solution.

initialCondition is an equation of the form:

dependentVar (*initialIndependentValue*) = *initialDependentValue*

The *initialIndependentValue* and *initialDependentValue* can be variables such as x0 and y0 that have no stored values. Implicit differentiation can help verify implicit solutions.

```
sin(y)=(y*e^(x)+cos(y))y'→ode [ENTER]
```

$$\sin(y)=(e^{x}\cdot y+\cos(y))\cdot y'$$

```
deSolve(ode and y(0)=0,x,y)→soln [ENTER]
```

$$\frac{^{-}(2\cdot\sin(y)+y^2)}{2}={}^{-}(e^{x}-1)\cdot e^{-x}\cdot\sin(y)$$

```
soln|x=0 and y=0 [ENTER]        true
```

```
d(right(eq)-left(eq),x)/
(d(left(eq)-right(eq),y))
→impdif(eq,x,y) [ENTER]
                                Done
```

```
ode|y'=impdif(soln,x,y) [ENTER]
                                true
```

```
DelVar ode,soln [ENTER]         Done
```

deSolve(*2ndOrderOde* **and** *initialCondition1* **and** *initialCondition2***,** *independentVar***,** *dependentVar***)** ⇒ *a particular solution*

Returns a particular solution that satisfies *2ndOrderOde* and has a specified value of the dependent variable and its first derivative at one point.

For *initialCondition1*, use the form:

dependentVar (*initialIndependentValue*) = *initialDependentValue*

For *initialCondition2*, use the form:

dependentVar' (*initialIndependentValue*) = *initial1stDerivativeValue*

```
deSolve(y''=y^(-1/2) and
y(0)=0 and y'(0)=0,t,y) [ENTER]
```

$$\frac{2\cdot y^{3/4}}{3}=t$$

```
solve(ans(1),y) [ENTER]
```

$$y=\frac{2^{2/3}\cdot(3\cdot t)^{4/3}}{4}\text{ and }t\ge 0$$

deSolve(*2ndOrderOde* **and** *boundaryCondition1* **and** *boundaryCondition2*, *independentVar*, *dependentVar***)** ⇒ *a particular solution*

Returns a particular solution that satisfies *2ndOrderOde* and has specified values at two different points.

```
deSolve(w''-2w'/x+(9+2/x^2)w=
x*e^(x) and w(π/6)=0 and
w(π/3)=0,x,w) [ENTER]
```

$$w=\frac{e^{\frac{\pi}{3}}\cdot x\cdot \cos(3\cdot x)}{10}-\frac{e^{\frac{\pi}{6}}\cdot x\cdot \sin(3\cdot x)}{10}+\frac{x\cdot e^{x}}{10}$$

det() MATH/Matrix menu

det(*squareMatrix*[, *tol*]**)** ⇒ *expression*

Returns the determinant of *squareMatrix*.

Optionally, any matrix element is treated as zero if its absolute value is less than *tol*. This tolerance is used only if the matrix has floating-point entries and does not contain any symbolic variables that have not been assigned a value. Otherwise, *tol* is ignored.

- If you use [◆] [ENTER] or set the mode to Exact/Approx=APPROXIMATE, computations are done using floating-point arithmetic.
- If *tol* is omitted or not used, the default tolerance is calculated as:

  5E⁻14 ∗ **max(dim(***squareMatrix***))** ∗ **rowNorm(***squareMatrix***)**

```
det([a,b;c,d]) [ENTER]          a·d - b·c
det([1,2;3,4]) [ENTER]                 ⁻2
det(identity(3) - x*[1,⁻2,3;
⁻2,4,1;⁻6,⁻2,7]) [ENTER]
          ⁻(98·x^3 - 55·x^2 + 12·x - 1)
```

```
[1E20,1;0,1]→mat1        [1.E20  1]
                         [0      1]
det(mat1) [ENTER]                   0
det(mat1,.1) [ENTER]            1.E20
```

diag() MATH/Matrix menu

diag(*list***)** ⇒ *matrix*
diag(*rowMatrix***)** ⇒ *matrix*
diag(*columnMatrix***)** ⇒ *matrix*

Returns a matrix with the values in the argument list or matrix in its main diagonal.

```
diag({2,4,6}) [ENTER]     [2 0 0]
                          [0 4 0]
                          [0 0 6]
```

diag(*squareMatrix***)** ⇒ *rowMatrix*

Returns a row matrix containing the elements from the main diagonal of *squareMatrix*.

squareMatrix must be square.

```
[4,6,8;1,2,3;5,7,9] [ENTER]   [4 6 8]
                              [1 2 3]
                              [5 7 9]
diag(ans(1)) [ENTER]          [4 2 9]
```

Dialog CATALOG

Dialog
block
EndDlog

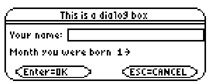
Generates a dialog box when the program is executed.

block can be either a single statement or a series of statements separated with the ":" character. Valid *block* options in the [F3] I/O, 1:Dialog menu item in the Program Editor are 1:Text, 2:Request, 4:DropDown, and 7:Title.

The variables in a dialog box can be given values that will be displayed as the default (or initial) value. If [ENTER] is pressed, the variables are updated from the dialog box and variable ok is set to 1. If [ESC] is pressed, its variables are not updated, and system variable ok is set to zero.

Program listing:

```
:Dlogtest()
:Prgm
:Dialog
:Title    "This is a dialog box"
:Request  "Your name",Str1
:Dropdown "Month you were born",
  seq(string(i),i,1,12),Var1
:EndDlog
:EndPrgm
```



dim() MATH/Matrix/Dimensions menu

dim(*list***)** ⇒ *integer*

Returns the dimension of *list*.

```
dim({0,1,2}) [ENTER]                 3
```

dim(*matrix***)** ⇒ *list*

Returns the dimensions of *matrix* as a two-element list {rows, columns}.

```
dim([1,⁻1,2;⁻2,3,5]) [ENTER]   {2 3}
```

dim(*string***)** ⇒ *integer*

Returns the number of characters contained in character string *string*.

```
dim("Hello") [ENTER]                 5
dim("Hello"&" there") [ENTER]       11
```

Disp CATALOG

Disp [*exprOrString1*] [**,** *exprOrString2*] **...**

Displays the current contents of the Program I/O screen. If one or more *exprOrString* is specified, each expression or character string is displayed on a separate line of the Program I/O screen.

An expression can include conversion operations such as **▸DD** and **▸Rect**. You can also use the ▸ operator to perform unit and number base conversions.

If Pretty Print = ON, expressions are displayed in pretty print.

From the Program I/O screen, you can press [F5] to display the Home screen, or a program can use **DispHome**.

```
Disp "Hello" [ENTER]             Hello
Disp cos(2.3) [ENTER]            ⁻.666…
{1,2,3,4}→L1 [ENTER]
Disp L1 [ENTER]            {1  2  3  4}
Disp 180_min▸_hr [ENTER]       3.•_hr
```

Note: To type an underscore (_), press:
[◆] [_]
[2nd] [_]
To type ▸, press [2nd] [▸].

DispG CATALOG

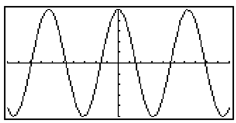
DispG

Displays the current contents of the Graph screen.

In function graphing mode:

Program segment:

```
   ⋮
:5∗cos(x)→y1(x)
:⁻10→xmin
:10→xmax
:⁻5→ymin
:5→ymax
:DispG
   ⋮
```



DispHome CATALOG

DispHome

Displays the current contents of the Home screen.

Program segment:

```
   ⋮
:Disp "The result is: ",xx
:Pause "Press Enter to quit"
:DispHome
:EndPrgm
```

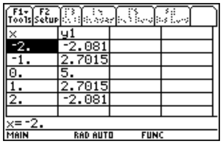
DispTbl CATALOG

DispTbl

Displays the current contents of the Table screen.

Note: The cursor pad is active for scrolling. Press [ESC] or [ENTER] to resume execution if in a program.

```
5*cos(x)→y1(x) [ENTER]
DispTbl [ENTER]
```



▸DMS MATH/Angle menu

expression **▸DMS**
list **▸DMS**
matrix **▸DMS**

Interprets the argument as an angle and displays the equivalent DMS (*DDDDDD°MM′SS.ss″*) number. See °, ', " on page 905 for DMS (degree, minutes, seconds) format.

Note: **▸DMS** will convert from radians to degrees when used in radian mode. If the input is followed by a degree symbol (°), no conversion will occur. You can use **▸DMS** only at the end of an entry line.

In Degree angle mode:

```
45.371 ▸DMS [ENTER]      45° 22'15.6"
{45.371,60} ▸DMS [ENTER]
                 {45° 22'15.6"   60° }
```

dotP() MATH/Matrix/Vector ops menu

dotP(*list1*, *list2***)** ⇒ *expression*

Returns the “dot” product of two lists.

```
dotP({a,b,c},{d,e,f}) [ENTER]
                      a·d+b·e+c·f
dotP({1,2},{5,6}) [ENTER]        17
```

---

dotP(*vector1*, *vector2***)** ⇒ *expression*

Returns the “dot” product of two vectors.

Both must be row vectors, or both must be column vectors.

```
dotP([a,b,c],[d,e,f]) [ENTER]
                      a·d+b·e+c·f
dotP([1,2,3],[4,5,6]) [ENTER]    32
```

DrawFunc CATALOG

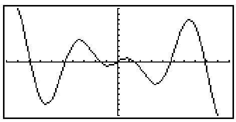
DrawFunc *expression*

Draws *expression* as a function, using x as the independent variable.

Note: Regraphing erases all drawn items.

In function graphing mode and ZoomStd window:

```
DrawFunc 1.25x*cos(x) [ENTER]
```



DrawInv CATALOG

DrawInv *expression*

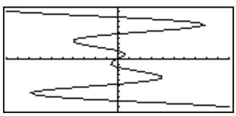
Draws the inverse of *expression* by plotting x values on the y axis and y values on the x axis.

x is the independent variable.

Note: Regraphing erases all drawn items.

In function graphing mode and ZoomStd window:

```
DrawInv 1.25x*cos(x) [ENTER]
```



DrawParm CATALOG

DrawParm *expression1*, *expression2*
[, *tmin*] [, *tmax*] [, *tstep*]

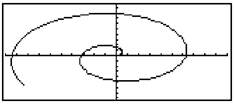
Draws the parametric equations *expression1* and *expression2*, using t as the independent variable.

Defaults for *tmin*, *tmax*, and *tstep* are the current settings for the Window variables tmin, tmax, and tstep. Specifying values does not alter the window settings. If the current graphing mode is not parametric, these three arguments are required.

Note: Regraphing erases all drawn items.

In function graphing mode and ZoomStd window:

```
DrawParm
t*cos(t),t*sin(t),0,10,.1 [ENTER]
```



DrawPol CATALOG

DrawPol *expression*[, θ*min*] [, θ*max*] [, θ*step*]

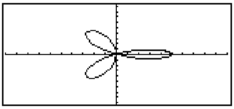
Draws the polar graph of *expression*, using θ as the independent variable.

Defaults for θ*min*, θ*max*, and θ*step* are the current settings for the Window variables θmin, θmax, and θstep. Specifying values does not alter the window settings. If the current graphing mode is not polar, these three arguments are required.

Note: Regraphing erases all drawn items.

In function graphing mode and ZoomStd window:

```
DrawPol 5*cos(3*θ),0,3.5,.1
[ENTER]
```



DrawSlp CATALOG

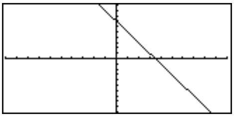
DrawSlp *x1*, *y1*, *slope*

Displays the graph and draws a line using the formula y−y1=slope·(x−x1).

Note: Regraphing erases all drawn items.

In function graphing mode and ZoomStd window:

```
DrawSlp 2,3,⁻2 [ENTER]
```



DropDown CATALOG

DropDown *titleString*, {*item1String*, *item2String*, ...}, *varName*

Displays a drop-down menu with the name *titleString* and containing the items **1:***item1String*, **2:***item2String*, and so forth. **DropDown** must be within a **Dialog...EndDlog** block.

If *varName* already exists and has a value within the range of items, the referenced item is displayed as the default selection. Otherwise, the menu's first item is the default selection.

When you select an item from the menu, the corresponding number of the item is stored in the variable *varName*. (If necessary, **DropDown** creates *varName*.)

See **Dialog** program listing example.

DrwCtour CATALOG

DrwCtour *expression*
DrwCtour *list*

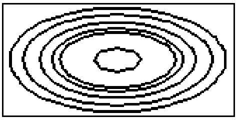
Draws contours on the current 3D graph at the z values specified by *expression* or *list*. The 3D graphing mode must already be set. **DrwCtour** automatically sets the graph format style to CONTOUR LEVELS.

By default, the graph automatically contains the number of equally spaced contours specified by the ncontour Window variable. **DrwCtour** draws contours in addition to the defaults.

To turn off the default contours, set ncontour to zero, either by using the Window screen or by storing 0 to the ncontour system variable.

In 3D graphing mode:

```
(1/5)x^2+(1/5)y^2-10→z1(x,y)
[ENTER]                                    Done
-10→xmin:10→xmax [ENTER]                     10
-10→ymin:10→ymax [ENTER]                     10
-10→zmin:10→zmax [ENTER]                     10
0→ncontour [ENTER]                            0
DrwCtour {-9,-4.5,-3,0,4.5,9}
[ENTER]
```



- Use the cursor to change the viewing angle. Press 0 (zero) to return to the original view.

  To toggle between different graph format styles, press:
  TI-89: [I]
  Voyage 200: F

- Press X, Y, or Z to look down the corresponding axis.

E — TI-89: [EE] **key** — Voyage 200: [2nd] [EE] **key**

*mantissa***E***exponent*

Enters a number in scientific notation. The number is interpreted as *mantissa* × 10^{exponent}.

Hint: If you want to enter a power of 10 without causing a decimal value result, use 10^*integer*.

```
2.3E4 [ENTER]                          23000.
2.3E9+4.1E15 [ENTER]                   4.1E15
3*10^4 [ENTER]                          30000
```

e^() — TI-89: [◆] [eˣ] **key** — Voyage 200: [2nd] [eˣ] **key**

e*^(expression1***)** ⇒ *expression*

Returns *e* raised to the *expression1* power.

Note: On the TI-89 Titanium, pressing [◆] [eˣ] to display e^(is different from pressing [alpha] [E]. On the Voyage 200, pressing [2nd][eˣ] to display e^ is different from accessing the character e from the QWERTY keyboard.

You can enter a complex number in r$e^{i\theta}$ polar form. However, use this form in Radian angle mode only; it causes a Domain error in Degree angle mode.

```
e^(1) [ENTER]                               e
e^(1.) [ENTER]                          2.718...
e^(3)^2 [ENTER]                            e^9
```

---

e*^(list1***)** ⇒ *list*

Returns *e* raised to the power of each element in *list1*.

```
e^({1,1.,0,.5}) [ENTER]
                  {e   2.718...   1   1.648...}
```

e^(*squareMatrix1***)** ⇒ *squareMatrix*

Returns the matrix exponential of *squareMatrix1*. This is *not* the same as calculating *e* raised to the power of each element. For information about the calculation method, refer to **cos()**.

squareMatrix1 must be diagonalizable. The result always contains floating-point numbers.

```
e^([1,5,3;4,2,1;6,⁻2,1]) [ENTER]

⎡782.209  559.617  456.509⎤
⎢680.546  488.795  396.521⎥
⎣524.929  371.222  307.879⎦
```

eigVc() MATH/Matrix menu

eigVc(*squareMatrix***)** ⇒ *matrix*

Returns a matrix containing the eigenvectors for a real or complex *squareMatrix*, where each column in the result corresponds to an eigenvalue. Note that an eigenvector is not unique; it may be scaled by any constant factor. The eigenvectors are normalized, meaning that if $V = [x_1, x_2, \ldots, x_n]$, then:

$$\sqrt{x_1{}^2 + x_2{}^2 + \ldots + x_n{}^2} = 1$$

squareMatrix is first balanced with similarity transformations until the row and column norms are as close to the same value as possible. The *squareMatrix* is then reduced to upper Hessenberg form and the eigenvectors are computed via a Schur factorization.

In Rectangular complex format mode:

```
[-1,2,5;3,-6,9;2,-5,7]→m1 [ENTER]
                  ⎡⁻1   2   5⎤
                  ⎢3   ⁻6   9⎥
                  ⎣2   ⁻5   7⎦
eigVc(m1) [ENTER]
⎡⁻.800…  .767…            .767…           ⎤
⎢.484…   .573…+.052…·i    .573…-.052…·i   ⎥
⎣.352…   .262…+.096…·i    .262…-.096…·i   ⎦
```

eigVl() MATH/Matrix menu

eigVl(*squareMatrix***)** ⇒ *list*

Returns a list of the eigenvalues of a real or complex *squareMatrix*.

squareMatrix is first balanced with similarity transformations until the row and column norms are as close to the same value as possible. The *squareMatrix* is then reduced to upper Hessenberg form and the eigenvalues are computed from the upper Hessenberg matrix.

In Rectangular complex format mode:

```
[-1,2,5;3,-6,9;2,-5,7]→m1 [ENTER]
                  ⎡⁻1   2   5⎤
                  ⎢3   ⁻6   9⎥
                  ⎣2   ⁻5   7⎦
eigVl(m1) [ENTER]
        {⁻4.409…  2.204…+.763…·i
                  2.204…-.763…·i}
```

Else See **If**, page 826.

ElseIf CATALOG See also **If**, page 826.

If *Boolean expression1* **Then**
  block1
ElseIf *Boolean expression2* **Then**
  block2
  ⋮
ElseIf *Boolean expressionN* **Then**
  blockN
EndIf
  ⋮

ElseIf can be used as a program instruction for program branching.

Program segment:

```
   ⋮
:If choice=1 Then
:  Goto option1
:  ElseIf choice=2 Then
:  Goto option2
:  ElseIf choice=3 Then
:  Goto option3
:  ElseIf choice=4 Then
:  Disp "Exiting Program"
:  Return
:EndIf
   ⋮
```

EndCustm See **Custom**, page 800.

EndDlog

See **Dialog**, page 807.

EndFor

See **For**, page 819.

EndFunc

See **Func**, page 820.

EndIf

See **If**, page 826.

EndLoop

See **Loop**, page 836.

EndPrgm

See **Prgm**, page 850.

EndTBar

See **ToolBar**, page 885.

EndTry

See **Try**, page 886.

EndWhile

See **While**, page 888.

entry() **CATALOG**

entry() ⇒ *expression*
entry(*integer***)** ⇒ *expression*

Returns a previous entry-line entry from the Home screen history area.

integer, if included, specifies which entry expression in the history area. The default is 1, the most recently evaluated entry. Valid range is from 1 to 99 and cannot be an expression.

Note: If the last entry is still highlighted on the Home screen, pressing [ENTER] is equivalent to executing **entry(1)**.

On the Home screen:

`1+1/x` [ENTER]	$\frac{1}{x}+1$
`1+1/entry(1)` [ENTER]	$2-\frac{1}{x+1}$
[ENTER]	$\frac{1}{2\cdot(2\cdot x+1)}+3/2$
[ENTER]	$5/3-\frac{1}{3\cdot(3\cdot x+2)}$
`entry(4)` [ENTER]	$\frac{1}{x}+1$

exact() **MATH/Number menu**

exact(*expression1* [**,** *tol*]**)** ⇒ *expression*
exact(*list1* [**,** *tol*]**)** ⇒ *list*
exact(*matrix1* [**,** *tol*]**)** ⇒ *matrix*

Uses Exact mode arithmetic regardless of the Exact/Approx mode setting to return, when possible, the rational-number equivalent of the argument.

tol specifies the tolerance for the conversion; the default is 0 (zero).

`exact(.25)` [ENTER]	1/4
`exact(.333333)` [ENTER]	$\frac{333333}{1000000}$
`exact(.33333,.001)`	1/3
`exact(3.5x+y)` [ENTER]	$\frac{7\cdot x}{2}+y$
`exact({.2,.33,4.125})` [ENTER]	$\{1/5\ \frac{33}{100}\ 33/8\}$

Exec CATALOG

Exec *string* [**,** *expression1*] [**,** *expression2*] **...**

Executes a *string* consisting of a series of Motorola 68000 op-codes. These codes act as a form of an assembly-language program. If needed, the optional *expressions* let you pass one or more arguments to the program.

For more information, check the TI Web site: **http://www.ti.com/calc**

Warning: Exec gives you access to the full power of the microprocessor. Please be aware that you can easily make a mistake that locks up the calculator and causes you to lose your data. We suggest you make a backup of the calculator contents before attempting to use the **Exec** command.

Exit CATALOG

Exit

Exits the current **For**, **While**, or **Loop** block.

Exit is not allowed outside the three looping structures (**For**, **While**, or **Loop**).

Program listing:

```
:0→temp
:For i,1,100,1
:  temp+i→temp
:  If temp>20
:  Exit
:EndFor
:Disp temp
```

Contents of **temp** after execution: 21

exp▸list() CATALOG

exp▸list(*expression*,*var***)** ⇒ *list*

Examines *expression* for equations that are separated by the word "or," and returns a list containing the right-hand sides of the equations of the form *var=expression*. This gives you an easy way to extract some solution values embedded in the results of the **solve()**, **cSolve()**, **fMin()**, and **fMax()** functions.

Note: exp▸list() is not necessary with the **zeros** and **cZeros()** functions because they return a list of solution values directly.

```
solve(x^2-x-2=0,x) [ENTER]   x=2 or x=⁻1

exp▸list(solve(x^2-x-2=0,x),x) [ENTER]
                                   {⁻1  2}
```

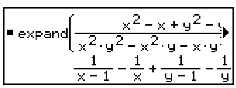
expand() MATH/Algebra menu

expand(*expression1* [**,** *var*]**)** ⇒ *expression*
expand(*list1* [**,***var*]**)** ⇒ *list*
expand(*matrix1* [**,***var*]**)** ⇒ *matrix*

expand(*expression1***)** returns *expression1* expanded with respect to all its variables. The expansion is polynomial expansion for polynomials and partial fraction expansion for rational expressions.

The goal of **expand()** is to transform *expression1* into a sum and/or difference of simple terms. In contrast, the goal of **factor()** is to transform *expression1* into a product and/or quotient of simple factors.

```
expand((x+y+1)^2) [ENTER]
     x²+2·x·y+2·x+y²+2·y+1

expand((x^2-x+y^2-y)/(x^2*y^2
  -x^2*y-x*y^2+x*y)) [ENTER]
```

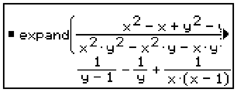


expand(*expression1,var***)** returns *expression* expanded with respect to *var*. Similar powers of *var* are collected. The terms and their factors are sorted with *var* as the main variable. There might be some incidental factoring or expansion of the collected coefficients. Compared to omitting *var*, this often saves time, memory, and screen space, while making the expression more comprehensible.

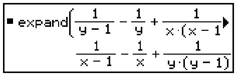
```
expand((x+y+1)^2,y) [ENTER]
```
$y^2 + 2 \cdot y \cdot (x+1) + (x+1)^2$

```
expand((x+y+1)^2,x) [ENTER]
```
$x^2 + 2 \cdot x \cdot (y+1) + (y+1)^2$

```
expand((x^2-x+y^2-y)/(x^2*y^2
  -x^2*y-x*y^2+x*y),y) [ENTER]
```



```
expand(ans(1),x) [ENTER]
```



Even when there is only one variable, using *var* might make the denominator factorization used for partial fraction expansion more complete.

Hint: For rational expressions, **propFrac()** is a faster but less extreme alternative to **expand()**.

Note: See also **comDenom()** for an expanded numerator over an expanded denominator.

```
expand((x^3+x^2-2)/(x^2-2))
[ENTER]
```
$\frac{2 \cdot x}{x^2-2} + x+1$

```
expand(ans(1),x) [ENTER]
```
$\frac{1}{x-\sqrt{2}} + \frac{1}{x+\sqrt{2}} + x+1$

expand(*expression1*,[*var*]**)** also distributes logarithms and fractional powers regardless of *var*. For increased distribution of logarithms and fractional powers, inequality constraints might be necessary to guarantee that some factors are nonnegative.

expand(*expression1*, [*var*]**)** also distributes absolute values, **sign()**, and exponentials, regardless of *var*.

Note: See also **tExpand()** for trigonometric angle-sum and multiple-angle expansion.

```
ln(2x*y)+√(2x*y) [ENTER]
```
$\ln(2 \cdot x \cdot y) + \sqrt{(2 \cdot x \cdot y)}$

```
expand(ans(1)) [ENTER]
```
$\ln(x \cdot y) + \sqrt{2} \cdot \sqrt{(x \cdot y)} + \ln(2)$

```
expand(ans(1))|y>=0 [ENTER]
```
$\ln(x) + \sqrt{2} \cdot \sqrt{x} \cdot \sqrt{y} + \ln(y) + \ln(2)$

```
sign(x*y)+abs(x*y)+e^(2x+y)
[ENTER]
```
$e^{2 \cdot x+y} + \text{sign}(x \cdot y) + |x \cdot y|$

```
expand(ans(1)) [ENTER]
```
$\text{sign}(x) \cdot \text{sign}(y) + |x| \cdot |y| + (e^x)^2 \cdot e^y$

expr() MATH/String menu

expr(*string***)** ⇒ *expression*

Returns the character string contained in *string* as an expression and immediately executes it.

```
expr("1+2+x^2+x") [ENTER]      x² + x + 3

expr("expand((1+x)^2)") [ENTER]
                        x² + 2 • x + 1

"Define cube(x)=x^3"→funcstr
[ENTER]
                 "Define cube(x)=x^3"

expr(funcstr) [ENTER]             Done

cube(2) [ENTER]                      8
```

ExpReg   **MATH/Statistics/Regressions menu**

ExpReg *list1*, *list2* [, [*list3*] [, *list4*, *list5*]]

Calculates the exponential regression and updates all the system statistics variables.

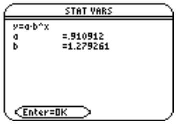
All the lists must have equal dimensions except for *list5*.

list1 represents xlist.
list2 represents ylist.
list3 represents frequency.
list4 represents category codes.
list5 represents category include list.

Note: *list1* through *list4* must be a variable name or c1–c99 (columns in the last data variable shown in the Data/Matrix Editor). *list5* does not have to be a variable name and cannot be c1–c99 .

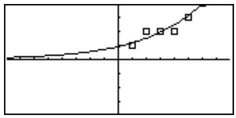
In function graphing mode:

```
{1,2,3,4,5,6,7,8}→L1 [ENTER]
                              {1 2 ...}
{1,2,2,2,3,4,5,7}→L2 [ENTER]
                              {1 2 ...}
ExpReg L1,L2 [ENTER]               Done
ShowStat [ENTER]
```




```
[ENTER]
Regeq(x)→y1(x) [ENTER]             Done
NewPlot 1,1,L1,L2 [ENTER]          Done
```

[◆] [GRAPH]



factor()   **MATH/Algebra menu**

factor(*expression1*[**,** *var*]**)** ⇒ *expression*
factor(*list1*[,*var*]**)** ⇒ *list*
factor(*matrix1*[,*var*]**)** ⇒ *matrix*

factor(*expression1*) returns *expression1* factored with respect to all of its variables over a common denominator.

expression1 is factored as much as possible toward linear rational factors without introducing new non-real subexpressions. This alternative is appropriate if you want factorization with respect to more than one variable.

```
factor(a^3*x^2-a*x^2-a^3+a)
[ENTER]
    a·(a-1)·(a+1)·(x-1)·(x+1)
factor(x^2+1) [ENTER]              x²+1
factor(x^2-4) [ENTER]   (x-2)·(x+2)
factor(x^2-3) [ENTER]              x²-3
factor(x^2-a) [ENTER]              x²-a
```

---

factor(*expression1,var***)** returns *expression1* factored with respect to variable *var*.

expression1 is factored as much as possible toward real factors that are linear in *var*, even if it introduces irrational constants or subexpressions that are irrational in other variables.

The factors and their terms are sorted with *var* as the main variable. Similar powers of *var* are collected in each factor. Include *var* if factorization is needed with respect to only that variable and you are willing to accept irrational expressions in any other variables to increase factorization with respect to *var*. There might be some incidental factoring with respect to other variables.

```
factor(a^3*x^2-a*x^2-a^3+a,x)
[ENTER]
          a·(a²-1)·(x-1)·(x+1)
factor(x^2-3,x) [ENTER]
                  (x+√3)·(x-√3)
factor(x^2-a,x) [ENTER]
                  (x+√a)·(x-√a)
```

For the AUTO setting of the Exact/Approx mode, including *var* permits approximation with floating-point coefficients where irrational coefficients cannot be explicitly expressed concisely in terms of the built-in functions. Even when there is only one variable, including *var* might yield more complete factorization.

Note: See also **comDenom()** for a fast way to achieve partial factoring when **factor()** is not fast enough or if it exhausts memory.

Note: See also **cFactor()** for factoring all the way to complex coefficients in pursuit of linear factors.

```
factor(x^5+4x^4+5x^3-6x-3)
[ENTER]
                    x⁵ + 4·x⁴ + 5·x³- 6·x - 3
factor(ans(1),x) [ENTER]
               (x-.964…)·(x +.611…)·
          (x + 2.125…)·(x² + 2.227…·
                              x + 2.392…)
```

---

factor(*rationalNumber***)** returns the rational number factored into primes. For composite numbers, the computing time grows exponentially with the number of digits in the second-largest factor. For example, factoring a 30-digit integer could take more than a day, and factoring a 100-digit number could take more than a century.

Note: To stop (break) a computation, press [ON].

If you merely want to determine if a number is prime, use **isPrime()** instead. It is much faster, particularly if *rationalNumber* is not prime and if the second-largest factor has more than five digits.

```
factor(152417172689) [ENTER]
                         123457·1234577
isPrime(152417172689) [ENTER]false
```

Fill — MATH/Matrix menu

Fill *expression, matrixVar* ⇒ *matrix*

Replaces each element in variable *matrixVar* with *expression*.

matrixVar must already exist.

```
[1,2;3,4]→amatrx [ENTER]       [1  2]
                               [3  4]
Fill 1.01,amatrx [ENTER]         Done
amatrx [ENTER]             [1.01 1.01]
                           [1.01 1.01]
```

Fill *expression, listVar* ⇒ *list*

Replaces each element in variable *listVar* with *expression*.

listVar must already exist.

```
{1,2,3,4,5}→alist [ENTER]
                          {1 2 3 4 5}
Fill 1.01,alist [ENTER]          Done
alist [ENTER]
         {1.01 1.01 1.01 1.01 1.01}
```

floor() — MATH/Number menu

floor(*expression***)** ⇒ *integer*

Returns the greatest integer that is ≤ the argument. This function is identical to **int()**.

The argument can be a real or a complex number.

```
floor(⁻2.14) [ENTER]              ⁻3.
```

---

floor(*list1***)** ⇒ *list*
floor(*matrix1***)** ⇒ *matrix*

Returns a list or matrix of the floor of each element.

Note: See also **ceiling()** and **int()**.

```
floor({3/2,0,⁻5.3}) [ENTER]
                           {1 0 ⁻6.}
floor([1.2,3.4;2.5,4.8]) [ENTER]
                             [1. 3.]
                             [2. 4.]
```

fMax() MATH/Calculus menu

fMax(*expression, var***)** ⇒ *Boolean expression*

Returns a Boolean expression specifying candidate values of *var* that maximize *expression* or locate its least upper bound.

Use the "|" operator to restrict the solution interval and/or specify the sign of other undefined variables.

For the APPROX setting of the Exact/Approx mode, **fMax()** iteratively searches for one approximate local maximum. This is often faster, particularly if you use the "|" operator to constrain the search to a relatively small interval that contains exactly one local maximum.

Note: See also **fMin()** and **max()**.

```
fMax(1-(x-a)^2-(x-b)^2,x) [ENTER]
                                  x = (a+b)/2
fMax(.5x^3-x-2,x) [ENTER]        x = ∞
fMax(.5x^3-x-2,x)|x≤1 [ENTER]
                                  x = ⁻.816...
fMax(a*x^2,x) [ENTER]
  x = ∞ or x = ⁻∞ or x = 0 or a = 0
fMax(a*x^2,x)|a<0 [ENTER]        x = 0
```

fMin() MATH/Calculus menu

fMin(*expression, var***)** ⇒ *Boolean expression*

Returns a Boolean expression specifying candidate values of *var* that minimize *expression* or locate its greatest lower bound.

Use the "|" operator to restrict the solution interval and/or specify the sign of other undefined variables.

For the APPROX setting of the Exact/Approx mode, **fMin()** iteratively searches for one approximate local minimum. This is often faster, particularly if you use the "|" operator to constrain the search to a relatively small interval that contains exactly one local minimum.

Note: See also **fMax()** and **min()**.

```
fMin(1-(x-a)^2-(x-b)^2,x)
[ENTER]
                         x = ∞ or x = ⁻∞
fMin(.5x^3-x-2,x)|x≥1 [ENTER]  x = 1
fMin(a*x^2,x) [ENTER]
  x = ∞ or x = ⁻∞ or x = 0 or a = 0
fMin(a*x^2,x)|a>0 and x>1 [ENTER]
                                  x = 1.
fMin(a*x^2,x)|a>0 [ENTER]        x = 0
```

FnOff CATALOG

FnOff

Deselects all Y= functions for the current graphing mode.

In split-screen, two-graph mode, **FnOff** only applies to the active graph.

FnOff [1] [, 2] ... [,99]

Deselects the specified Y= functions for the current graphing mode.

In function graphing mode:
FnOff 1,3 [ENTER] deselects y1(x) and y3(x).

In parametric graphing mode:
FnOff 1,3 [ENTER] deselects xt1(t), yt1(t), xt3(t), and yt3(t).

FnOn CATALOG

FnOn

Selects all Y= functions that are defined for the current graphing mode.

In split-screen, two-graph mode, **FnOn** only applies to the active graph.

FnOn [1] [, 2] ... [,99]

Selects the specified Y= functions for the current graphing mode.

Note: In 3D graphing mode, only one function at a time can be selected. FnOn 2 selects z2(x,y) and deselects any previously selected function. In the other graph modes, previously selected functions are not affected.

For CATALOG

For *var*, *low*, *high* [, *step*]
block
EndFor

Executes the statements in *block* iteratively for each value of *var*, from *low* to *high*, in increments of *step*.

var must not be a system variable.

step can be positive or negative. The default value is 1.

block can be either a single statement or a series of statements separated with the ":" character.

Program segment:

```
  :
:0→tempsum : 1→step
:For i,1,100,step
:  tempsum+i→tempsum
:EndFor
:Disp tempsum
  :
```

Contents of tempsum after execution: 5050

Contents of tempsum when step is changed to 2: 2500

format() MATH/String menu

format(*expression*[, *formatString*]**)** ⇒ *string*

Returns *expression* as a character string based on the format template.

expression must simplify to a number. *formatString* is a string and must be in the form: "F[*n*]", "S[*n*]", "E[*n*]", "G[*n*][*c*]", where [ ] indicate optional portions.

F[*n*]: Fixed format. *n* is the number of digits to display after the decimal point.

S[*n*]: Scientific format. *n* is the number of digits to display after the decimal point.

E[*n*]: Engineering format. *n* is the number of digits after the first significant digit. The exponent is adjusted to a multiple of three, and the decimal point is moved to the right by zero, one, or two digits.

G[*n*][*c*]: Same as fixed format but also separates digits to the left of the radix into groups of three. *c* specifies the group separator character and defaults to a comma. If *c* is a period, the radix will be shown as a comma.

[R*c*]: Any of the above specifiers may be suffixed with the R*c* radix flag, where *c* is a single character that specifies what to substitute for the radix point.

```
format(1.234567,"f3") [ENTER]
                        "1.235"
format(1.234567,"s2") [ENTER]
                       "1.23E 0"
format(1.234567,"e3") [ENTER]
                      "1.235E 0"
format(1.234567,"g3") [ENTER]
                        "1.235"
format(1234.567,"g3") [ENTER]
                    "1,234.567"
format(1.234567,"g3,r:") [ENTER]
                        "1:235"
```

fPart() MATH/Number menu

fPart(*expression1***)** ⇒ *expression*
fPart(*list1***)** ⇒ *list*
fPart(*matrix1***)** ⇒ *matrix*

Returns the fractional part of the argument.

For a list or matrix, returns the fractional parts of the elements.

The argument can be a real or a complex number.

```
fPart(⁻1.234) [ENTER]                ⁻.234

fPart({1, ⁻2.3, 7.003}) [ENTER]
                          {0  ⁻.3  .003}
```

Func CATALOG

Func
block
EndFunc

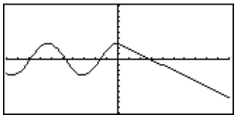
Required as the first statement in a multi-statement function definition.

block can be either a single statement or a series of statements separated with the ":" character.

Note: when() also can be used to define and graph piecewise-defined functions.

In function graphing mode, define a piecewise function:

```
Define g(x)=Func:If x<0 Then
  :Return 3*cos(x):Else:Return
  3-x:EndIf:EndFunc [ENTER]    Done

Graph g(x) [ENTER]
```



gcd() MATH/Number menu

gcd(*number1, number2***)** ⇒ *expression*

Returns the greatest common divisor of the two arguments. The **gcd** of two fractions is the **gcd** of their numerators divided by the **lcm** of their denominators.

In Auto or Approximate mode, the **gcd** of fractional floating-point numbers is 1.0.

```
gcd(18,33) [ENTER]                    3
```

---

gcd(*list1, list2***)** ⇒ *list*

Returns the greatest common divisors of the corresponding elements in *list1* and *list2*.

```
gcd({12,14,16},{9,7,5}) [ENTER]
                             {3 7 1}
```

---

gcd(*matrix1, matrix2***)** ⇒ *matrix*

Returns the greatest common divisors of the corresponding elements in *matrix1* and *matrix2*.

```
gcd([2,4;6,8],[4,8;12,16]) [ENTER]
                             [2  4]
                             [6  8]
```

Get CATALOG

Get *var*

Retrieves a CBL 2™/CBL™ (Calculator-Based Laboratory™) or CBR™ (Calculator-Based Ranger™) value from the link port and stores it in variable *var*.

Program segment:

```
   ⋮
:Send {3,1,⁻1,0}
:For i,1,99
:  Get data[i]
:  PtOn i,data[i]
:EndFor
   ⋮
```

GetCalc CATALOG

GetCalc *var*

Retrieves a value from the link port and stores it in variable *var*. This is for unit-to-unit linking.

Note: To get a variable to the link port from another unit, use [2nd] [VAR-LINK] on the other unit to select and send a variable, or do a **SendCalc** on the other unit.

Program segment:

```
  :
:Disp "Press Enter when ready"
:Pause
:GetCalc L1
:Disp "List L1 received"
  :
```

GetCalc *var[,port]*

Retrieves a value from the link port and stores it in variable *var* on the receiving TI-89 Titanium.

If the port is not specified, or *port = 0* is specified, the TI-89 Titanium waits for data from either port.

If *port = 1*, the TI-89 Titanium waits for data from the USB port.

If *port = 2*, the TI-89 Titanium waits for data from the I/O port.

getConfg() CATALOG

getConfg() ⇒ *ListPairs*

Returns a list of calculator attributes. The attribute name is listed first, followed by its value.

:

```
getConfg() [ENTER]
      {"Product Name" "Advanced
           Mathematics Software"
   "Version" "2.00, 09/25/1999"
         "Product ID" "03-1-4-68"
       "ID #" "01012 34567 ABCD"
                  "Cert. Rev. #" 0
              "Screen Width" 160
             "Screen Height" 100
              "Window Width" 160
              "Window Height" 67
               "RAM Size" 262132
               "Free RAM" 197178
           "Archive Size" 655360
          "Free Archive" 655340}
```

```
getConfg() [ENTER]
      {"Product Name" "Advanced
           Mathematics Software"
   "Version" "2.00, 09/25/1999"
         "Product ID" "01-1-4-80"
       "ID #" "01012 34567 ABCD"
                  "Cert. Rev. #" 0
              "Screen Width" 240
             "Screen Height" 120
              "Window Width" 240
              "Window Height" 91
               "RAM Size" 262144
               "Free RAM" 192988
           "Archive Size" 720896
          "Free Archive" 720874}
```

Note: Your screen may display different attribute values. The Cert. Rev. # attribute appears only if you have purchased and installed additional software into the calculator.

getDate() CATALOG

getDate() ⇒ *list*

Returns a list giving the date according to the current value of the clock. The list is in {*year,month,day*} format.

```
getDate() [ENTER]        {2002  2  22}
```

getDenom() MATH/Algebra/Extract menu

getDenom(*expression1***)** ⇒ *expression*

Transforms *expression1* into one having a reduced common denominator, and then returns its denominator.

```
getDenom((x+2)/(y-3)) [ENTER]    y - 3
getDenom(2/7) [ENTER]            7
getDenom(1/x+(y^2+y)/y^2) [ENTER]
                                 x·y
```

getDtFmt() CATALOG

getDtFmt() ⇒ *integer*

Returns an integer representing the date format that is currently set on the device.

Integer values:

```
1 = MM/DD/YY
2 = DD/MM/YY
3 = MM.DD.YY
4 = DD.MM.YY
5 = YY.MM.DD
6 = MM-DD-YY
7 = DD-MM-YY
8 = YY-MM-DD
```

getDtStr() CATALOG

getDtStr([*integer*]**)** ⇒ *string*

Returns a string of the current date in the current date format. For example, a returned string of *28/09/02* represents the 28th day of September, 2002 (when the date format is set to DD/MM/YY).

If you enter the optional integer that corresponds to a date format, the string returns the current date in the specified format.

Optional integer values:

```
1 = MM/DD/YY
2 = DD/MM/YY
3 = MM.DD.YY
4 = DD.MM.YY
5 = YY.MM.DD
6 = MM-DD-YY
7 = DD-MM-YY
8 = YY-MM-DD
```

getFold() CATALOG

getFold() ⇒ *nameString*

Returns the name of the current folder as a string.

```
getFold() [ENTER]               "main"
getFold()→oldfoldr [ENTER]      "main"
oldfoldr [ENTER]                "main"
```

getKey() CATALOG

getKey() ⇒ *integer*

Returns the key code of the key pressed. Returns 0 if no key is pressed.

The prefix keys (shift [↑], second function [2nd], option [◆], alpha [alpha], and drag [drag]) are not recognized by themselves; however, they modify the keycodes of the key that follows them. For example: [◆] [X] ≠ [X] ≠ [2nd] [X].

For a listing of key codes, see Appendix B.

Program listing:

```
:Disp
:Loop
:  getKey()→key
:  while key=0
:    getKey()→key
:  EndWhile
:  Disp key
:  If key = ord("a")
:  Stop
:EndLoop
```

getMode() CATALOG

getMode(*modeNameString***)** ⇒ *string*
getMode("ALL") ⇒ *ListStringPairs*

If the argument is a specific mode name, returns a string containing the current setting for that mode.

If the argument is **"ALL"**, returns a list of string pairs containing the settings of all the modes. If you want to restore the mode settings later, you must store the **getMode("ALL")** result in a variable, and then use **setMode()** to restore the modes.

For a listing of mode names and possible settings, see **setMode()**.

Note: To set or return information about the Unit System mode, use **setUnits()** or **getUnits()** instead of **setMode()** or **getMode()**.

```
getMode("angle") [ENTER]   "RADIAN"

getMode("graph") [ENTER] "FUNCTION"

getMode("all") [ENTER]
              {"Graph" "FUNCTION"
        "Display Digits" "FLOAT 6"
                 "Angle" "RADIAN"
    "Exponential Format" "NORMAL"
        "Complex Format" "REAL"
    "Vector Format" "RECTANGULAR"
             "Pretty Print" "ON"
           "Split Screen" "FULL"
            "Split 1 App" "Home"
           "Split 2 App" "Graph"
         "Number of Graphs" "1"
            "Graph 2" "FUNCTION"
      "Split Screen Ratio"  "1,1"
          "Exact/Approx" "AUTO"
                 "Base" "DEC"}
```

Note: Your screen may display different mode settings.

getNum() MATH/Algebra/Extract menu

getNum(*expression1***)** ⇒ *expression*

Transforms *expression1* into one having a reduced common denominator, and then returns its numerator.

```
getNum((x+2)/(y-3)) [ENTER]     x + 2

getNum(2/7) [ENTER]                 2

getNum(1/x+1/y) [ENTER]         x + y
```

getTime() CATALOG

getTime() ⇒ *list*

Returns a list giving the time according to the current value of the clock. The list is in {*hour,minute,second*} format. The time is returned in the 24 hour format.

getTmFmt() CATALOG

getTmFmt() ⇒ *integer*

Returns an integer representing the clock time format that is currently set on the device.

Integer values:

12 = 12 hour clock

24 = 24 hour clock

getTmStr() CATALOG

getTmStr([*integer*]**)** ⇒ *string*

Returns a string of the current clock time in the current time format.

If you enter the optional integer that corresponds to a clock time format, the string returns the current time in the specified format.

Optional integer values:

```
12 = 12 hour clock
24 = 24 hour clock
```

getTmZn() CATALOG

getTmZn() ⇒ *integer*

Returns an integer representing the time zone that is currently set on the device.

The returned integer represents the number of minutes the time zone is offset from Greenwich Mean Time (GMT), as established in Greenwich, England. For example, if the time zone is offset from GMT by two hours, the device would return 120 (minutes).

Integers for time zones west of GMT are negative.

Integers for time zones east of GMT are positive.

If Greenwich Mean Time is 14:07:07, it is:

8:07:07 a.m. in Denver, Colorado (Mountain Daylight Time)
(–360 minutes from GMT)

16:07:07 p.m. in Brussels, Belgium (Central European Standard Time)
(+120 minutes from GMT)

getType() CATALOG

getType(*var***)** ⇒ *string*

Returns a string indicating the data type of variable *var*.

If *var* has not been defined, returns the string "NONE".

```
{1,2,3}→temp [ENTER]          {1 2 3}
getType(temp) [ENTER]          "LIST"

2+3i→temp [ENTER]               2 + 3i
getType(temp) [ENTER]          "EXPR"

DelVar temp [ENTER]               Done
getType(temp) [ENTER]          "NONE"
```

Data Type	Variable Contents
"ASM"	Assembly-language program
"DATA"	Data type
"EXPR"	Expression (includes complex/arbitrary/undefined, ∞, ⁻∞, TRUE, FALSE, pi, *e*)
"FUNC"	Function
"GDB"	Graph data base
"LIST"	List
"MAT"	Matrix
"NONE"	Variable does not exist
"NUM"	Real number
"OTHER"	Miscellaneous data type for future use by software applications
"PIC"	Picture
"PRGM"	Program
"STR"	String
"TEXT"	Text type
"VAR"	Name of another variable

getUnits() CATALOG

getUnits() ⇒ *list*

Returns a list of strings that contain the current default units for all categories except constants, temperature, amount of substance, luminous intensity, and acceleration. *list* has the form:

{"*system*" "*cat1*" "*unit1*" "*cat2*" "*unit2*" …}

The first string gives the system (SI, ENG/US, or CUSTOM). Subsequent pairs of strings give a category (such as Length) and its default unit (such as _m for meters).

To set the default units, use **setUnits()**.

```
getUnits() [ENTER]
        {"SI"  "Area"  "NONE"
          "Capacitance"  "_F"
            "Charge"  "_coul"
                            … }
```

Note: Your screen may display different default units.

Goto CATALOG

Goto *labelName*

Transfers program control to the label *labelName*.

labelName must be defined in the same program using a **Lbl** instruction.

Program segment:

```
  ⋮
:0→temp
:1→i
:Lbl TOP
:  temp+i→temp
:  If i<10 Then
:    i+1→i
:    Goto TOP
:  EndIf
:Disp temp
  ⋮
```

Graph CATALOG

Graph *expression1*[, *expression2*] [, *var1*] [, *var2*]

The Smart Graph feature graphs the requested expressions/ functions using the current graphing mode.

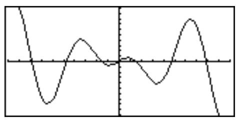
Expressions entered using the **Graph** or **Table** commands are assigned increasing function numbers starting with 1. They can be modified or individually deleted using the edit functions available when the table is displayed by pressing [F4] Header. The currently selected Y= functions are ignored.

If you omit an optional *var* argument, **Graph** uses the independent variable of the current graphing mode.

Note: Not all optional arguments are valid in all modes because you can never have all four arguments at the same time.

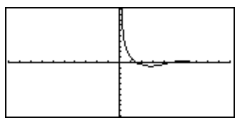
In function graphing mode and ZoomStd window:

```
Graph 1.25a*cos(a),a [ENTER]
```



In parametric graphing mode and ZoomStd window:

```
Graph
time,2cos(time)/time,time [ENTER]
```



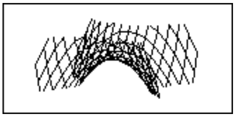
Some valid variations of this instruction are:

Function graphing	**Graph** *expr*, *x*
Parametric graphing	**Graph** *xExpr*, *yExpr*, *t*
Polar graphing	**Graph** *expr*, θ
Sequence graphing	Not allowed.
3D graphing	**Graph** *expr*, *x*, *y*
Diff Equations graphing	Not allowed.

Note: Use **ClrGraph** to clear these functions, or go to the Y= Editor to re-enable the system Y= functions.

In 3D graphing mode:

```
Graph (v^2 – w^2)/4,v,w [ENTER]
```



▶Hex   MATH/Base menu

integer1 **▶Hex** ⇒ *integer*

Converts *integer1* to a hexadecimal number. Binary or hexadecimal numbers always have a 0b or 0h prefix, respectively.

Zero, not the letter O, followed by b or h.

0b *binaryNumber*
0h *hexadecimalNumber*

A binary number can have up to 32 digits. A hexadecimal number can have up to 8.

Without a prefix, *integer1* is treated as decimal (base 10). The result is displayed in hexadecimal, regardless of the Base mode.

If you enter a decimal integer that is too large for a signed, 32-bit binary form, a symmetric modulo operation is used to bring the value into the appropriate range.

```
256▶Hex [ENTER]                 0h100
0b111100001111▶Hex [ENTER]      0hF0F
```

identity()   MATH/Matrix menu

identity(*expression***)** ⇒ *matrix*

Returns the identity matrix with a dimension of *expression*.

expression must evaluate to a positive integer.

```
identity(4) [ENTER]
```

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

If   CATALOG

If *Boolean expression*
statement

If *Boolean expression* **Then**
block
EndIf

If *Boolean expression* evaluates to true, executes the single statement *statement* or the block of statements *block* before continuing execution.

If *Boolean expression* evaluates to false, continues execution without executing the statement or block of statements.

block can be either a single statement or a sequence of statements separated with the ":" character.

Program segment:

```
   :
:If x<0
:Disp "x is negative"
   :
```

—or—

```
   :
:If x<0 Then
:  Disp "x is negative"
:  abs(x)→x
:EndIf
   :
```

If *Boolean expression* **Then**
block1
Else
block2
EndIf

If *Boolean expression* evaluates to true, executes *block1* and then skips *block2*.

If *Boolean expression* evaluates to false, skips *block1* but executes *block2*.

block1 and *block2* can be a single statement.

Program segment:

```
   :
:If x<0 Then
:  Disp "x is negative"
:    Else
:  Disp "x is positive or zero"
:EndIf
   :
```

If *Boolean expression1* **Then**
block1
ElseIf *Boolean expression2* **Then**
block2
⋮
ElseIf *Boolean expressionN* **Then**
blockN
EndIf

Allows for program branching. If *Boolean expression1* evaluates to true, executes *block1*. If *Boolean expression1* evaluates to false, evaluates *Boolean expression2*, etc.

Program segment:

```
   :
:If choice=1 Then
:  Goto option1
:  ElseIf choice=2 Then
:    Goto option2
:  ElseIf choice=3 Then
:    Goto option3
:  ElseIf choice=4 Then
:    Disp "Exiting Program"
:    Return
:EndIf
   :
```

imag() MATH/Complex menu

imag(*expression1***)** ⇒ *expression*

imag(*expression1***)** returns the imaginary part of the argument.

Note: All undefined variables are treated as real variables. See also **real()**.

```
imag(1+2i) [ENTER]          2
imag(z) [ENTER]             0
imag(x+iy) [ENTER]          y
```

imag(*list1***)** ⇒ *list*

Returns a list of the imaginary parts of the elements.

```
imag({⁻3,4⁻i,i}) [ENTER]    {0 ⁻1 1}
```

imag(*matrix1***)** ⇒ *matrix*

Returns a matrix of the imaginary parts of the elements.

```
imag([a,b;ic,id]) [ENTER]   [0 0]
                            [c d]
```

Indirection See **#()**, page 904.

Input CATALOG

Input

Pauses the program, displays the current Graph screen, and lets you update variables *xc* and *yc* (also *rc* and θ*c* for polar coordinate mode) by positioning the graph cursor.

When you press [ENTER], the program resumes.

Program segment:

```
   :
:© Get 10 points from the Graph
   Screen
:For i,1,10
:  Input
:  xc→XLIST[i]
:  yc→YLIST[i]
:EndFor
   :
```

Input [*promptString*,] *var*

Input [*promptString*], *var* pauses the program, displays *promptString* on the Program I/O screen, waits for you to enter an expression, and stores the expression in variable *var*.

If you omit *promptString*, "?" is displayed as a prompt.

Program segment:

```
   :
:For i,1,9,1
:  "Enter x" & string(i)→str1
:  Input str1,#(right(str1,2))
:EndFor
   :
```

InputStr CATALOG

InputStr [*promptString*,] *var*

Pauses the program, displays *promptString* on the Program I/O screen, waits for you to enter a response, and stores your response as a string in variable *var*.

If you omit *promptString*, "?" is displayed as a prompt.

Note: The difference between **Input** and **InputStr** is that **InputStr** always stores the result as a string so that " " are not required.

Program segment:

```
   :
:InputStr "Enter Your Name",str1
   :
```

inString() MATH/String menu

inString(*srcString*, *subString*[, *start*]**)** ⇒ *integer*

Returns the character position in string *srcString* at which the first occurrence of string *subString* begins.

start, if included, specifies the character position within *srcString* where the search begins. Default = 1 (the first character of *srcString*).

If *srcString* does not contain *subString* or *start* is > the length of *srcString*, returns zero.

```
inString("Hello there","the")
[ENTER]                                7

"ABCEFG"→s1:If inString(s1,
"D")=0:Disp "D not found." [ENTER]
                            D not found.
```

int() CATALOG

int(*expression***)** ⇒ *integer*
int(*list1***)** ⇒ *list*
int(*matrix1***)** ⇒ *matrix*

Returns the greatest integer that is less than or equal to the argument. This function is identical to **floor()**.

The argument can be a real or a complex number.

For a list or matrix, returns the greatest integer of each of the elements.

```
int(⁻2.5) [ENTER]                     ⁻3.

int([-1.234,0,0.37]) [ENTER]
                              [-2. 0 0.]
```

intDiv() CATALOG

intDiv(*number1*, *number2***)** ⇒ *integer*
intDiv(*list1*, *list2***)** ⇒ *list*
intDiv(*matrix1*, *matrix2***)** ⇒ *matrix*

Returns the signed integer part of argument 1 divided by argument 2.

For lists and matrices returns the signed integer part of argument 1 divided by argument 2 for each element pair.

```
intDiv(⁻7,2) [ENTER]                   ⁻3

intDiv(4,5) [ENTER]                     0

intDiv({12,⁻14,⁻16},{5,4,⁻3})
[ENTER]
                                {2 ⁻3 5}
```

integrate

See ∫(), page 902.

iPart() — MATH/Number menu

iPart(*number***)** ⇒ *integer*
iPart(*list1***)** ⇒ *list*
iPart(*matrix1***)** ⇒ *matrix*

Returns the integer part of the argument.

For lists and matrices, returns the integer part of each element.

The argument can be a real or a complex number.

```
iPart(⁻1.234) [ENTER]                    ⁻1.

iPart({3/2,⁻2.3,7.003}) [ENTER]
                                {1 ⁻2. 7.}
```

isClkOn() — CATALOG

isClkOn() ⇒ *true,false*

Determines if the clock is ON or OFF. Returns true if the clock is ON. Returns false if the clock is OFF.

isPrime() — MATH/Test menu

isPrime(*number***)** ⇒ *Boolean constant expression*

Returns true or false to indicate if *number* is a whole number ≥ 2 that is evenly divisible only by itself and 1.

If *number* exceeds about 306 digits and has no factors ≤ 1021, **isPrime(***number***)** displays an error message.

If you merely want to determine if *number* is prime, use **isPrime()** instead of **factor()**. It is much faster, particularly if *number* is not prime and has a second-largest factor that exceeds about five digits.

```
IsPrime(5) [ENTER]                  true
IsPrime(6) [ENTER]                 false
```

Function to find the next prime after a specified number:

```
Define nextPrim(n)=Func:Loop:
n+1→n:if isPrime(n):return n:
EndLoop:EndFunc [ENTER]             Done

nextPrim(7) [ENTER]                   11
```

Item — CATALOG

Item *itemNameString*
Item *itemNameString***,** *label*

Valid only within a **Custom...EndCustm** or **ToolBar...EndTBar** block. Sets up a drop-down menu element to let you paste text to the cursor position (**Custom**) or branch to a label (**ToolBar**).

Note: Branching to a label is not allowed within a **Custom** block.

See **Custom** example.

Lbl — CATALOG

Lbl *labelName*

Defines a label with the name *labelName* in the program.

You can use a **Goto** *labelName* instruction to transfer program control to the instruction immediately following the label.

labelName must meet the same naming requirements as a variable name.

Program segment:

```
  ⋮
:Lbl lbl1
:InputStr "Enter password",
str1
:If str1≠password
:  Goto lbl1
:Disp "Welcome to ..."
  ⋮
```

lcm() — MATH/Number menu

lcm(*number1*, *number2***)** ⇒ *expression*
lcm(*list1*, *list2***)** ⇒ *list*
lcm(*matrix1*, *matrix2***)** ⇒ *matrix*

Returns the least common multiple of the two arguments. The **lcm** of two fractions is the **lcm** of their numerators divided by the **gcd** of their denominators. The **lcm** of fractional floating-point numbers is their product.

For two lists or matrices, returns the least common multiples of the corresponding elements.

```
lcm(6,9) [ENTER]                                18

lcm({1/3,-14,16},{2/15,7,5}) [ENTER]
                                      {2/3 14 80}
```

left() — MATH/String menu

left(*sourceString*[**,** *num*]**)** ⇒ *string*

Returns the leftmost *num* characters contained in character string *sourceString*.

If you omit *num*, returns all of *sourceString*.

```
left("Hello",2) [ENTER]                        "He"
```

---

left(*list1*[**,** *num*]**)** ⇒ *list*

Returns the leftmost *num* elements contained in *list1*.

If you omit *num*, returns all of *list1*.

```
left({1,3,-2,4},3) [ENTER]
                                         {1 3 -2}
```

---

left(*comparison***)** ⇒ *expression*

Returns the left-hand side of an equation or inequality.

```
left(x<3) [ENTER]                                 x
```

limit() — MATH/Calculus menu

limit(*expression1*, *var*, *point*[**,** *direction*]**)** ⇒ *expression*
limit(*list1*, *var*, *point*[**,** *direction*]**)** ⇒ *list*
limit(*matrix1*, *var*, *point*[**,** *direction*]**)** ⇒ *matrix*

Returns the limit requested.

direction: negative=from left, positive=from right, otherwise=both. (If omitted, *direction* defaults to both.)

Limits at positive ∞ and at negative ∞ are always converted to one-sided limits from the finite side.

Depending on the circumstances, **limit()** returns itself or undef when it cannot determine a unique limit. This does not necessarily mean that a unique limit does not exist. undef means that the result is either an unknown number with finite or infinite magnitude, or it is the entire set of such numbers.

```
limit(2x+3,x,5) [ENTER]                          13

limit(1/x,x,0,1) [ENTER]                          ∞

limit(sin(x)/x,x,0) [ENTER]                       1

limit((sin(x+h)-sin(x))/h,h,0) [ENTER]
                                             cos(x)

limit((1+1/n)^n,n,∞) [ENTER]                      e
```

limit() uses methods such as L'Hopital's rule, so there are unique limits that it cannot determine. If *expression1* contains undefined variables other than *var*, you might have to constrain them to obtain a more concise result.

Limits can be very sensitive to rounding error. When possible, avoid the APPROX setting of the Exact/Approx mode and approximate numbers when computing limits. Otherwise, limits that should be zero or have infinite magnitude probably will not, and limits that should have finite non-zero magnitude might not.

```
limit(a^x,x,∞) [ENTER]                  undef
limit(a^x,x,∞)|a>1 [ENTER]                  ∞
limit(a^x,x,∞)|a>0 and a<1
[ENTER]                                     0
```

Line   CATALOG

Line *xStart*, *yStart*, *xEnd*, *yEnd*[, *drawMode*]

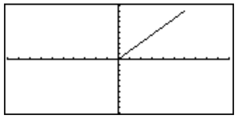
Displays the Graph screen and draws, erases, or inverts a line segment between the window coordinates (*xStart*, *yStart*) and (*xEnd*, *yEnd*), including both endpoints.

If *drawMode* = 1, draws the line (default).
If *drawMode* = 0, turns off the line.
If *drawMode* = ⁻1, turns a line that is on to off or off to on (inverts pixels along the line).

Note: Regraphing erases all drawn items. See also **PxlLine**.

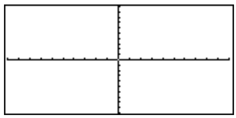
In the ZoomStd window, draw a line and then erase it.

```
Line 0,0,6,9 [ENTER]
```



[HOME]
[◆] [CALC HOME]

```
Line 0,0,6,9,0 [ENTER]
```



LineHorz   CATALOG

LineHorz *y* [, *drawMode*]

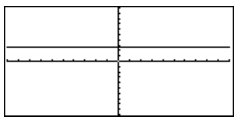
Displays the Graph screen and draws, erases, or inverts a horizontal line at window position *y*.

If *drawMode* = 1, draws the line (default).
If *drawMode* = 0, turns off the line.
If *drawMode* = ⁻1, turns a line that is on to off or off to on (inverts pixels along the line).

Note: Regraphing erases all drawn items. See also **PxlHorz**.

In a ZoomStd window:

```
LineHorz 2.5 [ENTER]
```



LineTan CATALOG

LineTan *expression1*, *expression2*

Displays the Graph screen and draws a line tangent to *expression1* at the point specified.

expression1 is an expression or the name of a function, where x is assumed to be the independent variable, and *expression2* is the x value of the point that is tangent.

Note: In the example shown, *expression1* is graphed separately. **LineTan** does not graph *expression1.*

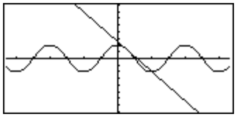
In function graphing mode and a ZoomTrig window:

Graph cos(x)

[HOME]

[◆] [CALC HOME]

LineTan cos(x),π/4 [ENTER]



LineVert CATALOG

LineVert *x* [, *drawMode*]

Displays the Graph screen and draws, erases, or inverts a vertical line at window position *x*.

If *drawMode* = 1, draws the line (default).
If *drawMode* = 0, turns off the line.
If *drawMode* = ⁻1, turns a line that is on to off or off to on (inverts pixels along the line).

Note: Regraphing erases all drawn items. See also **PxlVert**.

In a ZoomStd window:

LineVert ⁻2.5 [ENTER]

LinReg MATH/Statistics/Regressions menu

LinReg *list1*, *list2*[, [*list3*] [, *list4*, *list5*]]

Calculates the linear regression and updates all the system statistics variables.

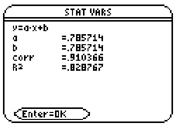
All the lists must have equal dimensions except for *list5*.

list1 represents xlist.
list2 represents ylist.
list3 represents frequency.
list4 represents category codes.
list5 represents category include list.

Note: *list1* through *list4* must be a variable name or c1–c99 (columns in the last data variable shown in the Data/Matrix Editor). *list5* does not have to be a variable name and cannot be c1–c99.

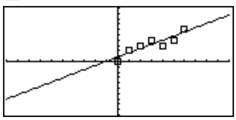
In function graphing mode:

```
{0,1,2,3,4,5,6}→L1 [ENTER]
                              {0 1 2 ...}
{0,2,3,4,3,4,6}→L2 [ENTER]
                              {0 2 3 ...}
LinReg L1,L2 [ENTER]                 Done
ShowStat [ENTER]
```




```
[ENTER]
Regeq(x)→y1(x) [ENTER]               Done
NewPlot 1,1,L1,L2 [ENTER]            Done
```

[◆] [GRAPH]



∆list() MATH/List menu

∆list(*list1***)** ⇒ *list*

Returns a list containing the differences between consecutive elements in *list1*. Each element of *list1* is subtracted from the next element of *list1*. The resulting list is always one element shorter than the original *list1*.

```
∆list({20,30,45,70}) [ENTER]
                              {10,15,25}
```

list▸mat() MATH/List menu

list▸mat(*list* [**,** *elementsPerRow*]**)** ⇒ *matrix*

Returns a matrix filled row-by-row with the elements from *list*.

elementsPerRow, if included, specifies the number of elements per row. Default is the number of elements in *list* (one row).

If *list* does not fill the resulting matrix, zeros are added.

```
list▸mat({1,2,3}) [ENTER]          [1 2 3]

list▸mat({1,2,3,4,5},2) [ENTER]
                                   [1 2]
                                   [3 4]
                                   [5 0]
```

ln() [2nd] [LN] key [LN] key

ln(*expression1***)** ⇒ *expression*
ln(*list1***)** ⇒ *list*

Returns the natural logarithm of the argument.

For a list, returns the natural logarithms of the elements.

```
ln(2.0) [ENTER]                      .693...
```

If complex format mode is REAL:

```
ln({⁻3,1.2,5}) [ENTER]
                   Error: Non-real result
```

If complex format mode is RECTANGULAR:

```
ln({⁻3,1.2,5}) [ENTER]
         {ln(3) + π•i  .182...  ln(5)}
```

---

ln(*squareMatrix1***)** ⇒ *squareMatrix*

Returns the matrix natural logarithm of *squareMatrix1*. This is *not* the same as calculating the natural logarithm of each element. For information about the calculation method, refer to **cos()** on.

squareMatrix1 must be diagonalizable. The result always contains floating-point numbers.

In Radian angle mode and Rectangular complex format mode:

```
ln([1,5,3;4,2,1;6,⁻2,1]) [ENTER]

[1.831...+1.734...•i   .009...−1.490...•i   ...]
[.448...−.725...•i     1.064...+.623•i      ...]
[⁻.266...−2.083...•i   1.124...+1.790...•i  ...]
```

LnReg   MATH/Statistics/Regressions menu

LnReg *list1*, *list2*[, [*list3*] [, *list4*, *list5*]]

Calculates the logarithmic regression and updates all the system statistics variables.

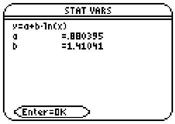
All the lists must have equal dimensions except for *list5*.

list1 represents xlist.
list2 represents ylist.
list3 represents frequency.
list4 represents category codes.
list5 represents category include list.

Note: *list1* through *list4* must be a variable name or c1–c99 (columns in the last data variable shown in the Data/Matrix Editor). *list5* does not have to be a variable name and cannot be c1–c99.

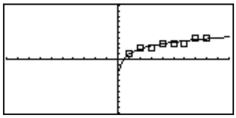
In function graphing mode:

```
{1,2,3,4,5,6,7,8}→L1 [ENTER]
                         {1 2 3 ...}
{1,2,2,3,3,3,4,4}→L2 [ENTER]
                         {1 2 2 ...}
LnReg L1,L2 [ENTER]              Done
ShowStat [ENTER]
```




```
[ENTER]
Regeq(x)→y1(x) [ENTER]           Done
NewPlot 1,1,L1,L2 [ENTER]        Done
```

[◆] [GRAPH]



Local   CATALOG

Local *var1*[, *var2*] [, *var3*] ...

Declares the specified *vars* as local variables. Those variables exist only during evaluation of a program or function and are deleted when the program or function finishes execution.

Note: Local variables save memory because they only exist temporarily. Also, they do not disturb any existing global variable values. Local variables must be used for **For** loops and for temporarily saving values in a multi-line function since modifications on global variables are not allowed in a function.

Program listing:

```
:prgmname()
:Prgm
:Local x,y
:Input "Enter x",x
:Input "Enter y",y
:Disp x*y
:EndPrgm
```

Note: *x* and *y* do not exist after the program executes.

Lock   CATALOG

Lock *var1*[, *var2*] ...

Locks the specified variables. This prevents you from accidentally deleting or changing the variable without first using the unlock instruction on that variable.

In the example to the right, the variable L1 is locked and cannot be deleted or modified.

Note: The variables can be unlocked using the **Unlock** command.

```
{1,2,3,4}→L1 [ENTER]        {1,2,3,4}

Lock L1 [ENTER]                  Done

DelVar L1 [ENTER]
Error: Variable is locked or protected
```

log() CATALOG

log(*expression1***)** ⇒ *expression*
log(*list1***)** ⇒ *list*

Returns the base-10 logarithm of the argument.

For a list, returns the base-10 logs of the elements.

```
log(2.0) [ENTER]                    .301…
```

If complex format mode is REAL:

```
log({⁻3,1.2,5}) [ENTER]
   Error: Non-real result
```

If complex format mode is RECTANGULAR:

```
log({⁻3,1.2,5}) [ENTER]
```

$$\left\{\frac{\ln(3)}{\ln(10)} + \frac{\pi}{\ln(10)}\cdot i \quad .079\ldots \quad \frac{\ln(5)}{\ln(10)}\right\}$$

---

log(*squareMatrix1***)** ⇒ *squareMatrix*

Returns the matrix base-10 logarithm of *squareMatrix1*. This is *not* the same as calculating the base-10 logarithm of each element. For information about the calculation method, refer to **cos()**.

squareMatrix1 must be diagonalizable. The result always contains floating-point numbers.

In Radian angle mode and Rectangular complex format mode:

```
log([1,5,3;4,2,1;6,⁻2,1]) [ENTER]
```

$$\begin{bmatrix} .795\ldots+.753\ldots\cdot i & .003\ldots-.647\ldots\cdot i & \ldots \\ .194\ldots-.315\ldots\cdot i & .462\ldots+.270\cdot i & \ldots \\ ^-.115\ldots-.904\ldots\cdot i & .488\ldots+.777\ldots\cdot i & \ldots \end{bmatrix}$$

Logistic MATH/Statistics/Regressions menu

Logistic *list1*, *list2* [, [*iterations*], [*list3*] [, *list4*, *list5*]]

Calculates the logistic regression and updates all the system statistics variables.

All the lists must have equal dimensions except for *list5*.

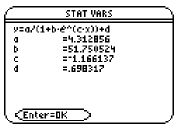
list1 represents xlist.
list2 represents ylist.
list3 represents frequency.
list4 represents category codes.
list5 represents category include list.

iterations specifies the maximum number of times a solution will be attempted. If omitted, 64 is used. Typically, larger values result in better accuracy but longer execution times, and vice versa.

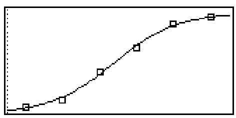
Note: *list1* through *list4* must be a variable name or c1–c99 (columns in the last data variable shown in the Data/Matrix Editor). *list5* does not have to be a variable name and cannot be c1–c99 .

In function graphing mode:

```
{1,2,3,4,5,6}→L1 [ENTER] {1 2 3 …}
{1,1.3,2.5,3.5,4.5,4.8}→L2
[ENTER]
                         {1 1.3 2.5 …}
Logistic L1,L2 [ENTER]            Done
ShowStat [ENTER]
```




```
[ENTER]
regeq(x)→y1(x) [ENTER]            Done
NewPlot 1,1,L1,L2 [ENTER]         Done
[◆] [GRAPH]
[F2] 9
```



Loop — CATALOG

Loop
block
EndLoop

Repeatedly executes the statements in *block*. Note that the loop will be executed endlessly, unless a **Goto** or **Exit** instruction is executed within *block*.

block is a sequence of statements separated with the ":" character.

Program segment:

```
  :
:1→i
:Loop
:  Rand(6)→die1
:  Rand(6)→die2
:  If die1=6 and die2=6
:    Goto End
:  i+1→i
:EndLoop
:Lbl End
:Disp "The number of rolls is", i
  :
```

LU — MATH/Matrix menu

LU *matrix*, *lMatName*, *uMatName*, *pMatName*[, *tol*]

Calculates the Doolittle LU (lower-upper) decomposition of a real or complex *matrix*. The lower triangular matrix is stored in *lMatName*, the upper triangular matrix in *uMatName*, and the permutation matrix (which describes the row swaps done during the calculation) in *pMatName*.

lMatName ∗ *uMatName* = *pMatName* ∗ *matrix*

Optionally, any matrix element is treated as zero if its absolute value is less than *tol*. This tolerance is used only if the matrix has floating-point entries and does not contain any symbolic variables that have not been assigned a value. Otherwise, *tol* is ignored.

- If you use [◆] [ENTER] or set the mode to Exact/Approx=APPROXIMATE, computations are done using floating-point arithmetic.
- If *tol* is omitted or not used, the default tolerance is calculated as:

  5E⁻14 ∗ **max(dim(***matrix***))** ∗ **rowNorm(***matrix***)**

The **LU** factorization algorithm uses partial pivoting with row interchanges.

`[6,12,18;5,14,31;3,8,18]→m1` [ENTER]

$$\begin{bmatrix}6 & 12 & 18\\5 & 14 & 31\\3 & 8 & 18\end{bmatrix}$$

`LU m1,lower,upper,perm` [ENTER] Done

`lower` [ENTER] $\begin{bmatrix}1 & 0 & 0\\5/6 & 1 & 0\\1/2 & 1/2 & 1\end{bmatrix}$

`upper` [ENTER] $\begin{bmatrix}6 & 12 & 18\\0 & 4 & 16\\0 & 0 & 1\end{bmatrix}$

`perm` [ENTER] $\begin{bmatrix}1 & 0 & 0\\0 & 1 & 0\\0 & 0 & 1\end{bmatrix}$

`[m,n;o,p]→m1` [ENTER] $\begin{bmatrix}m & n\\o & p\end{bmatrix}$

`LU m1,lower,upper,perm` [ENTER] Done

`lower` [ENTER] $\begin{bmatrix}1 & 0\\\frac{m}{o} & 1\end{bmatrix}$

`upper` [ENTER] $\begin{bmatrix}o & p\\0 & n-\frac{m\cdot p}{o}\end{bmatrix}$

`perm` [ENTER] $\begin{bmatrix}0 & 1\\1 & 0\end{bmatrix}$

mat▶list() MATH/List menu

mat▶list(*matrix***)** ⇒ *list*

Returns a list filled with the elements in *matrix*. The elements are copied from *matrix* row by row.

```
mat▶list([1,2,3]) [ENTER]    {1 2 3}
[1,2,3;4,5,6]→M1 [ENTER]
                              [1 2 3]
                              [4 5 6]
mat▶list(M1) [ENTER]   {1 2 3 4 5 6}
```

max() MATH/List menu

max(*expression1***,** *expression2***)** ⇒ *expression*
max(*list1***,** *list2***)** ⇒ *list*
max(*matrix1***,** *matrix2***)** ⇒ *matrix*

Returns the maximum of the two arguments. If the arguments are two lists or matrices, returns a list or matrix containing the maximum value of each pair of corresponding elements.

```
max(2.3,1.4) [ENTER]              2.3
max({1,2},{⁻4,3}) [ENTER]       {1 3}
```

---

max(*list***)** ⇒ *expression*

Returns the maximum element in *list*.

```
max({0,1,⁻7,1.3,.5}) [ENTER]    1.3
```

---

max(*matrix1***)** ⇒ *matrix*

Returns a row vector containing the maximum element of each column in *matrix1*.

Note: See also **fMax()** and **min()**.

```
max([1,⁻3,7;⁻4,0,.3]) [ENTER]
                          [1 0 7]
```

mean() MATH/Statistics menu

mean(*list*[, *freqlist*]**)** ⇒ *expression*

Returns the mean of the elements in *list*.

Each *freqlist* element counts the number of consecutive occurrences of the corresponding element in *list*.

```
mean({.2,0,1,⁻.3,.4}) [ENTER]    .26
mean({1,2,3},{3,2,1}) [ENTER]    5/3
```

---

mean(*matrix1*[, *freqmatrix*]**)** ⇒ *matrix*

Returns a row vector of the means of all the columns in *matrix1*.

Each *freqmatrix* element counts the number of consecutive occurrences of the corresponding element in *matrix1*.

In vector format rectangular mode:

```
mean([.2,0;-1,3;.4,-.5]) [ENTER]
                    [⁻.133... .833...]
mean([1/5,0;-1,3;2/5,-1/2])
[ENTER]
                        [⁻2/15  5/6]
mean([1,2;3,4;5,6],[5,3;4,1;
6,2]) [ENTER]         [47/15, 11/3]
```

median() MATH/Statistics menu

median(*list***)** ⇒ *expression*

Returns the median of the elements in *list1*.

```
median({.2,0,1,⁻.3,.4}) [ENTER]   .2
```

---

median(*matrix1***)** ⇒ *matrix*

Returns a row vector containing the medians of the columns in *matrix1*.

Note: All entries in the list or matrix must simplify to numbers.

```
median([.2,0;1,⁻.3;.4,⁻.5])
[ENTER]
                        [.4 ⁻.3]
```

MedMed MATH/Statistics/Regressions menu

MedMed *list1*, *list2*[, [*list3*] [, *list4*, *list5*]]

Calculates the median-median line and updates all the system statistics variables.

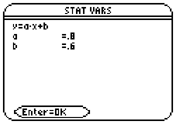
All the lists must have equal dimensions except for *list5*.

list1 represents xlist.
list2 represents ylist.
list3 represents frequency.
list4 represents category codes.
list5 represents category include list.

Note: *list1* through *list4* must be a variable name or c1–c99 (columns in the last data variable shown in the Data/Matrix Editor). *list5* does not have to be a variable name and cannot be c1–c99.

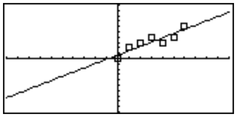
In function graphing mode:

```
{0,1,2,3,4,5,6}→L1 [ENTER] {0 1 2 ...}
{0,2,3,4,3,4,6}→L2 [ENTER] {0 2 3 ...
MedMed L1,L2 [ENTER]                Done
ShowStat [ENTER]
```




```
[ENTER]
Regeq(x)→y1(x) [ENTER]             Done
NewPlot 1,1,L1,L2 [ENTER]          Done
```

[◆] [GRAPH]



mid() MATH/String menu

mid(*sourceString*, *start*[, *count*]**)** ⇒ *string*

Returns *count* characters from character string *sourceString*, beginning with character number *start*.

If *count* is omitted or is greater than the dimension of *sourceString*, returns all characters from *sourceString*, beginning with character number *start*.

count must be ≥ 0. If *count* = 0, returns an empty string.

```
mid("Hello there",2) [ENTER]
                          "ello there"
mid("Hello there",7,3) [ENTER]
                                 "the"
mid("Hello there",1,5) [ENTER]
                               "Hello"
mid("Hello there",1,0) [ENTER]
                                    ""
```

---

mid(*sourceList*, *start* [, *count*]**)** ⇒ *list*

Returns *count* elements from *sourceList*, beginning with element number *start*.

If *count* is omitted or is greater than the dimension of *sourceList*, returns all elements from *sourceList*, beginning with element number *start*.

count must be ≥ 0. If count = 0, returns an empty list.

```
mid({9,8,7,6},3) [ENTER]         {7 6}
mid({9,8,7,6},2,2) [ENTER]       {8 7}
mid({9,8,7,6},1,2) [ENTER]       {9 8}
mid({9,8,7,6},1,0) [ENTER]          {}
```

---

mid(*sourceStringList*, *start*[, *count*]**)** ⇒ *list*

Returns *count* strings from the list of strings *sourceStringList*, beginning with element number *start*.

```
mid({"A","B","C","D"},2,2) [ENTER]
                             {"B" "C"}
```

min() MATH/List menu

min(*expression1*, *expression2***)** ⇒ *expression*
min(*list1*, *list2***)** ⇒ *list*
min(*matrix1*, *matrix2***)** ⇒ *matrix*

Returns the minimum of the two arguments. If the arguments are two lists or matrices, returns a list or matrix containing the minimum value of each pair of corresponding elements.

```
min(2.3,1.4) [ENTER]                1.4
min({1,2},{⁻4,3}) [ENTER]       {⁻4 2}
```

---

min(*list***)** ⇒ *expression*

Returns the minimum element of *list*.

```
min({0,1,⁻7,1.3,.5}) [ENTER]        ⁻7
```

---

min(*matrix1***)** ⇒ *matrix*

Returns a row vector containing the minimum element of each column in *matrix1*.

Note: See also **fMin()** and **max()**.

```
min([1,⁻3,7;⁻4,0,.3]) [ENTER]
                         [⁻4 ⁻3 .3]
```

mod() MATH/Number menu

mod(*expression1*, *expression2***)** ⇒ *expression*
mod(*list1*, *list2***)** ⇒ *list*
mod(*matrix1*, *matrix2***)** ⇒ *matrix*

Returns the first argument modulo the second argument as defined by the identities:

mod(x,0) ≡ x
mod(x,y) ≡ x− y floor(x/y)

When the second argument is non-zero, the result is periodic in that argument. The result is either zero or has the same sign as the second argument.

If the arguments are two lists or two matrices, returns a list or matrix containing the modulo of each pair of corresponding elements.

Note: See also **remain()**.

```
mod(7,0) [ENTER]                      7
mod(7,3) [ENTER]                      1
mod(⁻7,3) [ENTER]                     2
mod(7,⁻3) [ENTER]                    ⁻2
mod(⁻7,⁻3) [ENTER]                   ⁻1
mod({12,⁻14,16},{9,7,⁻5}) [ENTER]
                            {3 0 ⁻4}
```

MoveVar CATALOG

MoveVar *var*, *oldFolder*, *newFolder*

Moves variable *var* from *oldFolder* to *newFolder*. If *newFolder* does not exist, **MoveVar** creates it.

```
{1,2,3,4}→L1 [ENTER]          {1 2 3 4}
MoveVar L1,Main,Games [ENTER]      Done
```

mRow() MATH/Matrix/Row ops menu

mRow(*expression*, *matrix1*, *index***)** ⇒ *matrix*

Returns a copy of *matrix1* with each element in row *index* of *matrix1* multiplied by *expression*.

```
mRow(⁻1/3,[1,2;3,4],2) [ENTER]
                        [ 1    2  ]
                        [⁻1  ⁻4/3 ]
```

mRowAdd() MATH/Matrix/Row ops menu

mRowAdd(*expression*, *matrix1*, *index1*, *index2***)** ⇒ *matrix*

Returns a copy of *matrix1* with each element in row *index2* of *matrix1* replaced with:

expression × row *index1* + row *index2*

```
mRowAdd(⁻3,[1,2;3,4],1,2) [ENTER]
                           [1   2]
                           [0  -2]

mRowAdd(n,[a,b;c,d],1,2) [ENTER]
                   [a        b    ]
                   [a·n+c    b·n+d]
```

nCr() MATH/Probability menu

nCr(*expression1*, *expression2***)** ⇒ *expression*

For integer *expression1* and *expression2* with *expression1* ≥ *expression2* ≥ 0, **nCr()** is the number of combinations of *expression1* things taken *expression2* at a time. (This is also known as a binomial coefficient.) Both arguments can be integers or symbolic expressions.

nCr(*expression*, 0**)** ⇒ 1

nCr(*expression*, *negInteger***)** ⇒ 0

nCr(*expression*, *posInteger***)** ⇒ *expression*• (*expression*−1)... (*expression*−*posInteger*+1)/ *posInteger*!

nCr(*expression*, *nonInteger***)** ⇒ *expression*!/ ((*expression*−*nonInteger*)!•*nonInteger*!)

`nCr(z,3)` $\frac{z\cdot(z-2)\cdot(z-1)}{6}$

`ans(1)|z=5` 10

`nCr(z,c)` $\frac{z!}{c!(z-c)!}$

`ans(1)/nPr(z,c)` $\frac{1}{c!}$

---

nCr(*list1*, *list2***)** ⇒ *list*

Returns a list of combinations based on the corresponding element pairs in the two lists. The arguments must be the same size list.

`nCr({5,4,3},{2,4,2})` [ENTER]
`{10 1 3}`

---

nCr(*matrix1*, *matrix2***)** ⇒ *matrix*

Returns a matrix of combinations based on the corresponding element pairs in the two matrices. The arguments must be the same size matrix.

`nCr([6,5;4,3],[2,2;2,2])` [ENTER]
$\begin{bmatrix}15 & 10\\ 6 & 3\end{bmatrix}$

nDeriv() MATH/Calculus menu

nDeriv(*expression1*, *var*[, *h*]**)** ⇒ *expression*
nDeriv(*expression1*, *var*, *list***)** ⇒ *list*
nDeriv(*list*, *var*[, *h*]**)** ⇒ *list*
nDeriv(*matrix*, *var*[, *h*]**)** ⇒ *matrix*

Returns the numerical derivative as an expression. Uses the central difference quotient formula.

h is the step value. If *h* is omitted, it defaults to 0.001.

When using *list* or *matrix*, the operation gets mapped across the values in the list or across the matrix elements.

Note: See also **avgRC()** and ***d*()**.

`nDeriv(cos(x),x,h)` [ENTER]
`Error!`

`limit(nDeriv(cos(x),x,h),h,0)` [ENTER]
`⁻sin(x)`

`nDeriv(x^3,x,0.01)` [ENTER]
$3.\cdot(x^2+.000033)$

`nDeriv(cos(x),x)|x=π/2` [ENTER]
`⁻1.`

`nDeriv(x^2,x,{.01,.1})` [ENTER]
`{2.·x 2.·x}`

NewData CATALOG

NewData *dataVar*, *list1*[, *list2*] [, *list3*]...

Creates data variable *dataVar*, where the columns are the lists in order.

Must have at least one list.

list1, *list2*, ..., *listn* can be lists as shown, expressions that resolve to lists, or list variable names.

NewData makes the new variable current in the Data/Matrix Editor.

`NewData mydata,{1,2,3},{4,5,6}` [ENTER]
`Done`

(Go to the Data/Matrix Editor and open the *var* `mydata` to display the data variable below.)

DATA	c1	c2	c3
1	1	4	
2	2	5	
3	3	6	
4			

NewData *dataVar*, *matrix*

Creates data variable *dataVar* based on *matrix*.

NewData sysData, *matrix*

Loads the contents of *matrix* into the system data variable sysData.

NewFold CATALOG

NewFold *folderName*

Creates a user-defined folder with the name *folderName*, and then sets the current folder to that folder. After you execute this instruction, you are in the new folder.

```
NewFold games [ENTER]          Done
```

newList() CATALOG

newList(*numElements***)** ⇒ *list*

Returns a list with a dimension of *numElements*. Each element is zero.

```
newList(4) [ENTER]         {0 0 0 0}
```

newMat() CATALOG also Math/Matrix menu

newMat(*numRows*, *numColumns***)** ⇒ *matrix*

Returns a matrix of zeros with the dimension *numRows* by *numColumns*.

```
newMat(2,3) [ENTER]        [0 0 0]
                           [0 0 0]
```

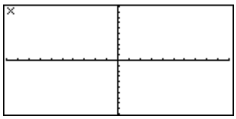
NewPic CATALOG

NewPic *matrix*, *picVar* [, *maxRow*][, *maxCol*]

Creates a pic variable *picVar* based on *matrix*. *matrix* must be an *n×2* matrix in which each row represents a pixel. Pixel coordinates start at 0,0. If *picVar* already exists, **NewPic** replaces it.

The default for *picVar* is the minimum area required for the matrix values. The optional arguments, *maxRow* and *maxCol*, determine the maximum boundary limits for *picVar*.

```
NewPic [1,1;2,2;3,3;4,4;5,5;
  5,1;4,2;2,4;1,5],xpic [ENTER] Done
RclPic xpic [ENTER]
```



NewPlot CATALOG

NewPlot *n*, *type*, *xList* [,[*yList*], [*frqList*], [*catList*], [*includeCatList*], [*mark*] [, *bucketSize*]]

Creates a new plot definition for plot number *n*.

type specifies the type of the graph plot.
1 = scatter plot
2 = xyline plot
3 = box plot
4 = histogram
5 = modified box plot

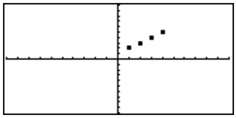
mark specifies the display type of the mark.
1 = ▫ (box)
2 = × (cross)
3 = + (plus)
4 = ▪ (square)
5 = • (dot)

bucketSize is the width of each histogram "bucket" (*type* = 4), and will vary based on the window variables xmin and xmax. *bucketSize* must be >0. Default = 1.

Note: *n* can be 1–9. Lists must be variable names or c1–c99 (columns in the last data variable shown in the Data/Matrix Editor), except for *includeCatList*, which does not have to be a variable name and cannot be c1–c99.

```
FnOff [ENTER]                       Done
PlotsOff [ENTER]                    Done
{1,2,3,4}→L1 [ENTER]           {1 2 3 4}
{2,3,4,5}→L2 [ENTER]           {2 3 4 5}
NewPlot 1,1,L1,L2,,,,4 [ENTER]      Done
```

Press [◆] [GRAPH] to display:



NewProb CATALOG

NewProb

```
NewProb [ENTER]                     Done
```

Performs a variety of operations that let you begin a new problem from a cleared state without resetting the memory.

- Clears all single-character variable names (Clear a–z) in the current folder, unless the variables are locked or archived.
- Turns off all functions and stat plots (**FnOff** and **PlotsOff**) in the current graphing mode.
- Perfoms **ClrDraw**, **ClrErr**, **ClrGraph**, **ClrHome**, **ClrIO**, and **ClrTable**.

nInt() MATH/Calculus menu

nInt(*expression1, var, lower, upper***)** ⇒ *expression*

```
nInt(e^(⁻x^2),x,⁻1,1) [ENTER]
                                1.493...
```

If the integrand *expression1* contains no variable other than *var*, and if *lower* and *upper* are constants, positive ∞, or negative ∞, then **nInt()** returns an approximation of ∫**(***expression1*, *var*, *lower*, *upper***)**. This approximation is a weighted average of some sample values of the integrand in the interval *lower*<*var*<*upper*.

The goal is six significant digits. The adaptive algorithm terminates when it seems likely that the goal has been achieved, or when it seems unlikely that additional samples will yield a worthwhile improvement.

```
nInt(cos(x),x,⁻π,π+1E⁻12) [ENTER]
                           ⁻1.041...E⁻12
```

```
∫(cos(x),x,⁻π,π+10^(⁻12)) [ENTER]
```

$$-\sin\left(\frac{1}{1000000000000}\right)$$

```
ans(1)[◆] [ENTER]                ⁻1.E⁻12
```

A warning is displayed ("Questionable accuracy") when it seems that the goal has not been achieved.

Nest **nInt()** to do multiple numeric integration. Integration limits can depend on integration variables outside them.

```
nInt(nInt(e^(⁻x*y)/√(x^2-y^2),
  y,⁻x,x),x,0,1) [ENTER]      3.304...
```

Note: See also ∫**()**.

norm() MATH/Matrix/Norms menu

norm(*matrix***)** ⇒ *expression*

Returns the Frobenius norm.

```
norm([a,b;c,d]) [ENTER]
```
$\sqrt{a^2+b^2+c^2+d^2}$

```
norm([1,2;3,4]) [ENTER]
```
$\sqrt{30}$

not MATH/Test menu

not *Boolean expression1* ⇒ *Boolean expression*

Returns true, false, or a simplified *Boolean expression1*.

```
not 2>=3 [ENTER]                 true
not x<2 [ENTER]                  x ≥ 2
not not innocent [ENTER]     innocent
```

---

not *integer1* ⇒ *integer*

Returns the one's complement of a real integer. Internally, *integer1* is converted to a signed, 32-bit binary number. The value of each bit is flipped (0 becomes 1, and vice versa) for the one's complement. Results are displayed according to the Base mode.

You can enter the integer in any number base. For a binary or hexadecimal entry, you must use the 0b or 0h prefix, respectively. Without a prefix, the integer is treated as decimal (base 10).

If you enter a decimal integer that is too large for a signed, 32-bit binary form, a symmetric modulo operation is used to bring the value into the appropriate range.

In Hex base mode:

```
not 0h7AC36 [ENTER]        0hFFF853C9
```

Important: Zero, not the letter O.

In Bin base mode:

```
0b100101▶dec [ENTER]               37

not 0b100101 [ENTER]
0b11111111111111111111111111011010

ans(1)▶dec [ENTER]                ⁻38
```

Note: A binary entry can have up to 32 digits (not counting the 0b prefix). A hexadecimal entry can have up to 8 digits.

Note: To type the ▶ conversion operator, press [2nd] [▶]. You can also select base conversions from the MATH/Base menu.

nPr() MATH/Probability menu

nPr(*expression1***,** *expression2***)** ⇒ *expression*

For integer *expression1* and *expression2* with *expression1* ≥ *expression2* ≥ 0, **nPr()** is the number of permutations of *expression1* things taken *expression2* at a time. Both arguments can be integers or symbolic expressions.

nPr(*expression*, 0**)** ⇒ 1

nPr(*expression*, *negInteger***)** ⇒ 1/((*expression*+1) · (*expression*+2)... (*expression*− *negInteger*))

nPr(*expression*, *posInteger***)** ⇒ *expression* · (*expression*− 1)... (*expression*− *posInteger*+1)

nPr(*expression*, *nonInteger***)** ⇒ *expression*!/ (*expression*− *nonInteger*)!

```
nPr(z,3) [ENTER]       z·(z-2)·(z-1)
ans(1)|z=5 [ENTER]                60
```

```
nPr(z,⁻3) [ENTER]
```
$\frac{1}{(z+1)\cdot(z+2)\cdot(z+3)}$

```
nPr(z,c) [ENTER]
```
$\frac{z!}{(z-c)!}$

```
ans(1)*nPr(z-c,⁻c) [ENTER]         1
```

---

nPr(*list1***,** *list2***)** ⇒ *list*

Returns a list of permutations based on the corresponding element pairs in the two lists. The arguments must be the same size list.

```
nPr({5,4,3},{2,4,2}) [ENTER]
                        {20 24 6}
```

nPr(*matrix1*, *matrix2***)** ⇒ *matrix*

Returns a matrix of permutations based on the corresponding element pairs in the two matrices. The arguments must be the same size matrix.

```
nPr([6,5;4,3],[2,2;2,2]) [ENTER]
                        [30  20]
                        [12   6]
```

nSolve() MATH/Algebra menu

nSolve(*equation*, *varOrGuess***)** ⇒ *number or error_string*

Iteratively searches for one approximate real numeric solution to *equation* for its one variable. Specify *varOrGuess* as:

variable
– or –
variable = *real number*

For example, x is valid and so is x=3.

nSolve() is often much faster than **solve()** or **zeros()**, particularly if the "|" operator is used to constrain the search to a small interval containing exactly one simple solution.

nSolve() attempts to determine either one point where the residual is zero or two relatively close points where the residual has opposite signs and the magnitude of the residual is not excessive. If it cannot achieve this using a modest number of sample points, it returns the string "no solution found."

If you use **nSolve()** in a program, you can use **getType()** to check for a numeric result before using it in an algebraic expression.

Note: See also **cSolve()**, **cZeros()**, **solve()**, and **zeros()**.

```
nSolve(x^2+5x-25=9,x) [ENTER]
                              3.844...
nSolve(x^2=4,x=⁻1) [ENTER]        ⁻2.
nSolve(x^2=4,x=1) [ENTER]          2.
```

Note: If there are multiple solutions, you can use a guess to help find a particular solution.

```
nSolve(x^2+5x-25=9,x)|x<0 [ENTER]
                             ⁻8.844...
nSolve(((1+r)^24-1)/r=26,r)|r>
0 and r<.25 [ENTER]            .0068...
nSolve(x^2=⁻1,x) [ENTER]
                  "no solution found"
```

OneVar MATH/Statistics menu

OneVar *list1* [[, *list2*] [, *list3*] [, *list4*]]

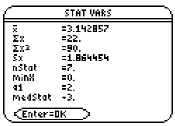
Calculates 1-variable statistics and updates all the system statistics variables.

All the lists must have equal dimensions except for *list4*.

list1 represents xlist.
list2 represents frequency.
list3 represents category codes.
list4 represents category include list.

Note: *list1* through *list3* must be a variable name or c1–c99 (columns in the last data variable shown in the Data/Matrix Editor). *list4* does not have to be a variable name and cannot be c1–c99.

```
{0,2,3,4,3,4,6}→L1 [ENTER]
OneVar L1 [ENTER]                 Done
ShowStat [ENTER]
```



or — MATH/Test menu

Boolean expression1 **or** *Boolean expression2* ⇒ *Boolean expression*

Returns true or false or a simplified form of the original entry.

Returns true if either or both expressions simplify to true. Returns false only if both expressions evaluate to false.

Note: See **xor**.

```
x≥3 or x≥4 [ENTER]                x ≥ 3
```

Program segment:

```
  ⋮
If x<0 or x≥5
  Goto END
  ⋮
If choice=1 or choice=2
  Disp "Wrong choice"
  ⋮
```

integer1 **or** *integer2* ⇒ *integer*

Compares two real integers bit-by-bit using an **or** operation. Internally, both integers are converted to signed, 32-bit binary numbers. When corresponding bits are compared, the result is 1 if either bit is 1; the result is 0 only if both bits are 0. The returned value represents the bit results, and is displayed according to the Base mode.

You can enter the integers in any number base. For a binary or hexadecimal entry, you must use the 0b or 0h prefix, respectively. Without a prefix, integers are treated as decimal (base 10).

If you enter a decimal integer that is too large for a signed, 32-bit binary form, a symmetric modulo operation is used to bring the value into the appropriate range.

Note: See **xor**.

In Hex base mode:

```
0h7AC36 or 0h3D5F [ENTER]    0h7BD7F
```

Important: Zero, not the letter O.

In Bin base mode:

```
0b100101 or 0b100 [ENTER]  0b100101
```

Note: A binary entry can have up to 32 digits (not counting the 0b prefix). A hexadecimal entry can have up to 8 digits.

ord() — MATH/String menu

ord(*string***)** ⇒ *integer*
ord(*list1***)** ⇒ *list*

Returns the numeric code of the first character in character string *string*, or a list of the first characters of each list element.

See Appendix B for a complete listing of character codes.

```
ord("hello") [ENTER]                104
char(104) [ENTER]                   "h"
ord(char(24)) [ENTER]                24
ord({"alpha","beta"}) [ENTER]
                                {97   98}
```

Output — CATALOG

Output *row*, *column*, *exprOrString*

Displays *exprOrString* (an expression or character string) on the Program I/O screen at the text coordinates (*row*, *column*).

An expression can include conversion operations such as ▶**DD** and ▶**Rect**. You can also use the ▶ operator to perform unit and number base conversions.

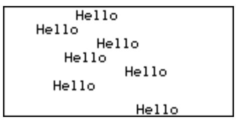
If Pretty Print = ON, *exprOrString* is "pretty printed."

From the Program I/O screen, you can press [F5] to display the Home screen, or a program can use **DispHome**.

Program segment:

```
  ⋮
:RandSeed 1147
:ClrIO
:For i,1,90,10
:  Output i, rand(100),"Hello"
:EndFor
  ⋮
```

Result after execution:



P▸Rx() MATH/Angle menu

P▸Rx(*rExpression*, θ*Expression***)** ⇒ *expression*
P▸Rx(*rList*, θ*List***)** ⇒ *list*
P▸Rx(*rMatrix*, θ*Matrix***)** ⇒ *matrix*

Returns the equivalent x-coordinate of the (r, θ) pair.

Note: The θ argument is interpreted as either a degree or radian angle, according to the current angle mode. If the argument is an expression, you can use ° or ʳ to override the angle mode setting temporarily.

In Radian angle mode:

```
P▸Rx(r,θ) [ENTER]                    cos(θ)·r
P▸Rx(4,60°) [ENTER]                         2
P▸Rx({⁻3,10,1.3},{π/3,⁻π/4,0})
[ENTER]
                         {⁻3/2  5·√2  1.3}
```

P▸Ry() MATH/Angle menu

P▸Ry(*rExpression*, θ*Expression***)** ⇒ *expression*
P▸Ry(*rList*, θ*List***)** ⇒ *list*
P▸Ry(*rMatrix*, θ*Matrix***)** ⇒ *matrix*

Returns the equivalent y-coordinate of the (r, θ) pair.

Note: The θ argument is interpreted as either a degree or radian angle, according to the current angle mode. If the argument is an expression, you can use ° or ʳ to override the angle mode setting temporarily.

In Radian angle mode:

```
P▸Ry(r,θ) [ENTER]                    sin(θ)·r
P▸Ry(4,60°) [ENTER]                      2·√3
P▸Ry({⁻3,10,1.3},{π/3,⁻π/4,0})
[ENTER]
```

$$\left\{ \frac{-3\cdot\sqrt{3}}{2} \quad -5\cdot\sqrt{2} \quad 0. \right\}$$

part() CATALOG

part(*expression1*[**,***nonNegativeInteger*]**)**

This advanced programming function lets you identify and extract all of the sub-expressions in the simplified result of *expression1*.

For example, if *expression1* simplifies to cos(π∗ x+3):

- The **cos()** function has one argument: (π∗ x+3).
- The sum of (π∗ x+3) has two operands: π∗ x and 3.
- The number 3 has no arguments or operands.
- The product π∗ x has two operands: π and x.
- The variable x and the symbolic constant π have no arguments or operands.

If x has a numeric value and you press [◆] [ENTER], the numeric value of π∗ x is calculated, the result is added to 3, and then the cosine is calculated. **cos()** is the **top-level** operator because it is applied **last**.

---

part(*expression1***)** ⇒ *number*

Simplifies *expression1* and returns the number of top-level arguments or operands. This returns 0 if *expression1* is a number, variable, or symbolic constant such as π, *e*, ***i***, or ∞.

```
part(cos(π∗x+3)) [ENTER]                    1
```

Note: cos(π∗ x+3) has one argument.

part(*expression1***, 0)** ⇒ *string*

Simplifies *expression1* and returns a string that contains the top-level function name or operator. This returns **string(***expression1***)** if *expression1* is a number, variable, or symbolic constant such as π, *e*, ***i***, or ∞.

```
part(cos(π∗x+3),0) [ENTER]              "cos"
```

part(*expression1*, *n***)** ⇒ *expression*

Simplifies *expression1* and returns the n^{th} argument or operand, where *n* is > 0 and ≤ the number of top-level arguments or operands returned by **part(***expression1***)**. Otherwise, an error is returned.

```
part(cos(π∗x+3),1) [ENTER]        3+π•x
```

Note: Simplification changed the order of the argument.

By combining the variations of **part()**, you can extract all of the sub-expressions in the simplified result of *expression1*. As shown in the example to the right, you can store an argument or operand and then use **part()** to extract further sub-expressions.

Note: When using **part()**, do not rely on any particular order in sums and products.

```
part(cos(π∗x+3)) [ENTER]                1
part(cos(π∗x+3),0) [ENTER]          "cos"
part(cos(π∗x+3),1)→temp [ENTER]
                                    3+π•x
temp [ENTER]                        π•x+3
part(temp,0) [ENTER]                  "+"
part(temp) [ENTER]                      2
part(temp,2) [ENTER]                    3
part(temp,1)→temp [ENTER]             π•x
part(temp,0) [ENTER]                  "∗"
part(temp) [ENTER]                      2
part(temp,1) [ENTER]                    π
part(temp,2) [ENTER]                    x
```

Expressions such as (x+y+z) and (x− y− z) are represented internally as (x+y)+z and (x− y)− z. This affects the values returned for the first and second argument. There are technical reasons why **part(**x+y+z,1**)** returns y+x instead of x+y.

```
part(x+y+z) [ENTER]                     2
part(x+y+z,2) [ENTER]                   z
part(x+y+z,1) [ENTER]                 y+x
```

Similarly, x∗ y∗ z is represented internally as (x∗ y)∗ z. Again, there are technical reasons why the first argument is returned as y•x instead of x•y.

```
part(x∗y∗z) [ENTER]                     2
part(x∗y∗z,2) [ENTER]                   z
part(x∗y∗z,1) [ENTER]                 y•x
```

When you extract sub-expressions from a matrix, remember that matrices are stored as lists of lists, as illustrated in the example to the right.

```
part([a,b,c;x,y,z],0) [ENTER]         "{"
part([a,b,c;x,y,z]) [ENTER]             2
part([a,b,c;x,y,z],2)→temp
[ENTER]
                                {x  y  z}
part(temp,0) [ENTER]                  "{"
part(temp) [ENTER]                      3
part(temp,3) [ENTER]                    z
delVar temp [ENTER]                  Done
```

The example Program Editor function to the right uses **getType()** and **part()** to partially implement symbolic differentiation. Studying and completing this function can help teach you how to differentiate manually. You could even include functions that the cannot differentiate, such as Bessel functions.

```
:d(y,x)
:Func
:Local f
:If getType(y)="VAR"
:  Return when(y=x,1,0,0)
:If part(y)=0
:  Return 0 © y=π,∞,i,numbers
:part(y,0)→f
:If f="⁻" © if negate
:  Return ⁻d(part(y,1),x)
:If f="−" © if minus
:  Return d(part(y,1),x)
     −d(part(y,2),x)
:If f="+"
:  Return d(part(y,1),x)
     +d(part(y,2),x)
:If f="∗"
:  Return
part(y,1)∗d(part(y,2),x)
     +part(y,2)∗d(part(y,1),x)
:If f="{"
:  Return seq(d(part(y,k),x),
     k,1,part(y))
:Return undef
:EndFunc
```

PassErr CATALOG

PassErr

Passes an error to the next level.

If “errornum” is zero, **PassErr** does not do anything.

The **Else** clause in the program should use **ClrErr** or **PassErr**. If the error is to be processed or ignored, use **ClrErr**. If what to do with the error is not known, use **PassErr** to send it to the next error handler. (See also **ClrErr**.)

See **ClrErr** program listing example.

Pause CATALOG

Pause [*expression*]

Suspends program execution. If you include *expression*, displays *expression* on the Program I/O screen.

expression can include conversion operations such as ▶**DD** and ▶**Rect**. You can also use the ▶ operator to perform unit and number base conversions.

If the result of *expression* is too big to fit on a single screen, you can use the cursor pad to scroll the display.

Program execution resumes when you press [ENTER].

Program segment:

```
  ⋮
:ClrIO
:DelVar temp
:1→temp[1]
:1→temp[2]
:Disp temp[2]
:© Guess the Pattern
:For i,3,20
:  temp[i−2]+temp[i−1]→temp[i]
:  Disp temp[i]
:  Disp temp,"Can you guess the
     next","number?"
:  Pause
:EndFor
  ⋮
```

PlotsOff CATALOG

PlotsOff [**1**] [**, 2**] [**, 3**] **...** [**, 9**]

Turns off the specified plots for graphing. When in 2-graph mode, only affects the active graph.

If no parameters, then turns off all plots.

```
PlotsOff 1,2,5 [ENTER]          Done
PlotsOff [ENTER]                Done
```

PlotsOn CATALOG

PlotsOn [**1**] [**, 2**] [**, 3**] **...** [**, 9**]

Turns on the specified plots for graphing. When in 2-graph mode, only affects the active graph.

If you do not include any arguments, turns on all plots.

```
PlotsOn 2,4,5 [ENTER]           Done
PlotsOn [ENTER]                 Done
```

▸Polar MATH/Matrix/Vector ops menu

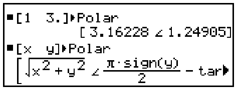
vector **▸Polar**

Displays *vector* in polar form [r ∠θ]. The vector must be of dimension 2 and can be a row or a column.

Note: **▸Polar** is a display-format instruction, not a conversion function. You can use it only at the end of an entry line, and it does not update ans.

Note: See also **▸Rect**.

```
[1,3.]▸Polar [ENTER]
[x,y]▸Polar [ENTER]
```



---

complexValue **▸Polar**

Displays *complexVector* in polar form.

- Degree angle mode returns (r∠θ).
- Radian angle mode returns r$e^{i\theta}$.

complexValue can have any complex form. However, an r$e^{i\theta}$ entry causes an error in Degree angle mode.

Note: You must use the parentheses for an (r∠θ) polar entry.

In Radian angle mode:

3+4*i*▸Polar [ENTER] $e^{i\cdot(\frac{\pi}{2}-\tan^{-1}(3/4))}\cdot 5$

(4∠π/3)▸Polar [ENTER] $e^{\frac{i\cdot\pi}{3}}\cdot 4$

In Degree angle mode:

3+4*i*▸Polar [ENTER] (5∠90−tan⁻¹(3/4))

polyEval() MATH/List menu

polyEval(*list1*, *expression1***)** ⇒ *expression*
polyEval(*list1*, *list2***)** ⇒ *expression*

Interprets the first argument as the coefficient of a descending-degree polynomial, and returns the polynomial evaluated for the value of the second argument.

polyEval({a,b,c},x) [ENTER]
$a\cdot x^2+b\cdot x+c$

polyEval({1,2,3,4},2) [ENTER] 26

polyEval({1,2,3,4},{2,⁻7}) [ENTER] {26 ⁻262}

PopUp CATALOG

PopUp *itemList*, *var*

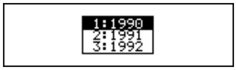
Displays a pop-up menu containing the character strings from *itemList*, waits for you to select an item, and stores the number of your selection in *var*.

The elements of *itemList* must be character strings: {*item1String*, *item2String*, *item3String*, ...}

If *var* already exists and has a valid item number, that item is displayed as the default choice.

itemList must contain at least one choice.

```
PopUp
{"1990","1991","1992"},var1
```
[ENTER]



PowerReg MATH/Statistics/Regressions menu

PowerReg *list1*, *list2*[, [*list3*] [, *list4*, *list5*]]

Calculates the power regression and updates all the system statistics variables.

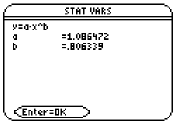
All the lists must have equal dimensions except for *list5*.

list1 represents xlist.
list2 represents ylist.
list3 represents frequency.
list4 represents category codes.
list5 represents category include list.

Note: *list1* through *list4* must be a variable name or c1–c99 (columns in the last data variable shown in the Data/Matrix Editor). *list5* does not have to be a variable name and cannot be c1–c99.

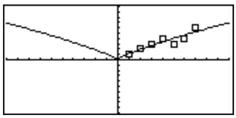
In function graphing mode:

```
{1,2,3,4,5,6,7}→L1 [ENTER]
                        {1 2 3 ...}
{1,2,3,4,3,4,6}→L2 [ENTER]
                        {1 2 3 ...}
PowerReg L1,L2 [ENTER]         Done
ShowStat [ENTER]
```




```
[ENTER]
Regeq(x)→y1(x) [ENTER]         Done
NewPlot 1,1,L1,L2 [ENTER]      Done
```

[♦] [GRAPH]



Prgm CATALOG

Prgm
⋮
EndPrgm

Required instruction that identifies the beginning of a program. Last line of program must be **EndPrgm**.

Program segment:

```
:prgmname()
:Prgm
:
:EndPrgm
```

Product (PI) See Π(), page 903.

product() MATH/List menu

product(*list*[, *start*[, *end*]]**)** ⇒ *expression*

Returns the product of the elements contained in *list*. *Start* and *end* are optional. They specify a range of elements.

```
product({1,2,3,4}) [ENTER]         24
product({2,x,y}) [ENTER]        2·x·y
product({4,5,8,9},2,3) [ENTER]     40
```

product(*matrix1***[**, *start***[**, *end***]])** ⇒ *matrix*

Returns a row vector containing the products of the elements in the columns of *matrix1*. *Start* and *end* are optional. They specify a range of rows.

```
product([1,2,3;4,5,6;7,8,9])
[ENTER]                    [28 80 162]

product([1,2,3;4,5,6;7,8,9],
1,2) [ENTER]                 [4,10,18]
```

Prompt CATALOG

Prompt *var1*[, *var2*] [, *var3*] **...**

Displays a prompt on the Program I/O screen for each variable in the argument list, using the prompt `var1?`. Stores the entered expression in the corresponding variable.

Prompt must have at least one argument.

Program segment:

```
   :
Prompt A,B,C
   :
EndPrgm
```

propFrac() MATH/Algebra menu

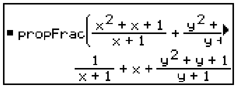
propFrac(*expression1***[**, *var***])** ⇒ *expression*

propFrac(*rational_number***)** returns *rational_number* as the sum of an integer and a fraction having the same sign and a greater denominator magnitude than numerator magnitude.

```
propFrac(4/3) [ENTER]          1 + 1/3
propFrac(⁻4/3) [ENTER]         ⁻1−1/3
```

propFrac(*rational_expression*,*var***)** returns the sum of proper ratios and a polynomial with respect to *var*. The degree of *var* in the denominator exceeds the degree of *var* in the numerator in each proper ratio. Similar powers of *var* are collected. The terms and their factors are sorted with *var* as the main variable.

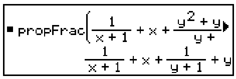
```
propFrac((x^2+x+1)/(x+1)+
  (y^2+y+1)/(y+1),x) [ENTER]
```



If *var* is omitted, a proper fraction expansion is done with respect to the most main variable. The coefficients of the polynomial part are then made proper with respect to their most main variable first and so on.

For rational expressions, **propFrac()** is a faster but less extreme alternative to **expand()**.

```
propFrac(ans(1))
```



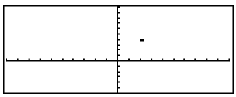
PtChg CATALOG

PtChg *x*, *y*
PtChg *xList*, *yList*

Displays the Graph screen and reverses the screen pixel nearest to window coordinates (*x*, *y*).

Note: PtChg through **PtText** show continuing similar examples.

```
PtChg 2,4 [ENTER]
```

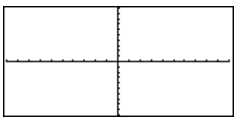


PtOff CATALOG

PtOff *x*, *y*
PtOff *xList*, *yList*

Displays the Graph screen and turns off the screen pixel nearest to window coordinates (*x*, *y*).

```
PtOff 2,4 [ENTER]
```

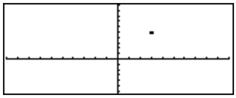


PtOn CATALOG

PtOn *x*, *y*
PtOn *xList*, *yList*

Displays the Graph screen and turns on the screen pixel nearest to window coordinates (*x*, *y*).

`PtOn 3,5` [ENTER]



ptTest() CATALOG

ptTest (*x*, *y***)** ⇒ *Boolean constant expression*
ptTest (*xList*, *yList***)** ⇒ *Boolean constant expression*

Returns true or false. Returns true only if the screen pixel nearest to window coordinates (*x*, *y*) is on.

`ptTest(3,5)` [ENTER] `true`

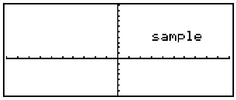
PtText CATALOG

PtText *string*, *x*, *y*

Displays the Graph screen and places the character string *string* on the screen at the pixel nearest the specified (*x*, *y*) window coordinates.

string is positioned with the upper-left corner of its first character at the coordinates.

`PtText "sample",3,5` [ENTER]



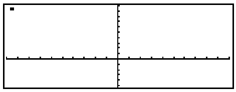
PxlChg CATALOG

PxlChg *row*, *col*
PxlChg *rowList*, *colList*

Displays the Graph screen and reverses the pixel at pixel coordinates (*row*, *col*).

Note: Regraphing erases all drawn items.

`PxlChg 2,4` [ENTER]



PxlCrcl CATALOG

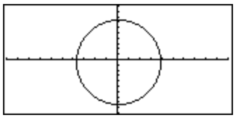
PxlCrcl *row*, *col*, *r* [, *drawMode*]

Displays the Graph screen and draws a circle centered at pixel coordinates (*row*, *col*) with a radius of *r* pixels.

If *drawMode* = 1, draws the circle (default).
If *drawMode* = 0, turns off the circle.
If *drawMode* = -1, inverts pixels along the circle.

Note: Regraphing erases all drawn items. See also **Circle**.

`PxlCrcl 40,80,30,1` [ENTER]
`PxlCrcl 50,125,40,1` [ENTER]



PxlHorz CATALOG

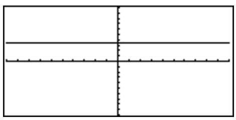
PxlHorz *row* [, *drawMode*]

Displays the Graph screen and draws a horizontal line at pixel position *row*.

If *drawMode* = 1, draws the line (default).
If *drawMode* = 0, turns off the line.
If *drawMode* = -1, turns a line that is on to off or off to on (inverts pixels along the line).

Note: Regraphing erases all drawn items. See also **LineHorz**.

`PxlHorz 25,1` [ENTER]



PxlLine CATALOG

PxlLine *rowStart*, *colStart*, *rowEnd*, *colEnd* [, *drawMode*]

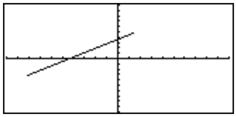
Displays the Graph screen and draws a line between pixel coordinates (*rowStart*, *colStart*) and (*rowEnd*, *colEnd*), including both endpoints.

If *drawMode* = 1, draws the line (default).
If *drawMode* = 0, turns off the line.
If *drawMode* = -1, turns a line that is on to off or off to on (inverts pixels along the line).

Note: Regraphing erases all drawn items. See also **Line.**

PxlLine 50,15,20,90,1 [ENTER]

PxlLine 80,20,30,150,1 [ENTER]



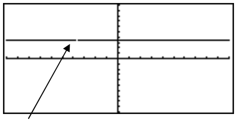
PxlOff CATALOG

PxlOff *row*, *col*
PxlOff *rowList*, *colList*

Displays the Graph screen and turns off the pixel at pixel coordinates (*row*, *col*).

Note: Regraphing erases all drawn items.

PxlHorz 25,1 [ENTER]
PxlOff 25,50 [ENTER]



25,50

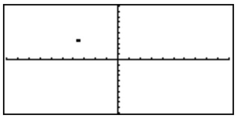
PxlOn CATALOG

PxlOn *row*, *col*
PxlOn *rowList*, *colList*

Displays the Graph screen and turns on the pixel at pixel coordinates (*row*, *col*).

Note: Regraphing erases all drawn items.

PxlOn 25,50 [ENTER]



pxlTest() CATALOG

pxlTest (*row*, *col***)** ⇒ *Boolean expression*
pxlTest (*rowList*, *colList***)** ⇒ *Boolean expression*

Returns true if the pixel at pixel coordinates (*row*, *col*) is on. Returns false if the pixel is off.

Note: Regraphing erases all drawn items.

PxlOn 25,50 [ENTER]

[HOME]

[◆] [CALC HOME]

PxlTest(25,50) [ENTER] true
PxlOff 25,50 [ENTER]

[HOME]

[◆] [CALC HOME]

PxlTest(25,50) [ENTER] false

PxlText CATALOG

PxlText *string*, *row*, *col*

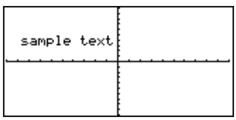
Displays the Graph screen and places character string *string* on the screen, starting at pixel coordinates (*row*, *col*).

string is positioned with the upper-left corner of its first character at the coordinates.

Note: Regraphing erases all drawn items.

PxlText "sample text",20,10 [ENTER]

PxlText "sample text",20,50 [ENTER]



PxlVert CATALOG

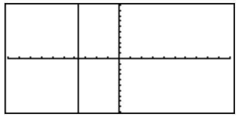
PxlVert *col* [**,** *drawMode*]

Draws a vertical line down the screen at pixel position *col*.

If *drawMode* = 1, draws the line (default).
If *drawMode* = 0, turns off the line.
If *drawMode* = -1, turns a line that is on to off or off to on (inverts pixels along the line).

Note: Regraphing erases all drawn items. See also **LineVert**.

`PxlVert 50,1` [ENTER]



QR MATH/Matrix menu

QR *matrix***,** *qMatName***,** *rMatName*[**,** *tol*]

Calculates the Householder QR factorization of a real or complex *matrix*. The resulting Q and R matrices are stored to the specified *MatNames*. The Q matrix is unitary. The R matrix is upper triangular.

Optionally, any matrix element is treated as zero if its absolute value is less than *tol*. This tolerance is used only if the matrix has floating-point entries and does not contain any symbolic variables that have not been assigned a value. Otherwise, *tol* is ignored.

- If you use [◆] [ENTER] or set the mode to Exact/Approx=APPROXIMATE, computations are done using floating-point arithmetic.
- If *tol* is omitted or not used, the default tolerance is calculated as:

  5E-14 * **max(dim(***matrix***))** * **rowNorm(***matrix***)**

The floating-point number (9.) in m1 causes results to be calculated in floating-point form.

`[1,2,3;4,5,6;7,8,9.]→m1` [ENTER]

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9. \end{bmatrix}$$

`QR m1,qm,rm` [ENTER] Done

`qm` [ENTER]

$$\begin{bmatrix} .123\ldots & .904\ldots & .408\ldots \\ .492\ldots & .301\ldots & ^{-}.816\ldots \\ .861\ldots & ^{-}.301\ldots & .408\ldots \end{bmatrix}$$

`rm` [ENTER]

$$\begin{bmatrix} 8.124\ldots & 9.601\ldots & 11.078\ldots \\ 0. & .904\ldots & 1.809\ldots \\ 0. & 0. & 0. \end{bmatrix}$$

`[m,n;o,p]→m1` [ENTER]

$$\begin{bmatrix} m & n \\ o & p \end{bmatrix}$$

The QR factorization is computed numerically using Householder transformations. The symbolic solution is computed using Gram-Schmidt. The columns in *qMatName* are the orthonormal basis vectors that span the space defined by *matrix*.

`QR m1,qm,rm` [ENTER] Done

`qm` [ENTER]

$$\begin{bmatrix} \frac{m}{\sqrt{m^2+o^2}} & \frac{^{-}\mathrm{sign}(m\cdot p-n\cdot o)\cdot o}{\sqrt{m^2+o^2}} \\ \frac{o}{\sqrt{m^2+o^2}} & \frac{m\cdot \mathrm{sign}(m\cdot p-n\cdot o)}{\sqrt{m^2+o^2}} \end{bmatrix}$$

`rm` [ENTER]

$$\begin{bmatrix} \sqrt{m^2+o^2} & \frac{m\cdot n+o\cdot p}{\sqrt{m^2+o^2}} \\ 0 & \frac{|m\cdot p-n\cdot o|}{\sqrt{m^2+o^2}} \end{bmatrix}$$

QuadReg MATH/Statistics/Regressions menu

QuadReg *list1*, *list2*[, [*list3*] [, *list4*, *list5*]]

Calculates the quadratic polynomial regression and updates the system statistics variables.

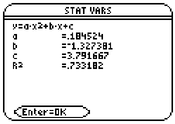
All the lists must have equal dimensions except for *list5*.

list1 represents xlist.
list2 represents ylist.
list3 represents frequency.
list4 represents category codes.
list5 represents category include list.

Note: *list1* through *list4* must be a variable name or c1–c99. (columns in the last data variable shown in the Data/Matrix Editor). *list5* does not have to be a variable name and cannot be c1–c99 .

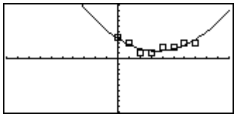
In function graphing mode:

```
{0,1,2,3,4,5,6,7}→L1 [ENTER]
                          {1 2 3 ...}
{4,3,1,1,2,2,3,3}→L2 [ENTER]
                          {4 3 1 ...}
QuadReg L1,L2 [ENTER]            Done
ShowStat [ENTER]
```




```
[ENTER]
Regeq(x)→y1(x) [ENTER]           Done
NewPlot 1,1,L1,L2 [ENTER]        Done
```

[◆] [GRAPH]



QuartReg MATH/Statistics/Regressions menu

QuartReg *list1*, *list2*[, [*list3*] [, *list4*, *list5*]]

Calculates the quartic polynomial regression and updates the system statistics variables.

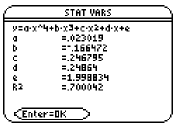
All the lists must have equal dimensions except for *list5*.

list1 represents xlist.
list2 represents ylist.
list3 represents frequency.
list4 represents category codes.
list5 represents category include list.

Note: *list1* through *list4* must be a variable name or c1–c99 (columns in the last data variable shown in the Data/Matrix Editor). *list5* does not have to be a variable name and cannot be c1–c99.

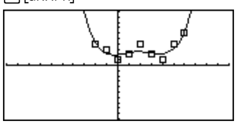
In function graphing mode:

```
{⁻2,⁻1,0,1,2,3,4,5,6}→L1 [ENTER]
                         {⁻2 ⁻1 0 ...}
{4,3,1,2,4,2,1,4,6}→L2 [ENTER]
                          {4 3 1 ...}
QuartReg L1,L2 [ENTER]           Done
ShowStat [ENTER]
```




```
[ENTER]
Regeq(x)→y1(x) [ENTER]           Done
NewPlot 1,1,L1,L2 [ENTER]        Done
```

[◆] [GRAPH]



R▸Pθ() MATH/Angle menu

R▸Pθ (*xExpression*, *yExpression***)** ⇒ *expression*
R▸Pθ (*xList*, *yList***)** ⇒ *list*
R▸Pθ (*xMatrix*, *yMatrix***)** ⇒ *matrix*

Returns the equivalent θ-coordinate of the (*x,y*) pair arguments.

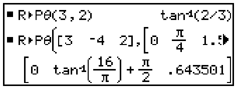
Note: The result is returned as either a degree or radian angle, according to the current angle mode.

In Degree angle mode:

R▸Pθ(x,y) [ENTER]

In Radian angle mode:

R▸Pθ(3,2) [ENTER]
R▸Pθ([3,-4,2],[0,π/4,1.5]) [ENTER]



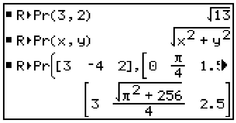
R▸Pr() MATH/Angle menu

R▸Pr (*xExpression*, *yExpression***)** ⇒ *expression*
R▸Pr (*xList*, *yList***)** ⇒ *list*
R▸Pr (*xMatrix*, *yMatrix***)** ⇒ *matrix*

Returns the equivalent r-coordinate of the (*x,y*) pair arguments.

In Radian angle mode:

R▸Pr(3,2) [ENTER]
R▸Pr(x,y) [ENTER]
R▸Pr([3,-4,2],[0,π/4,1.5]) [ENTER]



rand() MATH/Probability menu

rand([*n***])** ⇒ *expression*

n is an integer ≠ zero.

With no parameter, returns the next random number between 0 and 1 in the sequence. When an argument is positive, returns a random integer in the interval [1, *n*].
When an argument is negative, returns a random integer in the interval [⁻ *n*, ⁻ 1].

```
RandSeed 1147 [ENTER]          Done
```
(Sets the random-number seed.)

```
rand() [ENTER]                 .158...
rand(6) [ENTER]                5
rand(⁻100) [ENTER]             ⁻49
```

randMat() MATH/Probability menu

randMat(*numRows*, *numColumns***)** ⇒ *matrix*

Returns a matrix of integers between -9 and 9 of the specified dimension.

Both arguments must simplify to integers.

```
RandSeed 1147 [ENTER]          Done
randMat(3,3) [ENTER]     [ 8  ⁻3   6]
                         [⁻2   3  ⁻6]
                         [ 0   4  ⁻6]
```

Note: The values in this matrix will change each time you press [ENTER].

randNorm() MATH/Probability menu

randNorm(*mean*, *sd***)** ⇒ *expression*

Returns a decimal number from the specific normal distribution. It could be any real number but will be heavily concentrated in the interval [*mean*-3∗ *sd*, *mean*+3∗ *sd*].

```
RandSeed 1147 [ENTER]          Done
randNorm(0,1) [ENTER]          .492...
randNorm(3,4.5) [ENTER]        −3.543...
```

randPoly() MATH/Probability menu

randPoly(*var***,** *order***)** ⇒ *expression*

Returns a polynomial in *var* of the specified order. The coefficients are random integers in the range ⁻9 through 9. The leading coefficient will not be zero.

order must be 0–99.

```
RandSeed 1147 [ENTER]                Done
randPoly(x,5) [ENTER]
          ⁻2·x^5+3·x^4−6·x^3+4·x−6
```

RandSeed MATH/Probability menu

RandSeed *number*

If *number* = 0, sets the seeds to the factory defaults for the random-number generator. If *number* ≠ 0, it is used to generate two seeds, which are stored in system variables seed1 and seed2.

```
RandSeed 1147 [ENTER]                Done
rand() [ENTER]                       .158...
```

RclGDB CATALOG

RclGDB *GDBvar*

Restores all the settings stored in the Graph database variable GDBvar.

For a listing of the settings, see **StoGDB**.

Note: It is necessary to have something saved in GDBvar before you can restore it.

```
RclGDB GDBvar [ENTER]                Done
```

RclPic CATALOG

RclPic *picVar* [**,** *row***,** *column*]

Displays the Graph screen and adds the picture stored in *picVar* at the upper left-hand corner pixel coordinates (*row*, *column*) using OR logic.

picVar must be a picture data type.

Default coordinates are (0, 0).

real() MATH/Complex menu

real(*expression1***)** ⇒ *expression*

Returns the real part of the argument.

Note: All undefined variables are treated as real variables. See also **imag()**.

```
real(2+3i) [ENTER]                   2
real(z) [ENTER]                      z
real(x+iy) [ENTER]                   x
```

---

real(*list1***)** ⇒ *list*

Returns the real parts of all elements.

```
real({a+i*b,3,i}) [ENTER]     {a  3  0}
```

---

real(*matrix1***)** ⇒ *matrix*

Returns the real parts of all elements.

```
real([a+i*b,3;c,i]) [ENTER]    [a  3]
                               [c  0]
```

▸Rect — MATH/Matrix/Vector ops menu

vector **▸Rect**

Displays *vector* in rectangular form [x, y, z]. The vector must be of dimension 2 or 3 and can be a row or a column.

Note: **▸Rect** is a display-format instruction, not a conversion function. You can use it only at the end of an entry line, and it does not update ans.

Note: See also **▸Polar**.

```
[3,∠π/4,∠π/6]▸Rect [ENTER]
```
$$\left[\frac{3\cdot\sqrt{2}}{4}\quad \frac{3\cdot\sqrt{2}}{4}\quad \frac{3\cdot\sqrt{3}}{2}\right]$$

```
[a,∠b,∠c] [ENTER] [a·cos(b)·sin(c)
   a·sin(b)·sin(c)   a·cos(c)]
```

complexValue **▸Rect**

Displays *complexValue* in rectangular form a+b*i*. The *complexValue* can have any complex form. However, an $re^{i\theta}$ entry causes an error in Degree angle mode.

Note: You must use parentheses for an (r∠θ) polar entry.

In Radian angle mode:

4*e*^(π/3)▸Rect [ENTER] $4\cdot e^{\frac{\pi}{3}}$

(4∠π/3)▸Rect [ENTER] $2+2\cdot\sqrt{3}\cdot i$

In Degree angle mode:

(4∠60)▸Rect [ENTER] $2+2\cdot\sqrt{3}\cdot i$

Note: To type **▸Rect** from the keyboard, press [2nd] [▸] for the ▸ operator. To type ∠, press [2nd] [∠].

ref() — MATH/Matrix menu

ref(*matrix1*[, *tol*]**)** ⇒ *matrix*

Returns the row echelon form of *matrix1*.

Optionally, any matrix element is treated as zero if its absolute value is less than *tol*. This tolerance is used only if the matrix has floating-point entries and does not contain any symbolic variables that have not been assigned a value. Otherwise, *tol* is ignored.

- If you use [◆] [ENTER] or set the mode to Exact/Approx=APPROXIMATE, computations are done using floating-point arithmetic.
- If *tol* is omitted or not used, the default tolerance is calculated as:

  5E⁻14 ∗ **max(dim(***matrix1***))** ∗ **rowNorm(***matrix1***)**

Note: See also **rref()**.

```
ref([⁻2,⁻2,0,⁻6;1,⁻1,9,⁻9;⁻5,
2,4,⁻4]) [ENTER]
```
$$\begin{bmatrix}1 & -2/5 & -4/5 & 4/5\\ 0 & 1 & 4/7 & 11/7\\ 0 & 0 & 1 & -62/71\end{bmatrix}$$

[a,b,c;e,f,g]→m1 [ENTER] $\begin{bmatrix}a & b & c\\ e & f & g\end{bmatrix}$

ref(m1) [ENTER] $\begin{bmatrix}1 & \frac{f}{e} & \frac{g}{e}\\ 0 & 1 & \frac{a\cdot g-c\cdot e}{a\cdot f-b\cdot e}\end{bmatrix}$

remain() — MATH/Number menu

remain(*expression1*, *expression2***)** ⇒ *expression*
remain(*list1*, *list2***)** ⇒ *list*
remain(*matrix1*, *matrix2***)** ⇒ *matrix*

Returns the remainder of the first argument with respect to the second argument as defined by the identities:

remain(x,0) ≡ x
remain(x,y) ≡ x− y∗iPart(x/y)

```
remain(7,0) [ENTER]            7
remain(7,3) [ENTER]            1
remain(⁻7,3) [ENTER]          ⁻1
remain(7,⁻3) [ENTER]           1
remain(⁻7,⁻3) [ENTER]         ⁻1
remain({12,⁻14,16},{9,7,⁻5})
[ENTER]
                        {3  0  1}
```

As a consequence, note that **remain(**− x,y**)** ≡ − **remain(**x,y**)**. The result is either zero or it has the same sign as the first argument.

Note: See also **mod()**.

```
remain([9,⁻7;6,4],[4,3;4,⁻3]) ENTER
```

$$\begin{bmatrix} 1 & ^-1 \\ 2 & 1 \end{bmatrix}$$

Rename CATALOG

Rename *oldVarName*, *newVarName*

Renames the variable *oldVarName* as *newVarName*.

```
{1,2,3,4}→L1 ENTER          {1,2,3,4}
Rename L1, list1 ENTER           Done
list1 ENTER                 {1,2,3,4}
```

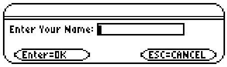
Request CATALOG

Request *promptString*, *var*

If **Request** is inside a **Dialog**...**EndDlog** construct, it creates an input box for the user to type in data. If it is a stand-alone instruction, it creates a dialog box for this input. In either case, if *var* contains a string, it is displayed and highlighted in the input box as a default choice. *promptString* must be ≤ 20 characters.

This instruction can be stand-alone or part of a dialog construct.

```
Request "Enter Your Name",str1
ENTER
```



Return CATALOG

Return [*expression***]**

Returns *expression* as the result of the function. Use within a **Func**...**EndFunc** block, or **Prgm**...**EndPrgm** block.

Note: Use **Return** without an argument to exit a program.

Note: Enter the text as one long line on the Home screen (without line breaks).

```
Define factoral(nn)=Func
 :local answer,count:1→answer
 :For count,1,nn
 :answer∗count→answer:EndFor
 :Return answer:EndFunc ENTER Done

factoral(3) ENTER                   6
```

right() MATH/List menu

right(*list1***[,** *num***])** ⇒ *list*

Returns the rightmost *num* elements contained in *list1*.

If you omit *num*, returns all of *list1*.

```
right({1,3,⁻2,4},3) ENTER
                         {3 ⁻2 4}
```

---

right(*sourceString***[,** *num***])** ⇒ *string*

Returns the rightmost *num* characters contained in character string *sourceString*.

If you omit *num*, returns all of *sourceString*.

```
right("Hello",2) ENTER           "lo"
```

---

right(*comparison***)** ⇒ *expression*

Returns the right side of an equation or inequality.

```
right(x<3) ENTER                    3
```

rotate() MATH/Base menu

rotate(*integer1*[**,***#ofRotations*]**)** ⇒ *integer*

Rotates the bits in a binary integer. You can enter *integer1* in any number base; it is converted automatically to a signed, 32-bit binary form. If the magnitude of *integer1* is too large for this form, a symmetric modulo operation brings it within the range.

If *#of Rotations* is positive, the rotation is to the left. If *#of Rotations* is negative, the rotation is to the right. The default is ⁻1 (rotate right one bit).

For example, in a right rotation:

Each bit rotates right.

0b00000000000000111101011000011010 1

Rightmost bit rotates to leftmost.

produces:

0b10000000000000111101011000011010

The result is displayed according to the Base mode.

In Bin base mode:

```
rotate(0b1111010110000110101) [ENTER]
0b10000000000000111101011000011010
rotate(256,1) [ENTER]  0b1000000000
```

In Hex base mode:

```
rotate(0h78E) [ENTER]          0h3C7
rotate(0h78E,⁻2) [ENTER]  0h800001E3
rotate(0h78E,2) [ENTER]        0h1E38
```

Important: To enter a binary or hexadecimal number, always use the 0b or 0h prefix (zero, not the letter O).

---

rotate(*list1*[**,***#ofRotations*]**)** ⇒ *list*

Returns a copy of *list1* rotated right or left by *#of Rotations* elements. Does not alter *list1*.

If *#of Rotations* is positive, the rotation is to the left. If *#of Rotations* is negative, the rotation is to the right. The default is ⁻1 (rotate right one element).

In Dec base mode:

```
rotate({1,2,3,4}) [ENTER]
                        {4 1 2 3}
rotate({1,2,3,4},⁻2) [ENTER]
                        {3 4 1 2}
rotate({1,2,3,4},1) [ENTER]
                        {2 3 4 1}
```

---

rotate(*string1*[**,***#ofRotations*]**)** ⇒ *string*

Returns a copy of *string1* rotated right or left by *#of Rotations* characters. Does not alter *string1*.

If *#of Rotations* is positive, the rotation is to the left. If *#of Rotations* is negative, the rotation is to the right. The default is ⁻1 (rotate right one character).

```
rotate("abcd") [ENTER]        "dabc"
rotate("abcd",⁻2) [ENTER]     "cdab"
rotate("abcd",1) [ENTER]      "bcda"
```

round() MATH/Number menu

round(*expression1*[**,** *digits*]**)** ⇒ *expression*

Returns the argument rounded to the specified number of digits after the decimal point.

digits must be an integer in the range 0–12. If *digits* is not included, returns the argument rounded to 12 significant digits.

Note: Display digits mode may affect how this is displayed.

```
round(1.234567,3) [ENTER]      1.235
```

---

round(*list1*[**,** *digits*]**)** ⇒ *list*

Returns a list of the elements rounded to the specified number of digits.

```
round({π,√(2),ln(2)},4) [ENTER]
                {3.1416 1.4142 .6931}
```

round(_matrix1_[**,** _digits_]**)** ⇒ _matrix_

Returns a matrix of the elements rounded to the specified number of digits.

```
round([ln(5),ln(3);π,e^(1)],1) [ENTER]
                    [1.6  1.1]
                    [3.1  2.7]
```

rowAdd() MATH/Matrix/Row ops menu

rowAdd(_matrix1_**,** _rIndex1_**,** _rIndex2_**)** ⇒ _matrix_

Returns a copy of _matrix1_ with row _rIndex2_ replaced by the sum of rows _rIndex1_ and _rIndex2_.

```
rowAdd([3,4;⁻3,⁻2],1,2) [ENTER]
                    [3  4]
                    [0  2]
rowAdd([a,b;c,d],1,2) [ENTER]
                    [a    b  ]
                    [a+c  b+d]
```

rowDim() MATH/Matrix/Dimensions menu

rowDim(_matrix_**)** ⇒ _expression_

Returns the number of rows in _matrix_.

Note: See also **colDim()**.

```
[1,2;3,4;5,6]→M1 [ENTER]
                    [1  2]
                    [3  4]
                    [5  6]
rowdim(M1) [ENTER]          3
```

rowNorm() MATH/Matrix/Norms menu

rowNorm(_matrix_**)** ⇒ _expression_

Returns the maximum of the sums of the absolute values of the elements in the rows in _matrix_.

Note: All matrix elements must simplify to numbers. See also **colNorm()**.

```
rowNorm([-5,6,-7;3,4,9;9,-9,-7])
[ENTER]                     25
```

rowSwap() MATH/Matrix/Row ops menu

rowSwap(_matrix1_**,** _rIndex1_**,** _rIndex2_**)** ⇒ _matrix_

Returns _matrix1_ with rows _rIndex1_ and _rIndex2_ exchanged.

```
[1,2;3,4;5,6]→Mat [ENTER]
                    [1  2]
                    [3  4]
                    [5  6]
rowSwap(Mat,1,3) [ENTER]
                    [5  6]
                    [3  4]
                    [1  2]
```

RplcPic CATALOG

RplcPic _picVar_[**,** _row_][**,** _column_]

Clears the Graph screen and places picture _picVar_ at pixel coordinates (_row_, _column_). If you do not want to clear the screen, use **RclPic**.

picVar must be a picture data type variable. _row_ and _column_, if included, specify the pixel coordinates of the upper left corner of the picture. Default coordinates are (0, 0).

Note: For less than full-screen pictures, only the area affected by the new picture is cleared.

rref() MATH/Matrix menu

rref(*matrix1*[, *tol*]**)** ⇒ *matrix*

Returns the reduced row echelon form of *matrix1*.

```
rref([⁻2,⁻2,0,⁻6;1,⁻1,9,⁻9;
⁻5,2,4,⁻4]) [ENTER]
```

$$\begin{bmatrix} 1 & 0 & 0 & 66/71 \\ 0 & 1 & 0 & \frac{147}{71} \\ 0 & 0 & 1 & ^{-}62/71 \end{bmatrix}$$

Optionally, any matrix element is treated as zero if its absolute value is less than *tol*. This tolerance is used only if the matrix has floating-point entries and does not contain any symbolic variables that have not been assigned a value. Otherwise, *tol* is ignored.

- If you use [◆] [ENTER] or set the mode to Exact/Approx=APPROXIMATE, computations are done using floating-point arithmetic.
- If *tol* is omitted or not used, the default tolerance is calculated as:

  5E⁻14 ∗ **max(dim(***matrix1***))** ∗ **rowNorm(***matrix1***)**

Note: See also **ref()**.

```
rref([a,b,x;c,d,y]) [ENTER]
```

$$\begin{bmatrix} 1 & 0 & \frac{d\cdot x-b\cdot y}{a\cdot d-b\cdot c} \\ 0 & 1 & \frac{^{-}(c\cdot x-a\cdot y)}{a\cdot d-b\cdot c} \end{bmatrix}$$

sec() MATH/Trig menu

sec(*expression1***)** ⇒ *expression*
sec(*list1***)** ⇒ *list*

Returns the secant of *expression1* or returns a list containing the secants of all elements in *list1*.

Note: The argument is interpreted as either a degree or radian angle, according to the current angle mode.

In Degree angle mode:

sec(45) [ENTER] $\sqrt{(2)}$

sec({1,2.3,4}) [ENTER]

$\left\{\frac{1}{\cos(1)} \quad 1.000\ldots \quad \frac{1}{\cos(4)}\right\}$

sec⁻¹() MATH/Trig menu

sec⁻¹(*expression1***)** ⇒ *expression*
sec⁻¹(*list1***)** ⇒ *list*

Returns the angle whose secant is *expression1* or returns a list containing the inverse secants of each element of *list1*.

Note: The result is interpreted as either a degree or radian angle, according to the current angle mode.

In Degree angle mode:

$sec^{-1}(1)$ [ENTER] 0

In Radian angle mode:

$sec^{-1}(\{1,2,5\})$ [ENTER]

$\left\{0 \quad \frac{\pi}{3} \quad \cos^{-1}(1/5)\right\}$

sech() MATH/Hyperbolic menu

sech(*expression1***)** ⇒ *expression*
sech(*list1***)** ⇒ *list*

Returns the hyperbolic secant of *expression1* or returns a list containing the hyperbolic secants of the *list1* elements.

sech(3) [ENTER] $\frac{1}{\cosh(3)}$

sech({1,2.3,4}) [ENTER]

$\left\{\frac{1}{\cosh(1)} \quad .198\ldots \quad \frac{1}{\cosh(4)}\right\}$

sech^{-1}() MATH/Hyperbolic menu

sech⁻¹(*expression1***)** ⇒ *expression*
sech⁻¹ (*list1***)** ⇒ *list*

Returns the inverse hyperbolic secant of *expression1* or returns a list containing the inverse hyperbolic secants of each element of *list1*.

```
In Radian angle and Rectangular
complex mode:

sech⁻¹(1) [ENTER]                    0

sech⁻¹({1,-2,2.1}) [ENTER]
```

$0 \quad (\frac{2 \cdot \pi}{3}) \cdot i \quad 1.074... \cdot i$

Send CATALOG

Send *list*

CBL 2™/CBL™ (Calculator-Based Laboratory™) or CBR™ (Calculator-Based Ranger™) instruction. Sends *list* to the link port.

Program segment:

```
   ⋮
:Send {1,0}
:Send {1,2,1}
   ⋮
```

SendCalc CATALOG

SendCalc *var*

Sends variable *var* to the link port, where another unit linked to that port can receive the variable value. The receiving unit must be on the Home screen or must execute **GetCalc** from a program.

If you send from a TI-89, TI-92 Plus, or Voyage™ 200 to a TI-92, an error occurs if the TI-92 executes **GetCalc** from a program. In this case, the sending unit must use **SendChat** instead.

Program segment:

```
   ⋮
:a+b→x
:SendCalc x
   ⋮
```

SendCalc *var[,port]*

Sends contents of *var* from a TI-89 Titanium to another TI-89 Titanium.

If the port is not specified, or *port = 0* is specified, the TI-89 Titanium sends data using the USB port if connected, if not, it will send using the I/O port.

If *port = 1*, the TI-89 Titanium sends data using the USB port only.

If *port = 2*, the TI-89 Titanium sends data using the I/O port only.

SendChat CATALOG

SendChat *var*

A general alternative to **SendCalc**, this is useful if the receiving unit is a TI-92 (or for a generic "chat" program that allows either a TI-92, Voyage™ 200, or TI-92 Plus to be used). Refer to **SendCalc** for more information.

SendChat sends a variable only if that variable is compatible with the TI-92, which is typically true in "chat" programs. However, **SendChat** will not send an archived variable, a TI-89 graph data base, etc.

Program segment:

```
   ⋮
:a+b→x
:SendChat x
   ⋮
```

seq() MATH/List menu

seq(*expression*, *var*, *low*, *high*[, *step*]**)** ⇒ *list*

Increments *var* from *low* through *high* by an increment of *step*, evaluates *expression*, and returns the results as a list. The original contents of *var* are still there after **seq()** is completed.

var cannot be a system variable.

The default value for *step* = 1.

```
seq(n^2,n,1,6) [ENTER]
                 {1 4 9 16 25 36}
seq(1/n,n,1,10,2) [ENTER]
              {1 1/3 1/5 1/7 1/9}
sum(seq(1/n^2,n,1,10,1)) [ENTER]
                           196...
                           ------
                           127...
```

or press [◆] [ENTER] to get: 1.549...

setDate() CATALOG

setDate(*year,month,day***)** ⇒ *listold*

Sets the clock to the date given in the argument and returns a list. (**Note:** The *year* must fall in the range 1997 - 2132.) The returned list is in {*yearold,monthold,dayold*} format. The returned date is the previous clock value.

Enter the year as a four-digit integer. The month and day can be either one- or two-digit integers.

```
setDate(2001,10,31) [ENTER]
                     {2001  11  1}
```

setDtFmt() CATALOG

setDtFmt(*integer***)** ⇒ *integerold*

Sets the date format for the desktop according to the argument and returns the previous date format value.

Integer values:

```
1 = MM/DD/YY   5 = YY.MM.DD
2 = DD/MM/YY   6 = MM-DD-YY
3 = MM.DD.YY   7 = DD-MM-YY
4 = DD.MM.YY   8 = YY-MM-DD
```

setFold() CATALOG

setFold(*newfolderName***)** ⇒ *oldfolderString*

Returns the name of the current folder as a string and sets *newfolderName* as the current folder.

The folder *newfolderName* must exist.

```
newFold chris [ENTER]            Done
setFold(main) [ENTER]         "chris"
setFold(chris)→oldfoldr [ENTER]
                               "main"
1→a [ENTER]                         1
setFold(#oldfoldr) [ENTER]    "chris"
a [ENTER]                           a
chris\a [ENTER]                     1
```

setGraph() CATALOG

setGraph(*modeNameString*, *settingString***)** ⇒ *string*

Sets the Graph mode *modeNameString* to *settingString*, and returns the previous setting of the mode. Storing the previous setting lets you restore it later.

modeNameString is a character string that specifies which mode you want to set. It must be one of the mode names from the table below.

settingString is a character string that specifies the new setting for the mode. It must be one of the settings listed below for the specific mode you are setting.

```
setGraph("Graph Order","Seq")
[ENTER]                         "SEQ"
setGraph("Coordinates","Off")
[ENTER]                        "RECT"
```

Note: Capitalization and blank spaces are optional when entering mode names.

Mode Name	Settings	
"Coordinates"	"Rect", "Polar", "Off"	
"Graph Order"	"Seq", "Simul" [1]	
"Grid"	"Off", "On" [2]	
"Axes"	"Off", "On" "Off", "Axes", "Box"	(not 3D graph mode) (3D graph mode)
"Leading Cursor"	"Off", "On" [2]	
"Labels"	"Off", "On"	
"Style"	"Wire Frame", "Hidden Surface", "Contour Levels", "Wire and Contour", "Implicit Plot" [3]	
"Seq Axes"	"Time", "Web", "U1-vs-U2" [4]	
"DE Axes"	"Time", "t-vs-y' ", "y-vs-y' ", "y1-vs-y2", "y1-vs-y2' ", "y1'-vs-y2' " [5] **Tip:** To type a prime symbol ('), press [2nd] [′].	
"Solution Method"	"RK", "Euler" [5]	
"Fields"	"SlpFld", "DirFld", "FldOff" [5]	

[1]Not available in Sequence, 3D, or Diff Equations graph mode.
[2]Not available in 3D graph mode.
[3]Applies only to 3D graph mode.
[4]Applies only to Sequence graph mode.
[5]Applies only to Diff Equations graph mode.

setMode() CATALOG

setMode(*modeNameString*, *settingString***)** ⇒ *string*
setMode(*list***)** ⇒ *stringList*

Sets mode *modeNameString* to the new setting *settingString*, and returns the current setting of that mode.

modeNameString is a character string that specifies which mode you want to set. It must be one of the mode names from the table below.

settingString is a character string that specifies the new setting for the mode. It must be one of the settings listed below for the specific mode you are setting.

list contains pairs of keyword strings and will set them all at once. This is recommended for multiple-mode changes. The example shown may not work if each of the pairs is entered with a separate **setMode()** in the order shown.

Use **setMode(***var***)** to restore settings saved with **getMode("ALL")→** *var*.

Note: To set or return information about the Unit System mode, use **setUnits()** or **getUnits()** instead of **setMode()** or **getMode()**.

```
setMode("Angle","Degree")
[ENTER]                         "RADIAN"

sin(45) [ENTER]                    √2/2

setMode("Angle","Radian")
[ENTER]                         "DEGREE"

sin(π/4) [ENTER]                   √2/2

setMode("Display Digits",
"Fix 2") [ENTER]                 "FLOAT"

π [◆] [ENTER]                       3.14

setMode ("Display Digits",
"Float") [ENTER]                 "FIX 2"

π [◆] [ENTER]                     3.141...

setMode ({"Split Screen",
"Left-Right","Split 1 App",
"Graph","Split 2
App","Table"})
[ENTER]
          {"Split 2 App" "Graph"
            "Split 1 App" "Home"
          "Split Screen" "FULL"}
```

Note: Capitalization and blank spaces are optional when entering mode names. Also, the results in these examples may be different on your unit.

Mode Name	Settings
"Graph"	"Function", "Parametric", "Polar", "Sequence", "3D", "Diff Equations"
"Display Digits"	"Fix 0", "Fix 1", ..., "Fix 12", "Float", "Float 1", ..., "Float 12"
"Angle"	"Radian", "Degree"
"Exponential Format"	"Normal", "Scientific", "Engineering"
"Complex Format"	"Real", "Rectangular", "Polar"
"Vector Format"	"Rectangular", "Cylindrical", "Spherical"
"Pretty Print"	"Off", "On"
"Split Screen"	"Full", "Top-Bottom", "Left-Right"
"Split 1 App"	"Home", "Y= Editor", "Window Editor", "Graph", "Table", "Data/Matrix Editor", "Program Editor", "Text Editor", "Numeric Solver", "*Flash App*"
"Split 2 App"	"Home", "Y= Editor", "Window Editor", "Graph", "Table", "Data/Matrix Editor", "Program Editor", "Text Editor", "Numeric Solver", "*Flash App*"
"Number of Graphs"	"1", "2"
"Graph2"	"Function", "Parametric", "Polar", "Sequence", "3D", "Diff Equations"
"Split Screen Ratio"	"1:1", "1:2", "2:1" (Voyage™ 200 only)
"Exact/Approx"	"Auto", "Exact", "Approximate"
"Base"	"Dec", "Hex", "Bin"
"Language"	"English", "*Alternate Language*"
"Apps Desktop"	"Off", "On"

setTable() CATALOG

setTable(*modeNameString*, *settingString***)** ⇒ *string*

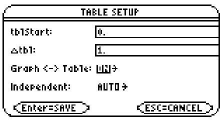
Sets the table parameter *modeNameString* to *settingString*, and returns the previous setting of the parameter. Storing the previous setting lets you restore it later.

modeNameString is a character string that specifies which parameter you want to set. It must be one of the parameters from the table below.

settingString is a character string that specifies the new setting for the parameter. It must be one of the settings listed below for the specific parameter you are setting.

```
setTable("Graph <->
Table","ON")
[ENTER]                         "OFF"

setTable("Independent","AUTO")
[ENTER]                         "ASK"
```

[◆] [TblSet]



Note: Capitalization and blank spaces are optional when entering parameters.

Parameter Name	Settings
"Graph <-> Table"	"Off", "On"
"Independent"	"Auto", "Ask"

setTime() CATALOG

setTime(*hour,minute,second***)** ⇒ *listold*

Sets the clock to the time given in the argument and returns a list. The list is in {*hourold,minuteold,secondold*} format. The returned time is the previous clock value.

Enter the hour in the 24 hour format, in which 13 = 1 p.m.

```
setTime(11,32,50)
                        {10  44  49}
```

setTmFmt() CATALOG

setTmFmt(*integer***)** ⇒ *integerold*

Sets the time format for the desktop according to the argument and returns the previous time format value.

Integer values:

```
12 = 12 hour clock

24 = 24 hour clock
```

setTmZn() CATALOG

setTmZn(*integer***)** ⇒ *integerold*

Sets the time zone according to the argument and returns the previous time zone value.

The time zone is defined by an integer that gives the minutes offset from Greenwich Mean Time (GMT), as established in Greenwich, England. For example, if the time zone is offset from GMT by two hours, the device would return 120 (minutes).

Integers for time zones west of GMT are negative.

Integers for time zones east of GMT are positive.

If Greenwich Mean Time is 14:07:07, it is:

7:07:07 a.m. in Denver, Colorado (Mountain Standard Time)
(–420 minutes from GMT)

15:07:07 p.m. in Brussels, Belgium (Central European Standard Time)
(+60 minutes from GMT)

setUnits() CATALOG

setUnits(*list1***)** ⇒ *list*

Sets the default units to the values specified in *list1*, and returns a list of the previous defaults.

- To specify the built-in SI (metric) or ENG/US system, *list1* uses the form:

  {"SI"} or {"ENG/US"}

- To specify a custom set of default units, *list1* uses the form:

  {"CUSTOM", "*cat1*", "*unit1*"[, "*cat2*", "*unit2*", …]}

  where each *cat* and *unit* pair specifies a category and its default unit. (You can specify built-in units only, not user-defined units.) Any category not specified will use its previous custom unit.

- To return to the previous custom default units, *list1* uses the form:

  {"CUSTOM"}

If you want different defaults depending on the situation, create separate lists and save them to unique list names. To use a set of defaults, specify that list name in **setUnits()**.

You can use **setUnits()** to restore settings previously saved with **setUnits()** → *var* or with **getUnits()** → *var*.

All unit names must begin with an underscore _.

[◆] [_]
[2nd] [_]

You can also select units from a menu by pressing:

[2nd] [UNITS]
[◆] [UNITS]

```
setUnits({"SI"}) [ENTER]
          {"SI"  "Area"  "NONE"
      "Capacitance"  "_F"  ...}

setUnits({"CUSTOM","Length",
"_cm","Mass","_gm"}) [ENTER]
          {"SI"  "Length"  "_m"
            "Mass"  "_kg"  ...}
```

Note: Your screen may display different units.

Shade CATALOG

Shade *expr1*, *expr2*, [*xlow*], [*xhigh*], [*pattern*], [*patRes*]

Displays the Graph screen, graphs *expr1* and *expr2*, and shades areas in which *expr1* is less than *expr2*. (*expr1* and *expr2* must be expressions that use x as the independent variable.)

xlow and *xhigh*, if included, specify left and right boundaries for the shading. Valid inputs are between xmin and xmax. Defaults are xmin and xmax.

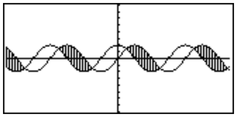
pattern specifies one of four shading patterns:
1 = vertical (default)
2 = horizontal
3 = negative-slope 45°
4 = positive-slope 45°

patRes specifies the resolution of the shading patterns:
1= solid shading
2= 1 pixel spacing (default)
3= 2 pixels spacing
⋮
10= 9 pixels spacing

Note: Interactive shading is available on the Graph screen through the **Shade** instruction. Automatic shading of a specific function is available through the **Style** instruction. **Shade** is not valid in 3D graphing mode.

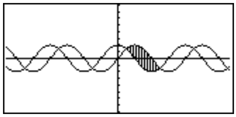
In the ZoomTrig viewing window:

```
Shade cos(x),sin(x) [ENTER]
```



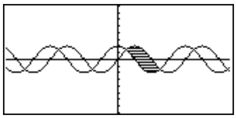
[HOME]
[◆] [CALC HOME]

```
ClrDraw [ENTER]                Done
Shade cos(x),sin(x),0,5 [ENTER]
```



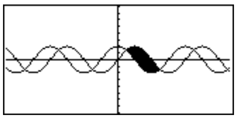
[HOME]
[◆] [CALC HOME]

```
ClrDraw [ENTER]                Done
Shade cos(x),sin(x),0,5,2 [ENTER]
```



[HOME]
[◆] [CALC HOME]

```
ClrDraw [ENTER]                Done
Shade cos(x),sin(x),0,5,2,1
[ENTER]
```



shift() CATALOG

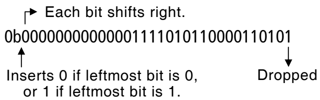
shift(*integer1*[**,***#ofShifts*]**)** ⇒ *integer*

Shifts the bits in a binary integer. You can enter *integer1* in any number base; it is converted automatically to a signed, 32-bit binary form. If the magnitude of *integer1* is too large for this form, a symmetric modulo operation brings it within the range.

If *#ofShifts* is positive, the shift is to the left. If *#ofShifts* is negative, the shift is to the right. The default is ⁻1 (shift right one bit).

In a right shift, the rightmost bit is dropped and 0 or 1 is inserted to match the leftmost bit. In a left shift, the leftmost bit is dropped and 0 is inserted as the rightmost bit.

For example, in a right shift:



produces:

0b00000000000000111101011000011010

The result is displayed according to the Base mode. Leading zeros are not shown.

In Bin base mode:

```
shift(0b1111010110000110101) [ENTER]
                0b111101011000011010

shift(256,1) [ENTER]
                        0b1000000000
```

In Hex base mode:

```
shift(0h78E) [ENTER]          0h3C7

shift(0h78E,⁻2) [ENTER]       0h1E3

shift(0h78E,2) [ENTER]       0h1E38
```

Important: To enter a binary or hexadecimal number, always use the 0b or 0h prefix (zero, not the letter O).

---

shift(*list1* [**,***#ofShifts*]**)** ⇒ *list*

Returns a copy of *list1* shifted right or left by *#ofShifts* elements. Does not alter *list1*.

If *#ofShifts* is positive, the shift is to the left. If *#ofShifts* is negative, the shift is to the right. The default is ⁻1 (shift right one element).

Elements introduced at the beginning or end of *list* by the shift are set to the symbol "undef".

In Dec base mode:

```
shift({1,2,3,4}) [ENTER]
                  {undef 1 2 3}

shift({1,2,3,4},⁻2) [ENTER]
               {undef undef 1 2}

shift({1,2,3,4},1) [ENTER]
                  {2 3 4 undef}
```

---

shift(*string1* [**,***#ofShifts*]**)** ⇒ *string*

Returns a copy of *string1* shifted right or left by *#ofShifts* characters. Does not alter *string1*.

If *#ofShifts* is positive, the shift is to the left. If *#ofShifts* is negative, the shift is to the right. The default is ⁻1 (shift right one character).

Characters introduced at the beginning or end of *string* by the shift are set to a space.

```
shift("abcd") [ENTER]        " abc"

shift("abcd",⁻2) [ENTER]     "  ab"

shift("abcd",1) [ENTER]      "bcd "
```

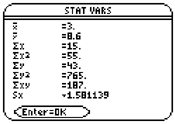
ShowStat CATALOG

ShowStat

Displays a dialog box containing the last computed statistics results if they are still valid. Statistics results are cleared automatically if the data to compute them has changed.

Use this instruction after a statistics calculation, such as **LinReg**.

```
{1,2,3,4,5}→L1 [ENTER]  {1 2 3 4 5}
{0,2,6,10,25}→L2 [ENTER]
                    {0 2 6 10 25}
TwoVar L1,L2 [ENTER]
ShowStat [ENTER]
```



sign() MATH/Number menu

sign(*expression1***)** ⇒ *expression*
sign(*list1***)** ⇒ *list*
sign(*matrix1***)** ⇒ *matrix*

For real and complex *expression1*, returns *expression1*/**abs(***expression1***)** when *expression1*≠ 0.

Returns 1 if *expression1* is positive.
Returns ⁻1 if *expression1* is negative.
sign(0) returns ±1 if the complex format mode is REAL; otherwise, it returns itself.
sign(0) represents the unit circle in the complex domain.

For a list or matrix, returns the signs of all the elements.

```
sign(⁻3.2) [ENTER]                ⁻1.
sign({2,3,4,⁻5}) [ENTER]
                      {1  1  1 ⁻1}
sign(1+abs(x)) [ENTER]             1
```

If complex format mode is REAL:

```
sign([⁻3,0,3]) [ENTER]     [⁻1 ±1 1]
```

simult() MATH/Matrix menu

simult(*coeffMatrix*, *constVector*[, *tol*]**)** ⇒ *matrix*

Returns a column vector that contains the solutions to a system of linear equations.

coeffMatrix must be a square matrix that contains the coefficients of the equations.

constVector must have the same number of rows (same dimension) as *coeffMatrix* and contain the constants.

Optionally, any matrix element is treated as zero if its absolute value is less than *tol*. This tolerance is used only if the matrix has floating-point entries and does not contain any symbolic variables that have not been assigned a value. Otherwise, *tol* is ignored.

- If you use [◆] [ENTER] or set the mode to Exact/Approx=APPROXIMATE, computations are done using floating-point arithmetic.
- If *tol* is omitted or not used, the default tolerance is calculated as:

  5E⁻14 * **max(dim(***coeffMatrix***))** * **rowNorm(***coeffMatrix***)**

Solve for x and y: x + 2y = 1
3x + 4y = ⁻1

```
simult([1,2;3,4],[1;⁻1]) [ENTER]
                              [⁻3]
                              [ 2]
```

The solution is x= ⁻3 and y=2.

Solve: ax + by = 1
cx + dy = 2

```
[a,b;c,d]→matx1 [ENTER]        [a b]
                               [c d]
simult(matx1,[1;2]) [ENTER]
```

$$\begin{bmatrix} \dfrac{-(2\cdot b-d)}{a\cdot d-b\cdot c} \\ \dfrac{2\cdot a-c}{a\cdot d-b\cdot c} \end{bmatrix}$$

simult(*coeffMatrix*, *constMatrix*[, *tol*]**)** ⇒ *matrix*

Solves multiple systems of linear equations, where each system has the same equation coefficients but different constants.

Each column in *constMatrix* must contain the constants for a system of equations. Each column in the resulting matrix contains the solution for the corresponding system.

Solve: x + 2y = 1 ; 3x + 4y = ⁻1 and x + 2y = 2 ; 3x + 4y = ⁻3

```
simult([1,2;3,4],[1,2;⁻1,⁻3]) [ENTER]
```

$$\begin{bmatrix} -3 & -7 \\ 2 & 9/2 \end{bmatrix}$$

For the first system, x=⁻3 and y=2. For the second system, x=⁻7 and y=9/2.

sin()

[2nd] [SIN] **key**   [SIN] **key**

sin(*expression1***)** ⇒ *expression*
sin(*list1***)** ⇒ *list*

sin(*expression1***)** returns the sine of the argument as an expression.

sin(*list1***)** returns a list of the sines of all elements in *list1*.

Note: The argument is interpreted as either a degree or radian angle, according to the current angle mode. You can use ° or ʳ to override the angle mode setting temporarily.

In Degree angle mode:

```
sin((π/4)ʳ) [ENTER]        √2/2
sin(45) [ENTER]            √2/2
sin({0,60,90}) [ENTER]     {0  √3/2  1}
```

In Radian angle mode:

```
sin(π/4) [ENTER]           √2/2
sin(45°) [ENTER]           √2/2
```

sin(*squareMatrix1***)** ⇒ *squareMatrix*

Returns the matrix sine of *squareMatrix1*. This is *not* the same as calculating the sine of each element. For information about the calculation method, refer to **cos()**.

squareMatrix1 must be diagonalizable. The result always contains floating-point numbers.

In Radian angle mode:

```
sin([1,5,3;4,2,1;6,⁻2,1]) [ENTER]
```

$$\begin{bmatrix} .942... & -.045... & -.031... \\ -.045... & .949... & -.020... \\ -.048... & -.005... & .961... \end{bmatrix}$$

sin⁻¹()

[♦] [SIN⁻¹] **key**   [2nd] [SIN⁻¹] **key**

sin⁻¹(*expression1***)** ⇒ *expression*
sin⁻¹(*list1***)** ⇒ *list*

sin⁻¹(*expression1***)** returns the angle whose sine is *expression1* as an expression.

sin⁻¹(*list1***)** returns a list of the inverse sines of each element of *list1*.

Note: The result is returned as either a degree or radian angle, according to the current angle mode setting.

In Degree angle mode:

```
sin⁻¹(1) [ENTER]           90
```

In Radian angle mode:

```
sin⁻¹({0,.2,.5}) [ENTER]
                 {0  .201...  .523...}
```

sin⁻¹(*squareMatrix1***)** ⇒ *squareMatrix*

Returns the matrix inverse sine of *squareMatrix1*. This is *not* the same as calculating the inverse sine of each element. For information about the calculation method, refer to **cos()**.

squareMatrix1 must be diagonalizable. The result always contains floating-point numbers.

In Radian angle mode and Rectangular complex format mode:

```
sin⁻¹([1,5,3;4,2,1;6,⁻2,1])
[ENTER]
```

$$\begin{bmatrix} -.164...-.064...\cdot i & 1.490...-2.105...\cdot i & ... \\ .725...-1.515...\cdot i & .947...-.778...\cdot i & ... \\ 2.083...-2.632...\cdot i & -1.790...+1.271...\cdot i & ... \end{bmatrix}$$

sinh() MATH/Hyperbolic menu

sinh(*expression1***)** ⇒ *expression*
sinh(*list1***)** ⇒ *list*

sinh (*expression1***)** returns the hyperbolic sine of the argument as an expression.

sinh (*list***)** returns a list of the hyperbolic sines of each element of *list1*.

```
sinh(1.2) [ENTER]                1.509…
sinh({0,1.2,3.}) [ENTER]
        {0  1.509…  10.017…}
```

sinh(*squareMatrix1***)** ⇒ *squareMatrix*

Returns the matrix hyperbolic sine of *squareMatrix1*. This is *not* the same as calculating the hyperbolic sine of each element. For information about the calculation method, refer to **cos()**.

squareMatrix1 must be diagonalizable. The result always contains floating-point numbers.

In Radian angle mode:

```
sinh([1,5,3;4,2,1;6,⁻2,1])
[ENTER]
   [360.954  305.708  239.604]
   [352.912  233.495  193.564]
   [298.632  154.599  140.251]
```

$\sinh^{-1}$() MATH/Hyperbolic menu

sinh^{-1} **(***expression1***)** ⇒ *expression*
sinh^{-1} **(***list1***)** ⇒ *list*

sinh^{-1} **(***expression1***)** returns the inverse hyperbolic sine of the argument as an expression.

sinh^{-1} **(***list1***)** returns a list of the inverse hyperbolic sines of each element of *list1*.

```
sinh⁻¹(0) [ENTER]                    0
sinh⁻¹({0,2.1,3}) [ENTER]
        {0  1.487…  sinh⁻¹(3)}
```

sinh^{-1}**(***squareMatrix1***)** ⇒ *squareMatrix*

Returns the matrix inverse hyperbolic sine of *squareMatrix1*. This is *not* the same as calculating the inverse hyperbolic sine of each element. For information about the calculation method, refer to **cos()**.

squareMatrix1 must be diagonalizable. The result always contains floating-point numbers.

In Radian angle mode:

```
sinh⁻¹([1,5,3;4,2,1;6,⁻2,1])
[ENTER]
   [.041…   2.155…   1.158…]
   [1.463…  .926…    .112… ]
   [2.750…  ⁻1.528…  .572… ]
```

SinReg — MATH/Statistics/Regressions menu

SinReg *list1*, *list2* [, [*iterations*] , [*period*] [, *list3*, *list4*]]

Calculates the sinusoidal regression and updates all the system statistics variables.

All the lists must have equal dimensions except for *list4*.

list1 represents xlist.
list2 represents ylist.
list3 represents category codes.
list4 represents category include list.

iterations specifies the maximum number of times (1 through 16) a solution will be attempted. If omitted, 8 is used. Typically, larger values result in better accuracy but longer execution times, and vice versa.

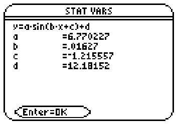
period specifies an estimated period. If omitted, the difference between values in *list1* should be equal and in sequential order. If you specify *period*, the differences between x values can be unequal.

Note: *list1* through *list3* must be a variable name or c1–c99 (columns in the last data variable shown in the Data/Matrix Editor). *list4* does not have to be a variable name and cannot be c1–c99.

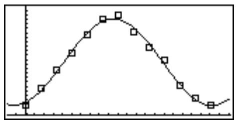
The output of **SinReg** is always in radians, regardless of the angle mode setting.

In function graphing mode:

```
seq(x,x,1,361,30)→L1 [ENTER]
                              {1 31 61 …}
{5.5,8,11,13.5,16.5,19,19.5,17,
14.5,12.5,8.5,6.5,5.5}→L2 [ENTER]
                              {5.5 8 11 …}
SinReg L1,L2 [ENTER]                 Done
ShowStat [ENTER]
```




```
[ENTER]
regeq(x)→y1(x) [ENTER]               Done
NewPlot 1,1,L1,L2 [ENTER]            Done
[◆] [GRAPH]
[F2] 9
```



solve() — MATH/Algebra menu

solve(*equation*, *var***)** ⇒ *Boolean expression*
solve(*inequality*, *var***)** ⇒ *Boolean expression*

Returns candidate real solutions of an equation or an inequality for *var*. The goal is to return candidates for all solutions. However, there might be equations or inequalities for which the number of solutions is infinite.

```
solve(a*x^2+b*x+c=0,x) [ENTER]
```

$$x = \frac{\sqrt{b^2-4\cdot a\cdot c}-b}{2\cdot a}$$
$$\text{or } x = \frac{-(\sqrt{b^2-4\cdot a\cdot c}+b)}{2\cdot a}$$

Solution candidates might not be real finite solutions for some combinations of values for undefined variables.

```
ans(1)| a=1 and b=1 and c=1
[ENTER]
                    Error: Non-real result
```

For the AUTO setting of the Exact/Approx mode, the goal is to produce exact solutions when they are concise, and supplemented by iterative searches with approximate arithmetic when exact solutions are impractical.

```
solve((x-a)e^(x)=-x*(x-a),x)
[ENTER]
                       x = a or x = -.567…
```

Due to default cancellation of the greatest common divisor from the numerator and denominator of ratios, solutions might be solutions only in the limit from one or both sides.

```
(x+1)(x-1)/(x-1)+x-3 [ENTER]
                                    2·x-2
solve(entry(1)=0,x) [ENTER]         x = 1
entry(2)|ans(1) [ENTER]             undef
limit(entry(3),x,1) [ENTER]             0
```

For inequalities of types ≥, ≤, <, or >, explicit solutions are unlikely unless the inequality is linear and contains only *var*.

```
solve(5x-2 ≥ 2x,x) [ENTER]        x ≥ 2/3
```

For the EXACT setting of the Exact/Approx mode, portions that cannot be solved are returned as an implicit equation or inequality.

```
exact(solve((x-a)e^(x)=-x*
  (x-a),x)) [ENTER]
                        e^x + x = 0 or x = a
```

Use the "|" operator to restrict the solution interval and/or other variables that occur in the equation or inequality. When you find a solution in one interval, you can use the inequality operators to exclude that interval from subsequent searches.

In Radian angle mode:

```
solve(tan(x)=1/x,x)|x>0 and x<1
[ENTER]                    x =.860…
```

false is returned when no real solutions are found. true is returned if **solve()** can determine that any finite real value of *var* satisfies the equation or inequality.

```
solve(x=x+1,x) [ENTER]        false
solve(x=x,x) [ENTER]           true
```

Since **solve()** always returns a Boolean result, you can use "and," "or," and "not" to combine results from **solve()** with each other or with other Boolean expressions.

```
2x-1≤1 and solve(x^2≠9,x) [ENTER]
                x ≤ 1 and x ≠ ⁻3
```

Solutions might contain a unique new undefined variable of the form @n*j* with *j* being an integer in the interval 1–255. Such variables designate an arbitrary integer.

In Radian angle mode:

```
solve(sin(x)=0,x) [ENTER]  x =@n1·π
```

In real mode, fractional powers having odd denominators denote only the real branch. Otherwise, multiple branched expressions such as fractional powers, logarithms, and inverse trigonometric functions denote only the principal branch. Consequently, **solve()** produces only solutions corresponding to that one real or principal branch.

```
solve(x^(1/3)=⁻1,x) [ENTER]   x = ⁻1
solve(√(x)=⁻2,x) [ENTER]      false
solve(⁻√(x)=⁻2,x) [ENTER]      x = 4
```

Note: See also **cSolve()**, **cZeros()**, **nSolve()**, and **zeros()**.

---

solve(*equation1* **and** *equation2* [**and** …], **{***varOrGuess1***,** *varOrGuess2* [**,** …]**})** ⇒ *Boolean expression*

Returns candidate real solutions to the simultaneous algebraic equations, where each *varOrGuess* specifies a variable that you want to solve for.

```
solve(y=x^2-2 and
x+2y=⁻1,{x,y}) [ENTER]
                    x=1 and y=⁻1
             or x=⁻3/2 and y=1/4
```

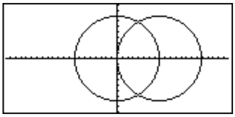
Optionally, you can specify an initial guess for a variable. Each *varOrGuess* must have the form:

variable
– or –
variable = *real or non-real number*

For example, x is valid and so is x=3.

If all of the equations are polynomials and if you do NOT specify any initial guesses, **solve()** uses the lexical Gröbner/Buchberger elimination method to attempt to determine **all** real solutions.

For example, suppose you have a circle of radius r at the origin and another circle of radius r centered where the first circle crosses the positive x-axis. Use **solve()** to find the intersections.



As illustrated by r in the example to the right, simultaneous *polynomial* equations can have extra variables that have no values, but represent given numeric values that could be substituted later.

```
solve(x^2+y^2=r^2 and
(x-r)^2+y^2=r^2,{x,y}) [ENTER]
```

$$x=\frac{r}{2} \text{ and } y=\frac{\sqrt{3}\cdot r}{2}$$
$$\text{or } x=\frac{r}{2} \text{ and } y=\frac{-\sqrt{3}\cdot r}{2}$$

You can also (or instead) include solution variables that do not appear in the equations. For example, you can include z as a solution variable to extend the previous example to two parallel intersecting cylinders of radius r.

```
solve(x^2+y^2=r^2 and
(x-r)^2+y^2=r^2,{x,y,z}) [ENTER]
```

$$x=\frac{r}{2} \text{ and } y=\frac{\sqrt{3}\cdot r}{2} \text{ and } z=@1$$
$$\text{or } x=\frac{r}{2} \text{ and } y=\frac{-\sqrt{3}\cdot r}{2} \text{ and } z=@1$$

The cylinder solutions illustrate how families of solutions might contain arbitrary constants of the form @*k*, where *k* is an integer suffix from 1 through 255. The suffix resets to 1 when you use **ClrHome** or [F1] 8:Clear Home.

For polynomial systems, computation time or memory exhaustion may depend strongly on the order in which you list solution variables. If your initial choice exhausts memory or your patience, try rearranging the variables in the equations and/or *varOrGuess* list.

If you do not include any guesses and if any equation is non-polynomial in any variable but all equations are linear in the solution variables, **solve()** uses Gaussian elimination to attempt to determine all real solutions.

```
solve(x+e^(z)*y=1 and
x-y=sin(z),{x,y}) [ENTER]
```

$$x=\frac{e^{z}\cdot \sin(z)+1}{e^{z}+1} \text{ and } y=\frac{-(\sin(z)-1)}{e^{z}+1}$$

If a system is neither polynomial in all of its variables nor linear in its solution variables, **solve()** determines at most one solution using an approximate iterative method. To do so, the number of solution variables must equal the number of equations, and all other variables in the equations must simplify to numbers.

```
solve(e^(z)*y=1 and
⁻y=sin(z),{y,z}) [ENTER]
                y=.041… and z=3.183…
```

Each solution variable starts at its guessed value if there is one; otherwise, it starts at 0.0.

Use guesses to seek additional solutions one by one. For convergence, a guess may have to be rather close to a solution.

```
solve(e^(z)*y=1 and
-y=sin(z),{y,z=2π}) [ENTER]
          y=.001… and z=6.281…
```

SortA — MATH/List menu

SortA *listName1*[**,** *listName2*] [**,** *listName3*] **...**
SortA *vectorName1*[**,** *vectorName2*] [**,** *vectorName3*] **...**

Sorts the elements of the first argument in ascending order.

If you include additional arguments, sorts the elements of each so that their new positions match the new positions of the elements in the first argument.

All arguments must be names of lists or vectors. All arguments must have equal dimensions.

```
{2,1,4,3}→list1 [ENTER]   {2,1,4,3}
SortA list1 [ENTER]            Done

list1 [ENTER]             {1 2 3 4}
{4,3,2,1}→list2 [ENTER]   {4 3 2 1}
SortA list2,list1 [ENTER]      Done

list2 [ENTER]             {1 2 3 4}
list1 [ENTER]             {4 3 2 1}
```

SortD — MATH/List menu

SortD *listName1*[**,** *listName2*] [**,** *listName3*] **...**
SortD *vectorName1*[**,***vectorName 2*] [**,***vectorName 3*] **...**

Identical to **SortA**, except **SortD** sorts the elements in descending order.

```
{2,1,4,3}→list1 [ENTER]   {2 1 4 3}
{1,2,3,4}→list2 [ENTER]   {1 2 3 4}
SortD list1,list2 [ENTER]      Done
list1 [ENTER]             {4 3 2 1}
list2 [ENTER]             {3 4 1 2}
```

►Sphere — MATH/Matrix/Vector ops menu

vector **►Sphere**

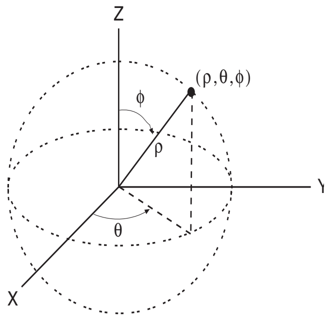
Displays the row or column vector in spherical form [ρ ∠θ ∠φ].

vector must be of dimension 3 and can be either a row or a column vector.

Note: **►Sphere** is a display-format instruction, not a conversion function. You can use it only at the end of an entry line.

```
[1,2,3]►Sphere
◆ [ENTER] [3.741… ∠1.107… ∠.640…]

[2,∠π/4,3]►Sphere
◆ [ENTER]  [3.605… ∠.785… ∠.588…]
```

[ENTER] $[\sqrt{13} \quad \angle\frac{\pi}{4} \quad \angle\cos^{-1}(\frac{3\cdot\sqrt{13}}{13})]$



startTmr() CATALOG

startTmr() ⇒ *integer*

Returns the current value of the clock in its integer representation, giving the *starttime* for a timer. You can enter the *starttime* as an argument in **checkTmr()** to determine how many seconds have elapsed.

You can run multiple timers simultaneously.

Note: See also **checkTmr()** and **timeCnv()**.

```
startTmr() [ENTER]          148083315
checkTmr(148083315)                34

startTmr()→Timer1
:
startTmr()→Timer2
:
checkTmr(Timer1)→Timer1Value
:
checkTmr(Timer2)→Timer2Value
```

stdDev() MATH/Statistics menu

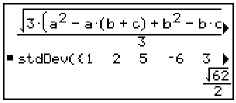
stdDev(*list*[, *freqlist*]**)** ⇒ *expression*

Returns the standard deviation of the elements in *list*.

Each *freqlist* element counts the number of consecutive occurrences of the corresponding element in *list*.

Note: *list* must have at least two elements.

```
stdDev({a,b,c}) [ENTER]
stdDev({1,2,5,⁻6,3,⁻2}) [ENTER]
```



```
stdDev({1.3,2.5,-6.4},{3,2,5})
[ENTER]                       4.33345
```

---

stdDev(*matrix1*[, *freqmatrix*]**)** ⇒ *matrix*

Returns a row vector of the standard deviations of the columns in *matrix1*.

Each *freqmatrix* element counts the number of consecutive occurrences of the corresponding element in *matrix1*.

Note: *matrix1* must have at least two rows.

```
stdDev([1,2,5;-3,0,1;.5,.7,3])
[ENTER]
            [2.179...   1.014...    2]

stdDev([-1.2,5.3;2.5,7.3;6,-4],
[4,2;3,3;1,7]) [ENTER]
                   [2.7005,5.44695]
```

StoGDB CATALOG

StoGDB *GDBvar*

Creates a Graph database (GDB) variable that contains the current:

* Graphing mode
* Y= functions
* Window variables
* Graph format settings
1- or 2-Graph setting (split screen and ratio settings if 2-Graph mode)
Angle mode
Real/complex mode
* Initial conditions if Sequence or Diff Equations mode
* Table flags
* tblStart, Δtbl, tblInput

You can use **RclGDB** *GDBvar* to restore the graph environment.

***Note:** These items are saved for both graphs in 2-Graph mode.

Stop CATALOG

Stop

Used as a program instruction to stop program execution.

Program segment:

```
  :
For i,1,10,1
  If i=5
  Stop
EndFor
  :
```

StoPic CATALOG

StoPic *picVar* [**,** *pxlRow***,** *pxlCol*] [**,** *width***,** *height*]

Displays the graph screen and copies a rectangular area of the display to the variable *picVar*.

pxlRow and *pxlCol*, if included, specify the upper-left corner of the area to copy (defaults are 0, 0).

width and *height*, if included, specify the dimensions, in pixels, of the area. Defaults are the width and height, in pixels, of the current graph screen.

Store

See → (store), page 908.

string() MATH/String menu

string(*expression***)** ⇒ *string*

Simplifies *expression* and returns the result as a character string.

```
string(1.2345) [ENTER]      "1.2345"
string(1+2) [ENTER]              "3"
string(cos(x)+√(3)) [ENTER]
                       "cos(x) + √(3)"
```

Style CATALOG

Style *equanum, stylePropertyString*

Sets the system graphing function *equanum* in the current graph mode to use the graphing property *stylePropertyString*.

equanum must be an integer from 1–99 and the function must already exist.

stylePropertyString must be one of: "Line", "Dot", "Square", "Thick", "Animate", "Path", "Above", or "Below".

Note that in parametric graphing, only the *xt* half of the pair contains the style information.

Valid style names vs. graphing mode:

Function:	all styles
Parametric/Polar:	line, dot, square, thick, animate, path
Sequence:	line, dot, square, thick
3D:	none
Diff Equations:	line, dot, square, thick, animate, path

Note: Capitalization and blank spaces are optional when entering *stylePropertyString* names.

```
Style 1,"thick" [ENTER]         Done
Style 10,"path" [ENTER]         Done
```

Note: In function graphing mode, these examples set the style of y1(x) to "Thick" and y10(x) to "Path".

subMat() CATALOG

subMat(*matrix1*[**,** *startRow*] [**,** *startCol*] [**,** *endRow*] [**,** *endCol*]**)** ⇒ *matrix*

Returns the specified submatrix of *matrix1*.

Defaults: *startRow*=1, *startCol*=1, *endRow*=last row, *endCol*=last column.

```
[1,2,3;4,5,6;7,8,9]→m1 [ENTER]
```
$$\begin{bmatrix}1&2&3\\4&5&6\\7&8&9\end{bmatrix}$$
```
subMat(m1,2,1,3,2) [ENTER]
```
$$\begin{bmatrix}4&5\\7&8\end{bmatrix}$$
```
subMat(m1,2,2) [ENTER]
```
$$\begin{bmatrix}5&6\\8&9\end{bmatrix}$$

Sum (Sigma) See Σ(), page 903.

sum() MATH/List menu

sum(*list*[**,** *start*[**,** *end*]]**)** ⇒ *expression*

Returns the sum of the elements in *list*.

Start and *end* are optional. They specify a range of elements.

```
sum({1,2,3,4,5}) [ENTER]        15
sum({a,2a,3a}) [ENTER]         6·a
sum(seq(n,n,1,10)) [ENTER]      55
sum({1,3,5,7,9},3) [ENTER]      21
```

---

sum(*matrix1*[**,** *start*[**,** *end*]]**)** ⇒ *matrix*

Returns a row vector containing the sums of the elements in the columns in *matrix1*.

Start and *end* are optional. They specify a range of rows.

```
sum([1,2,3;4,5,6]) [ENTER][5  7  9]
sum([1,2,3;4,5,6;7,8,9]) [ENTER]
                       [12 15 18]
sum([1,2,3;4,5,6;7,8,9],2,3)
[ENTER]
                       [11,13,15]
```

switch() CATALOG

switch([*integer1***])** ⇒ *integer*

Returns the number of the active window. Also can set the active window.

Note: Window 1 is left or top; Window 2 is right or bottom.

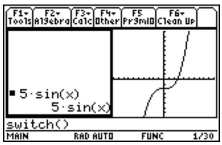
If *integer1* = 0, returns the active window number.

If *integer1* = 1, activates window 1 and returns the previously active window number.

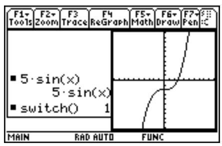
If *integer1* = 2, activates window 2 and returns the previously active window number.

If *integer1* is omitted, switches windows and returns the previously active window number.

integer1 is ignored if the TI-89 Titanium/Voyage™ 200 is not displaying a split screen.




```
switch() [ENTER]
```



T (transpose) MATH/Matrix menu

*matrix1*ᵀ ⇒ *matrix*

Returns the complex conjugate transpose of *matrix1*.

```
[1,2,3;4,5,6;7,8,9]→mat1 [ENTER]
```

$$\begin{bmatrix}1 & 2 & 3\\4 & 5 & 6\\7 & 8 & 9\end{bmatrix}$$

```
mat1ᵀ [ENTER]
```

$$\begin{bmatrix}1 & 4 & 7\\2 & 5 & 8\\3 & 6 & 9\end{bmatrix}$$

```
[a,b;c,d]→mat2 [ENTER]
```

$$\begin{bmatrix}a & b\\c & d\end{bmatrix}$$

```
mat2ᵀ [ENTER]
```

$$\begin{bmatrix}a & c\\b & d\end{bmatrix}$$

```
[1+i,2+i;3+i,4+i]→mat3 [ENTER]
```

$$\begin{bmatrix}1+i & 2+i\\3+i & 4+i\end{bmatrix}$$

```
mat3ᵀ [ENTER]
```

$$\begin{bmatrix}1-i & 3-i\\2-i & 4-i\end{bmatrix}$$

Table CATALOG

Table *expression1*[**,** *expression2*] [**,** *var1*]

Builds a table of the specified expressions or functions.

The expressions in the table can also be graphed. Expressions entered using the **Table** or **Graph** commands are assigned increasing function numbers starting with 1. The expressions can be modified or individually deleted using the edit functions available when the table is displayed by pressing [F4] Header. The currently selected functions in the Y= Editor are temporarily ignored.

To clear the functions created by **Table** or **Graph**, execute the **ClrGraph** command or display the Y= Editor.

If the *var* parameter is omitted, the current graph-mode independent variable is assumed. Some valid variations of this instruction are:

Function graphing:	**Table** *expr*, *x*
Parametric graphing:	**Table** *xExpr*, *yExpr*, *t*
Polar graphing:	**Table** *expr*, θ

Note: The **Table** command is not valid for 3D, sequence, or diff equations graphing. As an alternative, you may want to use **BldData**.

In function graphing mode.

```
Table 1.25x∗cos(x) [ENTER]
```

x	1		
0.	0.		
1.	.67538		
2.	-1.04		
3.	-3.712		
4.	-3.268		

```
Table cos(time),time [ENTER]
```

x	1	2	3
0.	0.	1.	
1.	.67538	.5403	
2.	-1.04	-.4161	
3.	-3.712	-.99	
4.	-3.268	-.6536	

tan() [2nd] [TAN] key [TAN] key

tan(*expression1***)** ⇒ *expression*
tan(*list1***)** ⇒ *list*

tan(*expression1***)** returns the tangent of the argument as an expression.

tan(*list1***)** returns a list of the tangents of all elements in *list1*.

Note: The argument is interpreted as either a degree or radian angle, according to the current angle mode. You can use ° or ʳ to override the angle mode temporarily.

In Degree angle mode:

```
tan((π/4)ʳ) [ENTER]                    1
tan(45) [ENTER]                        1
tan({0,60,90}) [ENTER]
                          {0  √3  undef}
```

In Radian angle mode:

```
tan(π/4) [ENTER]                       1
tan(45°) [ENTER]                       1
tan({π,π/3,-π,π/4}) [ENTER]
                              {0 √3 0 1}
```

---

tan(*squareMatrix1***)** ⇒ *squareMatrix*

Returns the matrix tangent of *squareMatrix1*. This is *not* the same as calculating the tangent of each element. For information about the calculation method, refer to **cos()**.

squareMatrix1 must be diagonalizable. The result always contains floating-point numbers.

In Radian angle mode:

```
tan([1,5,3;4,2,1;6,-2,1]) [ENTER]

[-28.291…   26.088…   11.114…]
[12.117…    -7.835…   -5.481…]
[36.818…   -32.806…  -10.459…]
```

tan⁻¹() [◆] [TAN⁻¹] key [2nd] [TAN⁻¹] key

tan⁻¹(*expression1***)** ⇒ *expression*
tan⁻¹(*list1***)** ⇒ *list*

tan⁻¹(*expression1***)** returns the angle whose tangent is *expression1* as an expression.

tan⁻¹(*list1***)** returns a list of the inverse tangents of each element of *list1*.

Note: The result is returned as either a degree or radian angle, according to the current angle mode setting.

In Degree angle mode:

```
tan⁻¹(1) [ENTER]                      45
```

In Radian angle mode:

```
tan⁻¹({0,.2,.5}) [ENTER]
                      {0  .197…  .463…}
```

---

tan⁻¹(*squareMatrix1***)** ⇒ *squareMatrix*

Returns the matrix inverse tangent of *squareMatrix1*. This is *not* the same as calculating the inverse tangent of each element. For information about the calculation method, refer to **cos()**.

squareMatrix1 must be diagonalizable. The result always contains floating-point numbers.

In Radian angle mode:

```
tan⁻¹([1,5,3;4,2,1;6,-2,1])
[ENTER]

[-.083…   1.266…   .622…]
[.748…    .630…   -.070…]
[1.686…  -1.182…   .455…]
```

tanh() MATH/Hyperbolic menu

tanh(*expression1***)** ⇒ *expression*
tanh(*list1***)** ⇒ *list*

tanh(*expression1***)** returns the hyperbolic tangent of the argument as an expression.

tanh(*list***)** returns a list of the hyperbolic tangents of each element of *list1*.

```
tanh(1.2) [ENTER]                  .833…
tanh({0,1}) [ENTER]        {0  tanh(1)}
```

tanh(*squareMatrix1***)** ⇒ *squareMatrix*

Returns the matrix hyperbolic tangent of *squareMatrix1*. This is *not* the same as calculating the hyperbolic tangent of each element. For information about the calculation method, refer to **cos()**.

squareMatrix1 must be diagonalizable. The result always contains floating-point numbers.

In Radian angle mode:

`tanh([1,5,3;4,2,1;6,⁻2,1])` [ENTER]

$$\begin{bmatrix} ^-.097... & .933... & .425... \\ .488... & .538... & ^-.129... \\ 1.282... & ^-1.034... & .428... \end{bmatrix}$$

tanh⁻¹() MATH/Hyperbolic menu

tanh⁻¹ (*expression1***)** ⇒ *expression*
tanh⁻¹ (*list1***)** ⇒ *list*

tanh⁻¹ (*expression1***)** returns the inverse hyperbolic tangent of the argument as an expression.

tanh⁻¹ (*list1***)** returns a list of the inverse hyperbolic tangents of each element of *list1*.

In rectangular complex format mode:

`tanh⁻¹(0)` [ENTER] 0

`tanh⁻¹({1,2.1,3})` [ENTER]

$\{\infty \quad .518... - 1.570...\cdot i \quad \frac{\ln(2)}{2} - \frac{\pi}{2}\cdot i\}$

tanh⁻¹(*squareMatrix1***)** ⇒ *squareMatrix*

Returns the matrix inverse hyperbolic tangent of *squareMatrix1*. This is *not* the same as calculating the inverse hyperbolic tangent of each element. For information about the calculation method, refer to **cos()**.

squareMatrix1 must be diagonalizable. The result always contains floating-point numbers.

In Radian angle mode and Rectangular complex format mode:

`tanh⁻¹([1,5,3;4,2,1;6,⁻2,1])` [ENTER]

$$\begin{bmatrix} ^-.099...+.164...\cdot i & .267...-1.490...\cdot i & ... \\ ^-.087...-.725...\cdot i & .479...-.947...\cdot i & ... \\ .511...-2.083...\cdot i & ^-.878...+1.790...\cdot i & ... \end{bmatrix}$$

taylor() MATH/Calculus menu

taylor(*expression1***,** *var***,** *order***[,** *point***])** ⇒ *expression*

Returns the requested Taylor polynomial. The polynomial includes non-zero terms of integer degrees from zero through *order* in (*var* minus *point*). **taylor()** returns itself if there is no truncated power series of this order, or if it would require negative or fractional exponents. Use substitution and/or temporary multiplication by a power of
(*var* minus *point*) to determine more general power series.

point defaults to zero and is the expansion point.

`taylor(e^(√(x)),x,2)` [ENTER]
`taylor(e^(t),t,4)|t=√(x)` [ENTER]

■ $\text{taylor}(e^{\sqrt{x}}, x, 2)$ $\quad \text{taylor}(e^{\sqrt{x}}, x, 2, 0)$

■ $\text{taylor}(e^{t}, t, 4) \mid t = \sqrt{x}$ $\quad \frac{x^2}{24} + \frac{x^{3/2}}{6} + \frac{x}{2} + \sqrt{x} + 1$

`taylor(1/(x*(x-1)),x,3)` [ENTER]

■ $\text{taylor}\left(\frac{1}{x\cdot(x-1)}, x, 3\right)$ $\quad \text{taylor}\left(\frac{1}{x\cdot(x-1)}, x, 3, 0\right)$

`expand(taylor(x/(x*(x-1)),`
`x,4)/x,x)` [ENTER]

■ $\text{expand}\left(\frac{\text{taylor}\left(\frac{x}{x\cdot(x-1)}, x ▸\right.}{x}\right)$ $\quad -x^3 - x^2 - x - \frac{1}{x} - 1$

tCollect() MATH\Algebra\Trig menu

tCollect(*expression1***)** ⇒ *expression*

Returns an expression in which products and integer powers of sines and cosines are converted to a linear combination of sines and cosines of multiple angles, angle sums, and angle differences. The transformation converts trigonometric polynomials into a linear combination of their harmonics.

Sometimes **tCollect()** will accomplish your goals when the default trigonometric simplification does not. **tCollect()** tends to reverse transformations done by **tExpand()**. Sometimes applying **tExpand()** to a result from **tCollect()**, or vice versa, in two separate steps simplifies an expression.

```
tCollect((cos(α))^2) [ENTER]
```
$$\frac{\cos(2\cdot\alpha)+1}{2}$$

```
tCollect(sin(α)cos(β)) [ENTER]
```
$$\frac{\sin(\alpha-\beta)+\sin(\alpha+\beta)}{2}$$

tExpand() MATH\Algebra\Trig menu

tExpand(*expression1***)** ⇒ *expression*

Returns an expression in which sines and cosines of integer-multiple angles, angle sums, and angle differences are expanded. Because of the identity $(\sin(x))^2+(\cos(x))^2=1$, there are many possible equivalent results. Consequently, a result might differ from a result shown in other publications.

Sometimes **tExpand()** will accomplish your goals when the default trigonometric simplification does not. **tExpand()** tends to reverse transformations done by **tCollect()**. Sometimes applying **tCollect()** to a result from **tExpand()**, or vice versa, in two separate steps simplifies an expression.

Note: Degree-mode scaling by π/180 interferes with the ability of **tExpand()** to recognize expandable forms. For best results, **tExpand()** should be used in Radian mode.

```
tExpand(sin(3φ)) [ENTER]
```
$$4\cdot\sin(\phi)\cdot(\cos(\phi))^2-\sin(\phi)$$

```
tExpand(cos(α−β)) [ENTER]
```
$$\cos(\alpha)\cdot\cos(\beta)+\sin(\alpha)\cdot\sin(\beta)$$

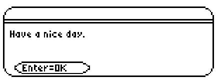
Text CATALOG

Text *promptString*

Displays the character string *promptString* dialog box.

If used as part of a **Dialog...EndDlog block**, *promptString* is displayed inside that dialog box. If used as a standalone instruction, **Text** creates a dialog box to display the string.

```
Text "Have a nice day." [ENTER]
                                Done
```



Then See **If,** page 826.

timeCnv() CATALOG

timeCnv(*seconds***)** ⇒ *list*

Converts seconds to units of time that can be more easily understood for evaluation. The list is in {*days,hours,minutes,seconds*} format.

Note: See also **checkTmr()** and **startTmr()**.

```
timeCnv(152442117)
                    {1764  9  1  57}
```

Title CATALOG

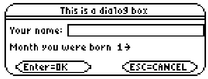
Title *titleString*, [*Lbl*]

Creates the title of a pull-down menu or dialog box when used inside a **Toolbar** or **Custom** construct, or a **Dialog...EndDlog** block.

Note: *Lbl* is only valid in the **Toolbar** construct. When present, it allows the menu choice to branch to a specified label inside the program.

Program segment:

```
 :
:Dialog
:Title   "This is a dialog
box"
:Request  "Your name",Str1
:Dropdown "Month you were
born",
  seq(string(i),i,1,12),Var1
:EndDlog
 :
```



tmpCnv() CATALOG

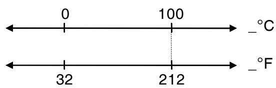
tmpCnv(*expression1_°tempUnit1*, *_°tempUnit2***)** ⇒ *expression_°tempUnit2*

Converts a temperature value specified by *expression1* from one unit to another. Valid temperature units are:

_°C	Celsius
_°F	Fahrenheit
_°K	Kelvin
_°R	Rankine

For °, press [2nd] [°].
For _, press [◆] [_].
For _, press [2nd] [_].

For example, 100_°C converts to 212_°F:



To convert a temperature range, use Δ**tmpCnv()** instead.

```
tmpCnv(100_°c,_°f) ENTER 212.•_°F
tmpCnv(32_°f,_°c) ENTER    0.•_°C
tmpCnv(0_°c,_°k) ENTER 273.15•_°K
tmpCnv(0_°f,_°r) ENTER 459.67•_°R
```

Note: To select temperature units from a menu, press:

[2nd] [UNITS]
[◆] [UNITS]

∆tmpCnv() CATALOG

∆tmpCnv(*expression1*_*°tempUnit1*, _*°tempUnit2***)** ⇒ *expression*_*°tempUnit2*

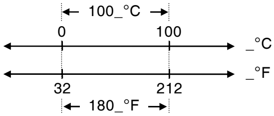
Converts a temperature range (the difference between two temperature values) specified by *expression1* from one unit to another. Valid temperature units are:

_°C	Celsius
_°F	Fahrenheit
_°K	Kelvin
_°R	Rankine

For °, press [2nd] [°].
For _, press [◆] [_].
For _, press [2nd] [_].

1_°C and 1_°K have the same magnitude, as do 1_°F and 1_°R. However, 1_°C is 9/5 as large as 1_°F.

For example, a 100_°C range (from 0_°C to 100_°C) is equivalent to a 180_°F range:



To convert a particular temperature value instead of a range, use **tmpCnv()**.

To get ∆, you can press [◆] [(] [↑] [D] (or [2nd] [CHAR] 1 5).

```
∆tmpCnv(100_°c,_°f) [ENTER]
                    180.•_°F
∆tmpCnv(180_°f,_°c) [ENTER]
                    100.•_°C
∆tmpCnv(100_°c,_°k) [ENTER]
                    100.•_°K
∆tmpCnv(100_°f,_°r) [ENTER]
                    100.•_°R
∆tmpCnv(1_°c,_°f) [ENTER]
                      1.8•_°F
```

Note: To select temperature units from a menu, press:

[2nd] [UNITS]
[◆] [UNITS]

Toolbar CATALOG

Toolbar
block
EndTBar

Creates a toolbar menu.

block can be either a single statement or a sequence of statements separated with the ":" character. The statements can be either Title or Item.

Items must have labels. A Title must also have a label if it does not have an item.

Program segment:

```
  :
:Toolbar
:  Title "Examples"
:  Item "Trig", t
:  Item "Calc", c
:  Item "Stop", Pexit
:EndTbar
  :
```

Note: When run in a program, this segment creates a menu with three choices that branch to three places in the program.

Trace CATALOG

Trace

Draws a Smart Graph and places the trace cursor on the first defined Y= function at the previously defined cursor position, or at the reset position if regraphing was necessary.

Allows operation of the cursor and most keys when editing coordinate values. Several keys, such as the function keys, [APPS], and [MODE], are not activated during trace.

Note: Press [ENTER] to resume operation.

Try CATALOG

Try
    block1
Else
    block2
EndTry

Executes *block1* unless an error occurs. Program execution transfers to *block2* if an error occurs in *block1*. Variable errornum contains the error number to allow the program to perform error recovery.

block1 and *block2* can be either a single statement or a series of statements separated with the ":" character.

Program segment:

```
   :
:Try
:  NewFold(temp)
:  Else
:  ©Already exists
:  ClrErr
:EndTry
   :
```

Note: See **ClrErr** and **PassErr**.

TwoVar MATH/Statistics menu

TwoVar *list1*, *list2*[, [*list3*] [, *list4*, *list5*]]

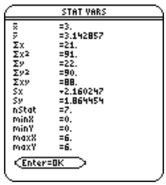
Calculates the **TwoVar** statistics and updates all the system statistics variables.

All the lists must have equal dimensions except for *list5*.

list1 represents xlist.
list2 represents ylist.
list3 represents frequency.
list4 represents category codes.
list5 represents category include list.

Note: *list1* through *list4* must be a variable name or c1–c99 (columns in the last data variable shown in the Data/Matrix Editor). *list5* does not have to be a variable name and cannot be c1–c99
.

```
{0,1,2,3,4,5,6}→L1 [ENTER]
                                {0 1 2 ...}
{0,2,3,4,3,4,6}→L2 [ENTER]
                                {0 2 3 ...}
TwoVar L1,L2 [ENTER]                   Done
ShowStat [ENTER]
```



Unarchiv CATALOG

Unarchiv *var1* [, *var2*] [, *var3*] …

Moves the specified variables from the user data archive memory to RAM.

You can access an archived variable the same as you would a variable in RAM. However, you cannot delete, rename, or store to an archived variable because it is locked automatically.

To archive variables, use **Archive**.

```
10→arctest [ENTER]                       10
Archive arctest [ENTER]                Done
5*arctest [ENTER]                        50
15→arctest [ENTER]
```


```
[ESC]
Unarchiv arctest [ENTER]               Done
15→arctest [ENTER]                       15
```

unitV() MATH/Matrix/Vector ops menu

unitV(*vector1***)** ⇒ *vector*

Returns either a row- or column-unit vector, depending on the form of *vector1*.

vector1 must be either a single-row matrix or a single-column matrix.

```
unitV([a,b,c]) [ENTER]
```
$$\left[\frac{a}{\sqrt{a^2+b^2+c^2}} \quad \frac{b}{\sqrt{a^2+b^2+c^2}} \quad \frac{c}{\sqrt{a^2+b^2+c^2}}\right]$$

```
unitV([1,2,1]) [ENTER]
```
$$\left[\frac{\sqrt{6}}{6} \quad \frac{\sqrt{6}}{3} \quad \frac{\sqrt{6}}{6}\right]$$

```
unitV([1;2;3]) [ENTER]
```
$$\begin{bmatrix}\frac{\sqrt{14}}{14}\\ \frac{\sqrt{14}}{7}\\ \frac{3\cdot\sqrt{14}}{14}\end{bmatrix}$$

Unlock CATALOG

Unlock *var1*[, *var2*][, *var3*]...

Unlocks the specified variables.

Note: The variables can be locked using the **Lock** command.

variance() MATH/Statistics menu

variance(*list*[, *freqlist*]**)** ⇒ *expression*

Returns the variance of *list*.

Each *freqlist* element counts the number of consecutive occurrences of the corresponding element in *list*.

Note: *list* must contain at least two elements.

```
variance({a,b,c}) [ENTER]
```
$$\frac{a^2-a\cdot(b+c)+b^2-b\cdot c+c^2}{3}$$

```
variance({1,2,5,⁻6,3,⁻2}) [ENTER]
                              31/2
variance({1,3,5},{4,6,2}) [ENTER]
                             68/33
```

---

variance(*matrix1*[, *freqmatrix*]**)** ⇒ *matrix*

Returns a row vector containing the variance of each column in *matrix1*.

Each *freqmatrix* element counts the number of consecutive occurrences of the corresponding element in *matrix1*.

Note: *matrix1* must contain at least two rows.

```
variance([1,2,5;⁻3,0,1;
.5,.7,3]) [ENTER]   [4.75  1.03  4]

variance([-1.1,2.2;3.4,5.1;
-2.3,4.3],[6,3;2,4;5,1]) [ENTER]
                 [3.91731,2.08411]
```

when() CATALOG

when(*condition*, *trueResult* [, *falseResult*]
[, *unknownResult*]**)** ⇒ *expression*

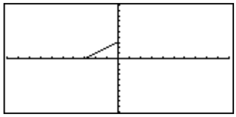
Returns *trueResult*, *falseResult*, or *unknownResult*, depending on whether *condition* is true, false, or unknown. Returns the input if there are too few arguments to specify the appropriate result.

Omit both *falseResult* and *unknownResult* to make an expression defined only in the region where *condition* is true.

```
when(x<0,x+3)|x=5 [ENTER]
                     when(x<0,3+x)
```

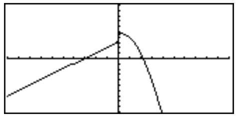
Use an undef *falseResult* to define an expression that graphs only on an interval.

```
ClrGraph [ENTER]
Graph when(x≥⁻π and
x<0,x+3,undef) [ENTER]
```



Omit only the *unknownResult* to define a two-piece expression.

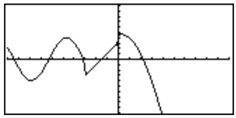
```
Graph when(x<0,x+3,5-x^2) [ENTER]
```



Nest **when()** to define expressions that have more than two pieces.

[HOME]
[♦] [CALC HOME]

```
ClrGraph [ENTER]                  Done
Graph when(x<0,when(x<⁻π,
4*sin(x),2x+3),5-x^2) [ENTER]
```



when() is helpful for defining recursive functions.

```
when(n>0,n*factoral(n-1),1)
→factoral(n) [ENTER]              Done
factoral(3) [ENTER]                  6
3! [ENTER]                           6
```

While   CATALOG

While *condition*
block
EndWhile

Executes the statements in *block* as long as *condition* is true.

block can be either a single statement or a sequence of statements separated with the ":" character.

Program segment:

```
  :
:1→i
:0→temp
:While i<=20
:  temp+1/i→temp
:  i+1→i
:EndWhile
:Disp "sum of reciprocals up to
  20",temp
  :
```

"With"   See |, page 907.

xor — MATH/Test menu

Boolean expression1 **xor** *Boolean expression2* ⇒ *Boolean expression*

Returns true if *Boolean expression1* is **true** and *Boolean expression2* is false, or vice versa. Returns false if *Boolean expression1* and *Boolean expression2* are both true or both false. Returns a simplified Boolean expression if either of the original Boolean expressions cannot be resolved to true or false.

Note: See **or**.

```
true xor true [ENTER]          false
(5>3) xor (3>5) [ENTER]         true
```

---

integer1 **xor** *integer2* ⇒ *integer*

Compares two real integers bit-by-bit using an **xor** operation. Internally, both integers are converted to signed, 32-bit binary numbers. When corresponding bits are compared, the result is 1 if either bit (but not both) is 1; the result is 0 if both bits are 0 or both bits are 1. The returned value represents the bit results, and is displayed according to the Base mode.

You can enter the integers in any number base. For a binary or hexadecimal entry, you must use the 0b or 0h prefix, respectively. Without a prefix, integers are treated as decimal (base 10).

If you enter a decimal integer that is too large for a signed, 32-bit binary form, a symmetric modulo operation is used to bring the value into the appropriate range.

Note: See **or**.

In Hex base mode:

```
0h7AC36 xor 0h3D5F [ENTER] 0h79169
```

Important: Zero, not the letter O.

In Bin base mode:

```
0b100101 xor 0b100 [ENTER]0b100001
```

Note: A binary entry can have up to 32 digits (not counting the 0b prefix). A hexadecimal entry can have up to 8 digits.

XorPic — CATALOG

XorPic *picVar*[, *row*] [, *column*]

Displays the picture stored in *picVar* on the current Graph screen.

Uses **xor** logic for each pixel. Only those pixel positions that are exclusive to either the screen or the picture are turned on. This instruction turns off pixels that are turned on in both images.

picVar must contain a pic data type.

row and *column*, if included, specify the pixel coordinates for the upper left corner of the picture. Defaults are (0, 0).

zeros() MATH/Algebra menu

zeros(*expression*, *var***)** ⇒ *list*

Returns a list of candidate real values of *var* that make *expression*=0. **zeros()** does this by computing **exp▶list(solve(***expression*=0,*var***)**,*var***)**.

For some purposes, the result form for **zeros()** is more convenient than that of **solve()**. However, the result form of **zeros()** cannot express implicit solutions, solutions that require inequalities, or solutions that do not involve *var*.

Note: See also **cSolve()**, **cZeros()**, and **solve()**.

```
zeros(a*x^2+b*x+c,x) [ENTER]
```

$$\left\{\frac{^-(\sqrt{b^2-4\cdot a\cdot c}+b)}{2\cdot a}\ \frac{\sqrt{b^2-4\cdot a\cdot c}-b}{2\cdot a}\right\}$$

```
a*x^2+b*x+c|x=ans(1)[2] [ENTER]     0
```

```
exact(zeros(a*(e^(x)+x)
  (sign (x)-1),x)) [ENTER]          {}
```

```
exact(solve(a*(e^(x)+x)
  (sign (x)-1)=0,x)) [ENTER]
```

$e^x + x = 0$ or $x>0$ or $a = 0$

---

zeros({*expression1*, *expression2***}**, **{***varOrGuess1*, *varOrGuess2* [**,** …]**})** ⇒ *matrix*

Returns candidate real zeros of the simultaneous algebraic *expressions*, where each *varOrGuess* specifies an unknown whose value you seek.

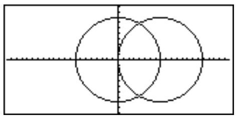
Optionally, you can specify an initial guess for a variable. Each *varOrGuess* must have the form:

variable
– or –
variable = *real or non-real number*

For example, x is valid and so is x=3.

If all of the expressions are polynomials and if you do NOT specify any initial guesses, **zeros()** uses the lexical Gröbner/Buchberger elimination method to attempt to determine **all** real zeros.

For example, suppose you have a circle of radius r at the origin and another circle of radius r centered where the first circle crosses the positive x-axis. Use **zeros()** to find the intersections.



As illustrated by r in the example to the right, simultaneous *polynomial* expressions can have extra variables that have no values, but represent given numeric values that could be substituted later.

```
zeros({x^2+y^2-r^2,
(x-r)^2+y^2-r^2},{x,y}) [ENTER]
```

$$\begin{bmatrix} \frac{r}{2} & \frac{\sqrt{3}\cdot r}{2} \\ \frac{r}{2} & \frac{^-\sqrt{3}\cdot r}{2} \end{bmatrix}$$

Each row of the resulting matrix represents an alternate zero, with the components ordered the same as the *varOrGuess* list. To extract a row, index the matrix by [*row*].

Extract row 2:

```
ans(1)[2] [ENTER]
```

$$\begin{bmatrix} \frac{r}{2} & \frac{^-\sqrt{3}\cdot r}{2} \end{bmatrix}$$

You can also (or instead) include unknowns that do not appear in the expressions. For example, you can include z as an unknown to extend the previous example to two parallel intersecting cylinders of radius r. The cylinder zeros illustrate how families of zeros might contain arbitrary constants in the form @*k*, where *k* is an integer suffix from 1 through 255. The suffix resets to 1 when you use **ClrHome** or [F1] 8:Clear Home.

```
zeros({x^2+y^2-r^2,
(x-r)^2+y^2-r^2},{x,y,z})
[ENTER]
```

$$\begin{bmatrix} \frac{r}{2} & \frac{\sqrt{3}\cdot r}{2} & @1 \\ \frac{r}{2} & \frac{^{-}\sqrt{3}\cdot r}{2} & @1 \end{bmatrix}$$

For polynomial systems, computation time or memory exhaustion may depend strongly on the order in which you list unknowns. If your initial choice exhausts memory or your patience, try rearranging the variables in the expressions and/or *varOrGuess* list.

If you do not include any guesses and if any expression is non-polynomial in any variable but all expressions are linear in the unknowns, **zeros()** uses Gaussian elimination to attempt to determine all real zeros.

```
zeros({x+e^(z)*y-1,x-y-sin(z)
},{x,y}) [ENTER]
```

$$\begin{bmatrix} \frac{e^{z}\cdot \sin(z)+1}{e^{z}+1} & \frac{^{-}(\sin(z)-1)}{e^{z}+1} \end{bmatrix}$$

If a system is neither polynomial in all of its variables nor linear in its unknowns, **zeros()** determines at most one zero using an approximate iterative method. To do so, the number of unknowns must equal the number of expressions, and all other variables in the expressions must simplify to numbers.

```
zeros({e^(z)*y-1,⁻y-sin(z)},
{y,z}) [ENTER]
```

[.041… 3.183…]

Each unknown starts at its guessed value if there is one; otherwise, it starts at 0.0.

Use guesses to seek additional zeros one by one. For convergence, a guess may have to be rather close to a zero.

```
zeros({e^(z)*y-1,⁻y-sin(z)},
{y,z=2π}) [ENTER]
```

[.001… 6.281…]

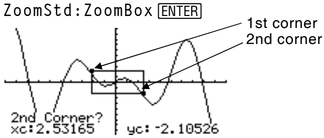
ZoomBox CATALOG

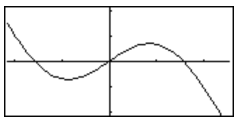
ZoomBox

Displays the Graph screen, lets you draw a box that defines a new viewing window, and updates the window.

In function graphing mode:

```
1.25x*cos(x)→y1(x) [ENTER]      Done
ZoomStd:ZoomBox [ENTER]
```





The display after defining ZoomBox by pressing [ENTER] the second time.

ZoomData CATALOG

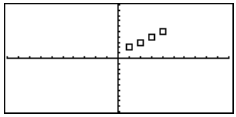
ZoomData

Adjusts the window settings based on the currently defined plots (and data) so that all statistical data points will be sampled, and displays the Graph screen.

Note: Does not adjust ymin and ymax for histograms.

In function graphing mode:

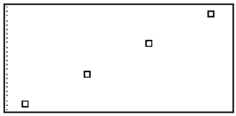
```
{1,2,3,4}→L1 [ENTER]          {1 2 3 4}
{2,3,4,5}→L2 [ENTER]          {2 3 4 5}
newPlot 1,1,L1,L2 [ENTER]          Done
ZoomStd [ENTER]
```



[HOME]

[◆] [CALC HOME]

```
ZoomData [ENTER]
```



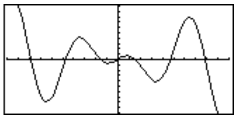
ZoomDec CATALOG

ZoomDec

Adjusts the viewing window so that ∆x and ∆y = 0.1 and displays the Graph screen with the origin centered on the screen.

In function graphing mode:

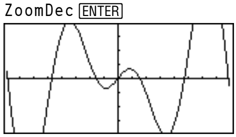
```
1.25x*cos(x)→y1(x) [ENTER]          Done
ZoomStd [ENTER]
```



[HOME]

[◆] [CALC HOME]

```
ZoomDec [ENTER]
```



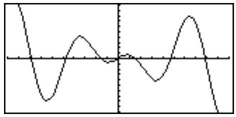
ZoomFit CATALOG

ZoomFit

Displays the Graph screen, and calculates the necessary window dimensions for the dependent variables to view all the picture for the current independent variable settings.

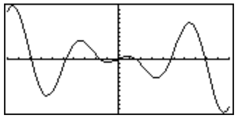
In function graphing mode:

`1.25x*cos(x)→y1(x)` [ENTER] `Done`
`ZoomStd` [ENTER]



[HOME]
[♦] [CALC HOME]

`ZoomFit` [ENTER]



ZoomIn CATALOG

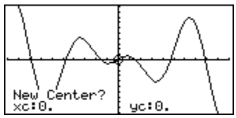
ZoomIn

Displays the Graph screen, lets you set a center point for a zoom in, and updates the viewing window.

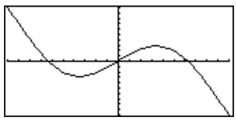
The magnitude of the zoom is dependent on the Zoom factors xFact and yFact. In 3D Graph mode, the magnitude is dependent on xFact, yFact, and zFact.

In function graphing mode:

`1.25x*cos(x)→y1(x)` [ENTER] `Done`
`ZoomStd:ZoomIn` [ENTER]



[ENTER]



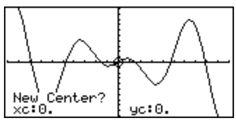
ZoomInt CATALOG

ZoomInt

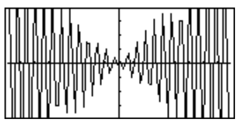
Displays the Graph screen, lets you set a center point for the zoom, and adjusts the window settings so that each pixel is an integer in all directions.

In function graphing mode:

`1.25x*cos(x)→y1(x)` [ENTER] `Done`
`ZoomStd:ZoomInt` [ENTER]



[ENTER]



ZoomOut CATALOG

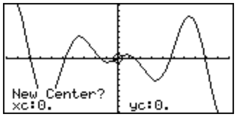
ZoomOut

Displays the Graph screen, lets you set a center point for a zoom out, and updates the viewing window.

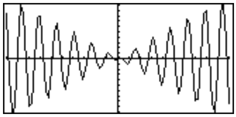
The magnitude of the zoom is dependent on the Zoom factors xFact and yFact. In 3D Graph mode, the magnitude is dependent on xFact, yFact, and zFact.

In function graphing mode:

```
1.25x*cos(x)→y1(x) [ENTER]      Done
ZoomStd:ZoomOut [ENTER]
```



[ENTER]



ZoomPrev CATALOG

ZoomPrev

Displays the Graph screen, and updates the viewing window with the settings in use before the last zoom.

ZoomRcl CATALOG

ZoomRcl

Displays the Graph screen, and updates the viewing window using the settings stored with the **ZoomSto** instruction.

ZoomSqr CATALOG

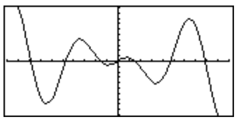
ZoomSqr

Displays the Graph screen, adjusts the x or y window settings so that each pixel represents an equal width and height in the coordinate system, and updates the viewing window.

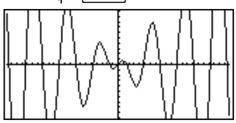
In 3D Graph mode, **ZoomSqr** lengthens the shortest two axes to be the same as the longest axis.

In function graphing mode:

```
1.25x*cos(x)→y1(x) [ENTER]      Done
ZoomStd [ENTER]
```



[HOME]
```
ZoomSqr [ENTER]
```



ZoomStd CATALOG

ZoomStd

Sets the window variables to the following standard values, and then updates the viewing window.

Function graphing:
x: [⁻10, 10, 1], y: [⁻10, 10, 1] and xres=2

Parametric graphing:
t: [0, 2π, π/24], x: [⁻10, 10, 1], y:[⁻10, 10, 1]

Polar graphing:
θ: [0, 2π, π/24], x: [⁻10, 10, 1], y: [⁻10, 10, 1]

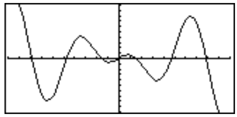
Sequence graphing:
nmin=1, nmax=10, plotStrt=1, plotStep=1,
x: [⁻10, 10, 1], y: [⁻10, 10, 1]

3D graphing:
eyeθ°=20, eyeϕ°=70, eyeψ°=0
x: [⁻10, 10, 14], y: [⁻10, 10, 14],
z: [⁻10, 10], ncontour=5

Differential equations graphing:
t: [0, 10, .1, 0], x: [⁻1, 10, 1], y: [⁻10, 10, 1],
ncurves=0, Estep=1, diftol=.001, fldres=14,
dtime=0

In function graphing mode:

```
1.25x*cos(x)→y1(x) [ENTER]      Done
ZoomStd [ENTER]
```



ZoomSto CATALOG

ZoomSto

Stores the current Window settings in the Zoom memory. You can use **ZoomRcl** to restore the settings.

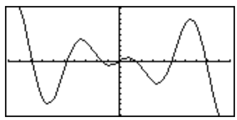
ZoomTrig CATALOG

ZoomTrig

Displays the Graph screen, sets Δx to π/24, and xscl to π/2, centers the origin, sets the y settings to [⁻4, 4, .5], and updates the viewing window.

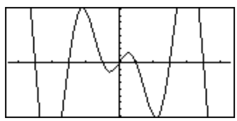
In function graphing mode:

```
1.25x*cos(x)→y1(x) [ENTER]      Done
ZoomStd [ENTER]
```



[HOME]
[♦] [CALC HOME]

```
ZoomTrig [ENTER]
```



+ (add) [+] **key**

expression1 + *expression2* ⇒ *expression*

Returns the sum of *expression1* and *expression2*.

```
56 [ENTER]                 56
ans(1)+4 [ENTER]           60
ans(1)+4 [ENTER]           64
ans(1)+4 [ENTER]           68
ans(1)+4 [ENTER]           72
```

---

list1 + *list2* ⇒ *list*
matrix1 + *matrix2* ⇒ *matrix*

Returns a list (or matrix) containing the sums of corresponding elements in *list1* and *list2* (or *matrix1* and *matrix2*).

Dimensions of the arguments must be equal.

```
{22,π,π/2}→L1 [ENTER]      {22 π π/2}
{10,5,π/2}→L2 [ENTER]      {10 5 π/2}
L1+L2 [ENTER]              {32 π+5 π}

ans(1)+{π,⁻5,⁻π} [ENTER]
                           {π+32 π 0}

[a,b;c,d]+[1,0;0,1] [ENTER]
```

$$\begin{bmatrix} a+1 & b \\ c & d+1 \end{bmatrix}$$

---

expression + *list1* ⇒ *list*
list1 + *expression* ⇒ *list*

Returns a list containing the sums of *expression* and each element in *list1*.

```
15+{10,15,20} [ENTER]      {25 30 35}

{10,15,20}+15 [ENTER]      {25 30 35}
```

---

expression + *matrix1* ⇒ *matrix*
matrix1 + *expression* ⇒ *matrix*

Returns a matrix with *expression* added to each element on the diagonal of *matrix1*. *matrix1* must be square.

Note: Use **.+** (dot plus) to add an expression to each element.

```
20+[1,2;3,4] [ENTER]
```

$$\begin{bmatrix} 21 & 2 \\ 3 & 24 \end{bmatrix}$$

− (subtract) [−] **key**

expression1 − *expression2* ⇒ *expression*

Returns *expression1* minus *expression2*.

```
6−2 [ENTER]                4
```

```
π−π/6 [ENTER]
```

$$\frac{5 \cdot \pi}{6}$$

---

list1 − *list2* ⇒ *list*
matrix1 − *matrix2* ⇒ *matrix*

Subtracts each element in *list2* (or *matrix2*) from the corresponding element in *list1* (or *matrix1*), and returns the results.

Dimensions of the arguments must be equal.

```
{22,π,π/2}−{10,5,π/2} [ENTER]
                           {12 π−5 0}

[3,4]−[1,2] [ENTER]        [2 2]
```

---

expression − *list1* ⇒ *list*
list1 − *expression* ⇒ *list*

Subtracts each *list1* element from *expression* or subtracts *expression* from each *list1* element, and returns a list of the results.

```
15−{10,15,20} [ENTER]      {5 0 -5}

{10,15,20}−15 [ENTER]      {-5 0 5}
```

expression − *matrix1* ⇒ *matrix*
matrix1 − *expression* ⇒ *matrix*

expression − *matrix1* returns a matrix of *expression* times the identity matrix minus *matrix1*. *matrix1* must be square.

matrix1 − *expression* returns a matrix of *expression* times the identity matrix subtracted from *matrix1*. *matrix1* must be square.

Note: Use **.−** (dot minus) to subtract an expression from each element.

`20-[1,2;3,4]` [ENTER] $\begin{bmatrix} 19 & ^-2 \\ ^-3 & 16 \end{bmatrix}$

∗ (multiply) [×] key

expression1 ∗ *expression2* ⇒ *expression*

Returns the product of *expression1* and *expression2*.

`2*3.45` [ENTER] 6.9

`x*y*x` [ENTER] $x^2 \cdot y$

---

list1∗ *list2* ⇒ *list*

Returns a list containing the products of the corresponding elements in *list1* and *list2*.

Dimensions of the lists must be equal.

`{1.0,2,3}*{4,5,6}` [ENTER] {4. 10 18}

`{2/a,3/2}*{a²,b/3}` [ENTER] $\{2\cdot a \quad \frac{b}{2}\}$

---

matrix1 ∗ *matrix2* ⇒ *matrix*

Returns the matrix product of *matrix1* and *matrix2*.

The number of rows in *matrix1* must equal the number of columns in *matrix2*.

`[1,2,3;4,5,6]*[a,d;b,e;c,f]` [ENTER]

$\blacksquare \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \cdot \begin{bmatrix} a & d \\ b & e \\ c & f \end{bmatrix}$

$\begin{bmatrix} a+2\cdot b+3\cdot c & d+2\cdot e+3 \\ 4\cdot a+5\cdot b+6\cdot c & 4\cdot d+5\cdot e\cdot \end{bmatrix}$ ▸

---

expression ∗ *list1* ⇒ *list*
list1 ∗ *expression* ⇒ *list*

Returns a list containing the products of *expression* and each element in *list1*.

`π*{4,5,6}` [ENTER] {4•π 5•π 6•π}

---

expression ∗ *matrix1* ⇒ *matrix*
matrix1 ∗ *expression* ⇒ *matrix*

Returns a matrix containing the products of *expression* and each element in *matrix1*.

Note: Use **.∗** (dot multiply) to multiply an expression by each element.

`[1,2;3,4]*.01` [ENTER] $\begin{bmatrix} .01 & .02 \\ .03 & .04 \end{bmatrix}$

`λ*identity(3)` [ENTER] $\begin{bmatrix} \lambda & 0 & 0 \\ 0 & \lambda & 0 \\ 0 & 0 & \lambda \end{bmatrix}$

/ (divide) [÷] key

expression1 / *expression2* ⇒ *expression*

Returns the quotient of *expression1* divided by *expression2*.

`2/3.45` [ENTER] .57971

`x^3/x` [ENTER] x^2

---

list1 / *list2* ⇒ *list*

Returns a list containing the quotients of *list1* divided by *list2.*

Dimensions of the lists must be equal.

`{1.0,2,3}/{4,5,6}` [ENTER] {.25 2/5 1/2}

expression / *list1* ⇒ *list*
list1 / *expression* ⇒ *list*

Returns a list containing the quotients of *expression* divided by *list1* or *list1* divided by *expression*.

```
a/{3,a,√(a)} [ENTER]
```
$\left\{\frac{a}{3}\quad 1\quad \sqrt{a}\right\}$

```
{a,b,c}/(a*b*c) [ENTER]
```
$\left\{\frac{1}{b\cdot c}\quad \frac{1}{a\cdot c}\quad \frac{1}{a\cdot b}\right\}$

---

matrix1 / *expression* ⇒ *matrix*

Returns a matrix containing the quotients of *matrix1*/*expression*.

Note: Use **./** (dot divide) to divide an expression by each element.

```
[a,b,c]/(a*b*c) [ENTER]
```
$\left[\frac{1}{b\cdot c}\quad \frac{1}{a\cdot c}\quad \frac{1}{a\cdot b}\right]$

^ (power) [^] key

expression1 ^ *expression2* ⇒ *expression*
list1 ^ *list2* ⇒ *list*

Returns the first argument raised to the power of the second argument.

For a list, returns the elements in *list1* raised to the power of the corresponding elements in *list2*.

In the real domain, fractional powers that have reduced exponents with odd denominators use the real branch versus the principal branch for complex mode.

```
4^2 [ENTER]                              16
{a,2,c}^{1,b,3} [ENTER]       {a  2^b  c^3}
```

---

expression ^ *list1* ⇒ *list*

Returns *expression* raised to the power of the elements in *list1*.

```
p^{a,2,⁻3} [ENTER]
```
$\left\{p^{a}\quad p^{2}\quad \frac{1}{p^{3}}\right\}$

---

list1 ^ *expression* ⇒ *list*

Returns the elements in *list1* raised to the power of *expression*.

```
{1,2,3,4}^⁻2 [ENTER]
                  {1  1/4  1/9  1/16}
```

---

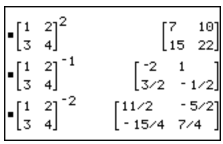
squareMatrix1 ^ *integer* ⇒ *matrix*

Returns *squareMatrix1* raised to the *integer* power.

squareMatrix1 must be a square matrix.

If *integer* = ⁻1, computes the inverse matrix.
If *integer* < ⁻1, computes the inverse matrix to an appropriate positive power.

```
[1,2;3,4]^2 [ENTER]
[1,2;3,4]^⁻1 [ENTER]
[1,2;3,4]^⁻2 [ENTER]
```



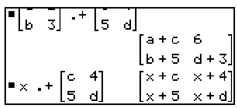
.+ (dot add) [.] [+] keys

matrix1 .+ *matrix2* ⇒ *matrix*
expression .+ *matrix1* ⇒ *matrix*

matrix1 .+ *matrix2* returns a matrix that is the sum of each pair of corresponding elements in *matrix1* and *matrix2*.

expression .+ *matrix1* returns a matrix that is the sum of *expression* and each element in *matrix1*.

```
[a,2;b,3].+[c,4;5,d] [ENTER]
x.+[c,4;5,d] [ENTER]
```



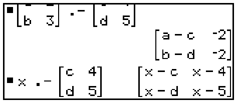
.– (dot subt.) [.] [–] keys

matrix1 .– *matrix2* ⇒ *matrix*
expression .– *matrix1* ⇒ *matrix*

matrix1 .– *matrix2* returns a matrix that is the difference between each pair of corresponding elements in *matrix1* and *matrix2*.

expression .– *matrix1* returns a matrix that is the difference of *expression* and each element in *matrix1*.

`[a,2;b,3].-[c,4;d,5]` [ENTER]
`x.-[c,4;d,5]` [ENTER]



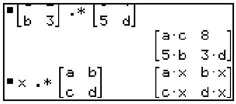
.∗ (dot mult.) [.] [×] keys

matrix1 .∗ *matrix2* ⇒ *matrix*
expression .∗ *matrix1* ⇒ *matrix*

matrix1 .∗ *matrix2* returns a matrix that is the product of each pair of corresponding elements in *matrix1* and *matrix2*.

expression .∗ *matrix1* returns a matrix containing the products of *expression* and each element in *matrix1*.

`[a,2;b,3].*[c,4;5,d]` [ENTER]
`x.*[a,b;c,d]` [ENTER]



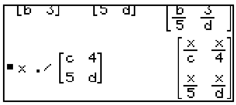
./ (dot divide) [.] [÷] keys

matrix1 ./ *matrix2* ⇒ *matrix*
expression ./ *matrix1* ⇒ *matrix*

matrix1 ./ *matrix2* returns a matrix that is the quotient of each pair of corresponding elements in *matrix1* and *matrix2*.

expression ./ *matrix1* returns a matrix that is the quotient of *expression* and each element in *matrix1*.

`[a,2;b,3]./[c,4;5,d]` [ENTER]
`x./[c,4;5,d]` [ENTER]



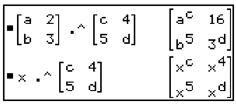
.^ (dot power) [.] [^] keys

matrix1 .^ *matrix2* ⇒ *matrix*
expression .^ *matrix1* ⇒ *matrix*

matrix1 .^ *matrix2* returns a matrix where each element in *matrix2* is the exponent for the corresponding element in *matrix1*.

expression .^ *matrix1* returns a matrix where each element in *matrix1* is the exponent for *expression*.

`[a,2;b,3].^[c,4;5,d]` [ENTER]
`x.^[c,4;5,d]` [ENTER]



⁻ (negate) [(-)] key and MATH/Base menu

⁻*expression1* ⇒ *expression*
⁻ *list1* ⇒ *list*
⁻ *matrix1* ⇒ *matrix*

Returns the negation of the argument.

For a list or matrix, returns all the elements negated.

If *expression1* is a binary or hexadecimal integer, the negation gives the two's complement.

`⁻2.43` [ENTER] `⁻2.43`

`⁻{⁻1,0.4,1.2E19}` [ENTER]
`{1 ⁻.4 ⁻1.2E19}`

`⁻a*⁻b` [ENTER] `a•b`

In Bin base mode:

`0b100101▶dec` [ENTER] `37`

└─ **Important:** Zero, not the letter O.

`⁻0b100101` [ENTER]
`0b11111111111111111111111111011011`

`ans(1)▶dec` [ENTER] `⁻37`

Note: To type ▶, press [2nd] [▶].

% (percent) **CHAR/Punctuation menu**

expression1 **%** ⇒ *expression*
list1 **%** ⇒ *list*
matrix1 **%** ⇒ *matrix*

Returns $\frac{argument}{100}$.

For a list or matrix, returns a list or matrix with each element divided by 100.

```
13% ◆ [ENTER]                .13

{1, 10, 100}% ◆ [ENTER]
                     {.01 .1  1.}
```

= (equal) [=] **key**

expression1 **=** *expression2* ⇒ *Boolean expression*
list1 **=** *list2* ⇒ *Boolean list*
matrix1 **=** *matrix2* ⇒ *Boolean matrix*

Returns true if *expression1* is determined to be equal to *expression2*.

Returns false if *expression1* is determined to not be equal to *expression2*.

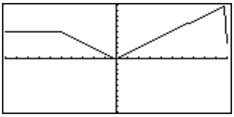
Anything else returns a simplified form of the equation.

For lists and matrices, returns comparisons element by element.

Example function listing using math test symbols: =, ≠, <, ≤, >, ≥

```
:g(x)
:Func
:If x≤⁻5 Then
:  Return 5
:  ElseIf x>⁻5 and x<0 Then
:  Return ⁻x
:  ElseIf x≥0 and x≠10 Then
:  Return x
:  ElseIf x=10 Then
:  Return 3
:EndIf
:EndFunc
```

```
Graph g(x) [ENTER]
```



≠ ◆ [=] **key**

expression1 **≠** *expression2* ⇒ *Boolean expression*
list1 **≠** *list2* ⇒ *Boolean list*
matrix1 **≠** *matrix2* ⇒ *Boolean matrix*

Returns true if *expression1* is determined to be not equal to *expression2*.

Returns false if *expression1* is determined to be equal to *expression2*.

Anything else returns a simplified form of the equation.

For lists and matrices, returns comparisons element by element.

See "=" (equal) example.

< [2nd] [<] key

expression1 < *expression2* ⇒ *Boolean expression*
list1 < *list2* ⇒ *Boolean list*
matrix1 < *matrix2* ⇒ *Boolean matrix*

See "=" (equal) example.

Returns true if *expression1* is determined to be less than *expression2*.

Returns false if *expression1* is determined to be greater than or equal to *expression2*.

Anything else returns a simplified form of the equation.

For lists and matrices, returns comparisons element by element.

≤ [♦] [0] key

expression1 ≤ *expression2* ⇒ *Boolean expression*
list1 ≤ *list2* ⇒ *Boolean list*
matrix1 ≤ *matrix2* ⇒ *Boolean matrix*

See "=" (equal) example.

Returns true if *expression1* is determined to be less than or equal to *expression2*.

Returns false if *expression1* is determined to be greater than *expression2*.

Anything else returns a simplified form of the equation.

For lists and matrices, returns comparisons element by element.

> [2nd] [>] key

expression1 > *expression2* ⇒ *Boolean expression*
list1 > *list2* ⇒ *Boolean list*
matrix1 > *matrix2* ⇒ *Boolean matrix*

See "=" (equal) example.

Returns true if *expression1* is determined to be greater than *expression2*.

Returns false if *expression1* is determined to be less than or equal to *expression2*.

Anything else returns a simplified form of the equation.

For lists and matrices, returns comparisons element by element.

≥ [♦] [.] key

expression1 ≥ *expression2* ⇒ *Boolean expression*
list1 ≥ *list2* ⇒ *Boolean list*
matrix1 ≥ *matrix2* ⇒ *Boolean matrix*

See "=" (equal) example.

Returns true if *expression1* is determined to be greater than or equal to *expression2*.

Returns false if *expression1* is determined to be less than *expression2*.

Anything else returns a simplified form of the equation.

For lists and matrices, returns comparisons element by element.

! (factorial) [calc] ◆ ÷ key [keyboard] 2nd W key

expression1! ⇒ *expression*
list1! ⇒ *list*
matrix1! ⇒ *matrix*

Returns the factorial of the argument.

For a list or matrix, returns a list or matrix of factorials of the elements.

The TI-89 computes a numeric value for only non-negative whole-number values.

```
5! [ENTER]                 120
{5,4,3}! [ENTER]     {120 24 6}
[1,2;3,4]! [ENTER]     [1  2 ]
                       [6  24]
```

& (append) [calc] ◆ × key [keyboard] 2nd H key

string1 & *string2* ⇒ *string*

Returns a text string that is *string2* appended to *string1*.

```
"Hello " & "Nick" [ENTER]
                   "Hello Nick"
```

∫() (integrate) 2nd [∫] key

∫(*expression1*, *var*[, *lower*] [,*upper*]) ⇒ *expression*
∫(*list1*,*var*[,*order*]) ⇒ *list*
∫(*matrix1*,*var*[,*order*]) ⇒ *matrix*

Returns the integral of *expression1* with respect to the variable *var* from *lower* to *upper*.

```
∫(x^2,x,a,b) [ENTER]
```

$\frac{b^3}{3} - \frac{a^3}{3}$

Returns an anti-derivative if *lower* and *upper* are omitted. A symbolic constant of integration such as C is omitted.

```
∫(x^2,x) [ENTER]
```

$\frac{x^3}{3}$

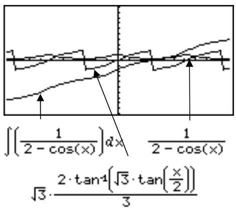
However, *lower* is added as a constant of integration if only *upper* is omitted.

```
∫(a*x^2,x,c) [ENTER]
```

$\frac{a \cdot x^3}{3} + c$

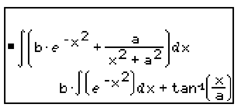
Equally valid anti-derivatives might differ by a numeric constant. Such a constant might be disguised—particularly when an anti-derivative contains logarithms or inverse trigonometric functions. Moreover, piecewise constant expressions are sometimes added to make an anti-derivative valid over a larger interval than the usual formula.

```
∫(1/(2-cos(x)),x)→tmp(x) [ENTER]
ClrGraph:Graph tmp(x):Graph
  1/(2-cos(x)):Graph √(3)
  (2tan⁻¹(√(3)(tan(x/2)))/3)
  [ENTER]
```



∫() returns itself for pieces of *expression1* that it cannot determine as an explicit finite combination of its built-in functions and operators.

```
∫(b*e^(⁻x^2)+a/(x^2+a^2),x)
[ENTER]
```



When *lower* and *upper* are both present, an attempt is made to locate any discontinuities or discontinuous derivatives in the interval *lower* < *var* < *upper* and to subdivide the interval at those places.

For the AUTO setting of the Exact/Approx mode, numerical integration is used where applicable when an anti-derivative or a limit cannot be determined.

For the APPROX setting, numerical integration is tried first, if applicable. Anti-derivatives are sought only where such numerical integration is inapplicable or fails.

`∫(e^(⁻x^2),x,⁻1,1)` [◆] [ENTER] `1.493...`

∫() can be nested to do multiple integrals. Integration limits can depend on integration variables outside them.

Note: See also **nInt()**.

`∫(∫(ln(x+y),y,0,x),x,0,a)` [ENTER]

$$\int_0^a \int_0^x \ln(x+y)\,dy\,dx$$

$$\frac{a^2 \cdot \ln(a)}{2} + a^2 \cdot (\ln(2) - 3/4)$$

√() (square root) [2nd] [√] key

√ **(***expression1***)** ⇒ *expression*
√ **(***list1***)** ⇒ *list*

Returns the square root of the argument.

For a list, returns the square roots of all the elements in *list1.*

`√(4)` [ENTER] `2`

`√({9,a,4})` [ENTER] `{3  √a  2}`

Π() (product) MATH/Calculus menu

Π(*expression1***,** *var***,** *low***,** *high***)** ⇒ *expression*

Evaluates *expression1* for each value of *var* from *low* to *high*, and returns the product of the results.

`Π(1/n,n,1,5)` [ENTER] $\frac{1}{120}$

`Π(k^2,k,1,n)` [ENTER] `(n!)²`

`Π({1/n,n,2},n,1,5)` [ENTER] $\{\frac{1}{120}\ \ 120\ \ 32\}$

Π(*expression1***,** *var***,** *low***,** *low*− 1**)** ⇒ 1

`Π(k,k,4,3)` [ENTER] `1`

Π(*expression1***,** *var***,** *low***,** *high***)** ⇒ 1/Π(*expression1, var, high*+1, *low*− 1) if *high* < *low*− 1

`Π(1/k,k,4,1)` [ENTER] `6`

`Π(1/k,k,4,1)*Π(1/k,k,2,4)` [ENTER] `1/4`

Σ() (sum) MATH/Calculus menu

Σ**(***expression1***,** *var***,** *low***,** *high***)** ⇒ *expression*

Evaluates *expression1* for each value of *var* from *low* to *high*, and returns the sum of the results.

`Σ(1/n,n,1,5)` [ENTER] $\frac{137}{60}$

`Σ(k^2,k,1,n)` [ENTER]

$$\frac{n\cdot(n+1)\cdot(2\cdot n+1)}{6}$$

`Σ(1/n^2,n,1,∞)` [ENTER] $\frac{\pi^2}{6}$

Σ**(***expression1***,** *var***,** *low***,** *low*− 1**)** ⇒ 0

`Σ(k,k,4,3)` [ENTER] `0`

Σ**(***expression1***,** *var***,** *low***,** *high***)** ⇒ ⁻Σ(*expression1, var, high*+1, *low*− 1) if *high* < *low*− 1

`Σ(k,k,4,1)` [ENTER] `⁻5`

`Σ(k,k,4,1)+Σ(k,k,2,4)` [ENTER] `4`

(indirection) **CATALOG**

*varNameString*

Refers to the variable whose name is *varNameString*. This lets you create and modify variables from a program using strings.

Program segment:

```
   :
:Request "Enter Your Name",str1
:NewFold #str1
   :

   :
:For i,1,5,1
:  ClrGraph
:  Graph i*x
:  StoPic #("pic" & string(i))
:EndFor
   :
```

^{r} (radian) **MATH/Angle menu**

expression1^r ⇒ *expression*
list1^r ⇒ *list*
matrix1^r ⇒ *matrix*

In Degree angle mode, multiplies *expression1* by 180/π. In Radian angle mode, returns *expression1* unchanged.

This function gives you a way to use a radian angle while in Degree mode. (In Degree angle mode, **sin()**, **cos()**, **tan()**, and polar-to-rectangular conversions expect the angle argument to be in degrees.)

Hint: Use ^{r} if you want to force radians in a function or program definition regardless of the mode that prevails when the function or program is used.

In Degree or Radian angle mode:

`cos((π/4)`^{r}`)` [ENTER] $\frac{\sqrt{2}}{2}$

`cos({0`^{r}`,(π/12)`^{r}`,⁻π`^{r}`})` [ENTER]

$\{1 \quad \frac{(\sqrt{3}+1)\cdot\sqrt{2}}{4} \quad -1\}$

° (degree) [2nd] [°] **key**

expression° ⇒ *value*
list1° ⇒ *list*
matrix1° ⇒ *matrix*

In Radian angle mode, multiplies *expression* by π/180. In Degree angle mode, returns *expression* unchanged.

This function gives you a way to use a degree angle while in Radian mode. (In Radian angle mode, **sin()**, **cos()**, **tan()**, and polar-to-rectangular conversions expect the angle argument to be in radians.)

In Radian angle mode:

`cos(45°)` [ENTER] $\frac{\sqrt{2}}{2}$

`cos({0,π/4,90°,30.12°})` [◆] [ENTER]
`{1 .707... 0 .864...}`

∠ (angle) [2nd] [∠] **key**

[*radius,∠θ_angle*] ⇒ *vector* (polar input)
[*radius,∠θ_angle,Z_coordinate*] ⇒ *vector* (cylindrical input)
[*radius,∠θ_angle,∠φ_angle*] ⇒ *vector* (spherical input)

Returns coordinates as a vector depending on the Vector Format mode setting: rectangular, cylindrical, or spherical.

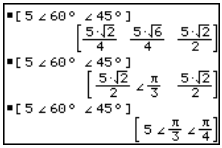
`[5,∠60°,∠45°]` [ENTER]

In Radian mode and vector format set to:

rectangular

cylindrical

spherical



(*magnitude ∠ angle*) ⇒ *complexValue* (polar input)

Enters a complex value in (r∠θ) polar form. The *angle* is interpreted according to the current Angle mode setting.

In Radian angle mode and Rectangular complex format mode:

`5+3i–(10∠π/4)` [ENTER] $5-5\cdot\sqrt{2}+(3-5\cdot\sqrt{2})\cdot i$

[◆][ENTER] ⁻2.071…–4.071…·*i*

°, ', " [2nd] [°] **key** (°), [2nd] [′] **key** ('), [2nd] ["] **key** (")

dd° *mm*' *ss.ss*" ⇒ *expression*

dd A positive or negative number
mm A non-negative number
ss.ss A non-negative number

Returns *dd*+(*mm*/60)+(*ss.ss*/3600).

This base-60 entry format lets you:

- Enter an angle in degrees/minutes/seconds without regard to the current angle mode.
- Enter time as hours/minutes/seconds.

In Degree angle mode:

`25°13'17.5"` [ENTER] 25.221…

`25°30'` [ENTER] 51/2

' (prime) [2nd] [′] **key**

variable'
variable''

Enters a prime symbol in a differential equation. A single prime symbol denotes a 1st-order differential equation, two prime symbols denote a 2nd-order, etc.

`deSolve(y''=y^(⁻1/2) and y(0)=0 and y'(0)=0,t,y)` [ENTER]

$\frac{2\cdot y^{3/4}}{3}=t$

_ (underscore) [◆] [_] **key** [2nd] [_] **key**

expression_unit

Designates the units for an *expression*. All unit names must begin with an underscore.

You can use pre-defined units or create your own units. For a list of pre-defined units, refer to the module about constants and measurement units. You can press:
[2nd] [UNITS]
[◆] [UNITS]
to select units from a menu, or you can type the unit names directly.

`3_m▸_ft` [ENTER] 9.842…·_ft

Note: To type ▸, press [2nd] [▸].

*variable_*

When *variable* has no value, it is treated as though it represents a complex number. By default, without the _, the variable is treated as real.

If *variable* has a value, the _ is ignored and *variable* retains its original data type.

Note: You can store a complex number to a variable without using _. However, for best results in calculations such as **cSolve()** and **cZeros()**, the _ is recommended.

Assuming z is undefined:

```
real(z) [ENTER]          z
real(z_) [ENTER]         real(z_)

imag(z) [ENTER]          0
imag(z_) [ENTER]         imag(z_)
```

▶ (convert) [2nd] [▶] **key**

expression_unit1 ▶ *_unit2* ⇒ *expression_unit2*

Converts an expression from one unit to another. The units must be in the same category.

The _ underscore character designates the units. For a list of valid pre-defined units, refer to the module about constants and measurement units. You can press:
[2nd] [UNITS]
[◆] [UNITS] to select units from a menu, or you can type the unit names directly.

To get the _ underscore when typing units directly, press:
[◆] [_]
[2nd] [_]

Note: The ▶ conversion operator does not handle temperature units. Use **tmpCnv()** and **∆tmpCnv()** instead.

```
3_m▶_ft [ENTER]          9.842…·_ft
```

10^() CATALOG

10^ (*expression1***)** ⇒ *expression*
10^ (*list1***)** ⇒ *list*

Returns 10 raised to the power of the argument.

For a list, returns 10 raised to the power of the elements in *list1*.

```
10^(1.5) [ENTER]         31.622…
10^{0,⁻2,2,a} [ENTER]
```

$$\{1 \quad \frac{1}{100} \quad 100 \quad 10^{a}\}$$

10^(*squareMatrix1***)** ⇒ *squareMatrix*

Returns 10 raised to the power of *squareMatrix1*. This is *not* the same as calculating 10 raised to the power of each element. For information about the calculation method, refer to **cos()**.

squareMatrix1 must be diagonalizable. The result always contains floating-point numbers.

```
10^([1,5,3;4,2,1;6,-2,1]) [ENTER]
```

$$\begin{bmatrix} 1.143\ldots E7 & 8.171\ldots E6 & 6.675\ldots E6 \\ 9.956\ldots E6 & 7.115\ldots E6 & 5.813\ldots E6 \\ 7.652\ldots E6 & 5.469\ldots E6 & 4.468\ldots E6 \end{bmatrix}$$

x^{-1} CATALOG (^-1)

expression1 $\mathbf{x}^{-1}$ ⇒ *expression*
list1 $\mathbf{x}^{-1}$ ⇒ *list*

Returns the reciprocal of the argument.

For a list, returns the reciprocals of the elements in *list1*.

```
3.1^⁻1 [ENTER]           .322581
{a,4,⁻.1,x-2}^⁻1 [ENTER]
```

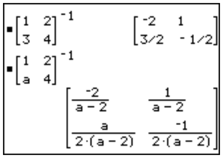
$$\{\frac{1}{a} \quad \frac{1}{4} \quad {}^{-}10. \quad \frac{1}{x-2}\}$$

squareMatrix1 **x**$^{-1}$ ⇒ *squareMatrix*

Returns the inverse of *squareMatrix1*.

squareMatrix1 must be a non-singular square matrix.

```
[1,2;3,4]^-1 [ENTER]
[1,2;a,4]^-1 [ENTER]
```



| ("with") [|] key [2nd] [|] key

expression **|** *Boolean expression1* [*and Boolean expression2*]...[*and Boolean expressionN*]

The "with" (|) symbol serves as a binary operator. The operand to the left of | is an expression. The operand to the right of | specifies one or more relations that are intended to affect the simplification of the expression. Multiple relations after | must be joined by a logical "and".

The "with" operator provides three basic types of functionality: substitutions, interval constraints, and exclusions.

```
x+1| x=3 [ENTER]                4
x+y| x=sin(y) [ENTER]    sin(y) + y
x+y| sin(y)=x [ENTER]         x + y
```

Substitutions are in the form of an equality, such as x=3 or y=sin(x). To be most effective, the left side should be a simple variable. *expression* | *variable* = *value* will substitute *value* for every occurrence of *variable* in *expression*.

```
x^3-2x+7→f(x) [ENTER]          Done
f(x)| x=√(3) [ENTER]         √3 + 7
(sin(x))^2+2sin(x)-6| sin(x)=d
[ENTER]
                           d²+2d-6
```

Interval constraints take the form of one or more inequalities joined by logical "and" operators. Interval constraints also permit simplification that otherwise might be invalid or not computable.

```
solve(x^2-1=0,x)|x>0 and x<2
[ENTER]
                              x = 1
√(x)*√(1/x)|x>0 [ENTER]           1
√(x)*√(1/x) [ENTER]      √(1/x) · √x
```

Exclusions use the "not equals" (/= or ≠) relational operator to exclude a specific value from consideration. They are used primarily to exclude an exact solution when using **cSolve()**, **cZeros()**, **fMax()**, **fMin()**, **solve()**, **zeros()**, etc.

```
solve(x^2-1=0,x)| x≠1 [ENTER]x = -1
```

→ (store)	STO▸ **key**	

expression → *var*
list → *var*
matrix → *var*
expression → *fun_name(parameter1,...)*
list → *fun_name(parameter1,...)*
matrix → *fun_name(parameter1,...)*

If variable *var* does not exist, creates *var* and initializes it to *expression*, *list*, or *matrix*.

If *var* already exists and if it is not locked or protected, replaces its contents with *expression*, *list*, or *matrix*.

Hint: If you plan to do symbolic computations using undefined variables, avoid storing anything into commonly used, one-letter variables such as a, b, c, x, y, z, etc.

```
π/4→myvar [ENTER]                    π/4
2cos(x)→Y1(x) [ENTER]               Done
{1,2,3,4}→Lst5 [ENTER]         {1 2 3 4}
[1,2,3;4,5,6]→MatG [ENTER]      [1 2 3]
                                [4 5 6]
"Hello"→str1 [ENTER]             "Hello"
```

© (comment)	**Program Editor/Control menu** or [TI-89] ◆) **key** [Voyage 200] 2nd X **key**

© **[***text***]**

© processes *text* as a comment line, which can be used to annotate program instructions.

© can be at the beginning or anywhere in the line. Everything to the right of ©, to the end of the line, is the comment.

Program segment:

```
  :
:© Get 10 points from the Graph
  screen
:For i,1,10 © This loops 10
times
  :
```

0b, 0h	[TI-89] 0 alpha [B] **keys**	[Voyage 200] 0 B **keys**
	[TI-89] 0 alpha [H] **keys**	[Voyage 200] 0 H **keys**

0b *binaryNumber*
0h *hexadecimalNumber*

Denotes a binary or hexadecimal number, respectively. To enter a binary or hex number, you must enter the 0b or 0h prefix regardless of the Base mode. Without a prefix, a number is treated as decimal (base 10).

Results are displayed according to the Base mode.

In Dec base mode:

```
0b10+0hF+10 [ENTER]          27
```

In Bin base mode:

```
0b10+0hF+10 [ENTER]     0b11011
```

In Hex base mode:

```
0b10+0hF+10 [ENTER]        0h1B
```

Appendix B: Technical Reference

This section contains a comprehensive list of TI-89 Titanium / Voyage™ 200 error messages and character codes. It also includes information about how certain TI-89 Titanium / Voyage™ 200 operations are calculated.

TI-89 Titanium / Voyage™ 200 Error Messages

This section lists error messages that may be displayed when input or internal errors are encountered. The number to the left of each error message represents an internal error number that is not displayed. If the error occurs inside a Try...EndTry block, the error number is stored in system variable *errornum*. Many of the error messages are self-explanatory and do not require descriptive information. However, additional information has been added for some error messages.

Error Number	Description
10	**A function did not return a value**
20	**A test did not resolve to TRUE or FALSE** Generally, undefined variables cannot be compared. For example, the test If a<b will cause this error if either a or b is undefined when the If statement is executed.
30	**Argument cannot be a folder name**
40	**Argument error**
50	**Argument mismatch** Two or more arguments must be of the same type. For example, **PtOn** *expression1,expression2* and **PtOn** *list1,list2* are both valid, but **PtOn** *expression,list* is a mismatch.
60	**Argument must be a Boolean expression or integer**
70	**Argument must be a decimal number**
80	**Argument must be a label name**
90	**Argument must be a list**
100	**Argument must be a matrix**
110	**Argument must be a Pic**
120	**Argument must be a Pic or string**
130	**Argument must be a string**
140	**Argument must be a variable name** For example, DelVar 12 is invalid because a number cannot be a variable name.
150	**Argument must be an empty folder name**

Error Number	Description
160	**Argument must be an expression** For example, zeros(2x+3=0,x) is invalid because the first argument is an equation.
161	**ASAP or Exec string too long**
163	**Attribute (8-digit number) of object (8-digit number) not found**
165	**Batteries too low for sending or receiving** Install new batteries before sending or receiving.
170	**Bound** For the interactive graph math functions like 2:Zero, the lower bound must be less than the upper bound to define the search interval.
180	**Break** The [ON] key was pressed during a long calculation or during program execution.
185	**Checksum error**
190	**Circular definition** This message is displayed to avoid running out of memory during infinite replacement of variable values during simplification. For example, a+1→a, where a is an undefined variable, will cause this error.
200	**Constraint expression invalid** For example, solve(3x^2−4=0, x) | x<0 or x>5 would produce this error message because the constraint is separated by “or” and not “and.”
205	**Data is too big to save to a variable. Please use F6 Util to reduce the size.** The size of the data in the editor exceeds the maximum size that can be saved in a variable. The F6 Util menu provides operations that can be used to reduce the size of the data.
210	**Data type** An argument is of the wrong data type.
220	**Dependent limit** A limit of integration is dependent on the integration variable. For example, ∫(x^2,x,1,x) is not allowed.
225	**Diff Eq setup**
230	**Dimension** A list or matrix index is not valid. For example, if the list {1,2,3,4} is stored in L1, then L1[5] is a dimension error because L1 only contains four elements.

Error Number	Description
240	**Dimension mismatch** Two or more arguments must be of the same dimension. For example, [1,2]+[1,2,3] is a dimension mismatch because the matrices contain a different number of elements.
250	**Divide by zero**
260	**Domain error** An argument must be in a specified domain. For example, ans(100) is not valid because the argument for **ans()** must be in the range 1–99.
270	**Duplicate variable name**
280	**Else and ElseIf invalid outside of If..EndIf block**
290	**EndTry is missing the matching Else statement**
295	**Excessive iteration**
300	**Expected 2 or 3-element list or matrix**
307	**Flash application extension (function or program) not found**
308	**Flash application not found**
310	**First argument of nSolve must be a univariate equation** The first argument must be an equation, and the equation cannot contain a non-valued variable other than the variable of interest. For example, nSolve(3x^2−4=0, x) is a valid equation; however, nSolve(3x^2−4, x) is not an equation, and nSolve(3x^2−y=0,x) is not a univariate equation because y has no value in this example.
320	**First argument of solve or cSolve must be an equation or inequality** For example, solve(3x^2−4, x) is invalid because the first argument is not an equation.
330	**Folder** An attempt was made in the VAR-LINK menu to store a variable in a folder that does not exist.
335	**Graph functions y1(x)...y99(x) not available in Diff Equations mode**
345	**Inconsistent units**
350	**Index out of range**
360	**Indirection string is not a valid variable name**
380	**Invalid ans()**
390	**Invalid assignment**

Error Number	Description
400	**Invalid assignment value**
405	**Invalid axes**
410	**Invalid command**
420	**Invalid folder name**
430	**Invalid for the current mode settings**
440	**Invalid implied multiply** For example, x(x+1) is invalid; whereas, x∗(x+1) is the correct syntax. This is to avoid confusion between implied multiplication and function calls.
450	**Invalid in a function or current expression** Only certain commands are valid in a user-defined function. Entries that are made in the Window Editor, Table Editor, Data/Matrix Editor, and Solver as well as system prompts such as Lower Bound cannot contain any commands or a colon (:). See also “Creating and Evaluating User-Defined Functions” in the *Calculator Home Screen* module.
460	**Invalid in Custom..EndCustm block**
470	**Invalid in Dialog..EndDlog block**
480	**Invalid in Toolbar..EndTBar block**
490	**Invalid in Try..EndTry block**
500	**Invalid label** Label names must follow the same rules used for naming variables.
510	**Invalid list or matrix** For example, a list inside a list such as {2,{3,4}} is not valid.
520	**Invalid outside Custom..EndCustm or ToolBar..EndTbar blocks** For example, an **Item** command is attempted outside a **Custom** or **ToolBar** structure.
530	**Invalid outside Dialog..EndDlog, Custom..EndCustm, or ToolBar..EndTBar blocks** For example, a Title command is attempted outside a **Dialog**, **Custom**, or **ToolBar** structure.
540	**Invalid outside Dialog..EndDlog block** For example, the **DropDown** command is attempted outside a **Dialog** structure.
550	**Invalid outside function or program** A number of commands are not valid outside a program or a function. For example, Local cannot be used unless it is in a program or function.

Error Number	Description
560	**Invalid outside Loop..EndLoop, For..EndFor, or While..EndWhile blocks** For example, the **Exit** command is valid only inside these loop blocks.
570	**Invalid pathname** For example, \\var is invalid.
575	**Invalid polar complex**
580	**Invalid program reference** Programs cannot be referenced within functions or expressions such as 1+p(x) where p is a program.
585	**Invalid relocation data in ASM program** The necessary relocation data in the ASM (Assembly) program is missing or corrupted.
590	**Invalid syntax block** A **Dialog..EndDlog** block is empty or has more than one title. A **Custom..EndCustm** block cannot contain PIC variables, and items must be preceded by a title. A **Toolbar..EndTBar** block must have a second argument if no items follow; or items must have a second argument and must be preceded by a title.
600	**Invalid table**
605	**Invalid use of units**
610	**Invalid variable name in a Local statement**
620	**Invalid variable or function name**
630	**Invalid variable reference**
640	**Invalid vector syntax**
650	**Link transmission** A transmission between two units was not completed. Verify that the connecting cable is connected firmly to both units.
665	**Matrix not diagonalizable**
670 673	**Memory** The calculation required more memory than was available at that time. If you get this error when you run a large program, you may need to break the program into separate, smaller programs or functions (where one program or function calls another).
680	**Missing (**

Error Number	Description
690	**Missing)**
700	**Missing "**
710	**Missing]**
720	**Missing }**
730	**Missing start or end of block syntax**
740	**Missing Then in the If..EndIf block**
750	**Name is not a function or program**
765	**No functions selected**
780	**No solution found** Using the interactive math features (F5:Math) in the Graph application can give this error. For example, if you attempt to find an inflection point of the parabola y1(x)=x^2 , which does not exist, this error will be displayed.
790	**Non-algebraic variable in expression** If a is the name of a PIC, GDB, MAC, FIG, etc., a+1 is invalid. Use a different variable name in the expression or delete the variable.
800	**Non-real result** For example, if the unit is in the REAL setting of the Complex Format mode, ln(⁻2) is invalid.
810	**Not enough memory to save current variable. Please delete unneeded variables on the Var-Link screen and re-open editor as current OR re-open editor and use F1 8 to clear editor.** This error message is caused by very low memory conditions inside the Data/Matrix Editor.
830	**Overflow**
840	**Plot setup**

Error Number	Description
850	**Program not found** A program reference inside another program could not be found in the provided path during execution.
860	**Recursion is limited to 255 calls deep**
	Rand type functions not allowed in graphing
870	**Reserved name or system variable**
875	**ROM-resident routine not available**
880	**Sequence setup**
885	**Signature error**
890	**Singular matrix**
895	**Slope fields need one selected function and are used for 1st-order equations only**
900	**Stat**
910	**Syntax** The structure of the entry is incorrect. For example, x+− y (x plus minus y) is invalid; whereas, x+ ⁻ y (x plus negative y) is correct.
930	**Too few arguments** The expression or equation is missing one or more arguments. For example, *d*(f(x)) is invalid; whereas, *d*(f(x),x) is the correct syntax.
940	**Too many arguments** The expression or equation contains an excessive number of arguments and cannot be evaluated.
950	**Too many subscripts**
955	**Too many undefined variables**
960	**Undefined variable**
965	**Unlicensed OS or Flash application**
970	**Variable in use so references or changes are not allowed**
980	**Variable is locked, protected, or archived**
990	**Variable name is limited to 8 characters**
1000	**Window variables domain**
1010	**Zoom**

Error Number	Description
	Warning: ∞^0 or undef^0 replaced by 1
	Warning: 0^0 replaced by 1
	Warning: 1^∞ or 1^undef replaced by 1
	Warning: cSolve may specify more zeros
	Warning: May produce false equation
	Warning: Expected finite real integrand
	Warning: May not be fully simplified
	Warning: More solutions may exist
	Warning: May introduce false solutions
	Warning: Operation may lose solutions
	Warning: Requires & returns 32 bit value
	Warning: Overflow replaced by ∞ or ⁻∞
	Warning: Questionable accuracy
	Warning: Questionable solution
	Warning: Solve may specify more zeros
	Warning: Trig argument too big to reduce
	Warning: Domain may be larger
	Warning: Domain may be smaller

TI-89 Titanium / Voyage™ 200 Modes

This section describes the modes of the TI-89 Titanium / Voyage™ 200 and lists the possible settings of each mode. These mode settings are displayed when you press [MODE].

Graph

Specifies the type of graphs you can plot.

1:FUNCTION	y(x) functions
2:PARAMETRIC	x(t) and y(t) parametric equations
3:POLAR	r(θ) polar equations
4:SEQUENCE	u(n) sequences
5:3D	z(x,y) 3D equations
6:DIFF EQUATIONS	y'(t) differential equations

Note: If you use a split screen with Number of Graphs = 2, Graph is for the top or left part of the screen and Graph 2 is for the bottom or right part.

Current Folder

Specifies the current folder. You can set up multiple folders with unique configurations of variables, graph databases, programs, etc.

***Note:** For detailed information about using folders, see Calculator Home Screen.*

1:main	Default folder included with the TI-89 Titanium / Voyage™ 200.
2: — (custom folders)	Other folders are available only if they have been created by a user.

Display Digits

Selects the number of digits. These decimal settings affect only how results are displayed—you can enter a number in any format.

Internally, the TI-89 Titanium / Voyage™ 200 retains decimal numbers with 14 significant digits. For display purposes, such numbers are rounded to a maximum of 12 significant digits.

1:FIX 0 2:FIX 1 … D:FIX 12	Results are always displayed with the selected number of decimal places.
E:FLOAT	The number of decimal places varies, depending on the result.
F:FLOAT 1 G:FLOAT 2 … Q:FLOAT 12	If the integer part has more than the selected number of digits, the result is rounded and displayed in scientific notation. For example, in FLOAT 4: 12345. is shown as 1.235E4

Angle

Specifies the units in which angle values are interpreted and displayed in trig functions and polar/rectangular conversions.

1:RADIAN
2:DEGREE
3:GRADIAN

Exponential Format

Specifies which notation format should be used. These formats affect only how an answer is displayed; you can enter a number in any format. Numeric answers can be displayed with up to 12 digits and a 3-digit exponent.

1:NORMAL	Expresses numbers in standard format. For example, 12345.67
2:SCIENTIFIC	Expresses numbers in two parts: • The significant digits display with one digit to the left of the decimal. • The power of 10 displays to the right of E. For example, 1.234567E4 means 1.234567×10^4
3:ENGINEERING	Similar to scientific notation. However: • The number may have one, two, or three digits before the decimal. • The power-of-10 exponent is a multiple of three. For example, 12.34567E3 means 12.34567×10^3

Note: If you select NORMAL, but the answer cannot be displayed in the number of digits selected by Display Digits, the TI-89 Titanium / Voyage™ 200 displays the answer in SCIENTIFIC notation. If Display Digits = FLOAT, scientific notation will be used for exponents of 12 or more and exponents of −4 or less.

Complex Format

Specifies whether complex results are displayed and, if so, their format.

1:REAL	Does not display complex results. (If a result is a complex number and the input does not contain the complex unit ***i***, an error message is displayed.)
2:RECTANGULAR	Displays complex numbers in the form: $a+b\boldsymbol{i}$
3:POLAR	Displays complex numbers in the form: $re^{i\theta}$

Vector Format

Determines how 2-element and 3-element vectors are displayed. You can enter vectors in any of the coordinate systems.

1:RECTANGULAR	Coordinates are in terms of x, y, and z. For example, [3,5,2] represents x = 3, y = 5, and z = 2.
2:CYLINDRICAL	Coordinates are in terms of r, θ, and z. For example, [3,∠45,2] represents r = 3, θ = 45, and z = 2.
3:SPHERICAL	Coordinates are in terms of r, θ, and ϕ. For example, [3, ∠45, ∠90] represents r = 3, θ = 45, and ϕ = 90.

Pretty Print

Determines how results are displayed on the Home screen.

1:OFF	Results are displayed in a linear, one-dimensional form. For example, π^2, π/2, or √((x-3)/x)
2:ON	Results are displayed in conventional mathematical format. For example, π^2, $\frac{\pi}{2}$, or $\sqrt{\frac{x-3}{x}}$

Note: For a complete description of these settings, refer to "Formats of Displayed Results" in the *Operating the Calculator* module.

Split Screen

Lets you split the screen into two parts. For example, you can display a graph and see the Y= Editor at the same time.

1:FULL	The screen is not split.
2:TOP-BOTTOM	The applications are shown in two screens that are above and below each other.
3:LEFT-RIGHT	The applications are shown in two screens that are to the left and right of each other.

To determine what and how information is displayed on a split screen, use this mode in conjunction with other modes such as Split 1 App, Split 2 App, Number of Graphs, and Split Screen Ratio. (Split Screen Ratio is available on the Voyage™ 200 only.)

Split 1 App and Split 2 App

Specifies which application is displayed on the screen.

- For a full screen, only Split 1 App is active.
- For a split screen, Split 1 App is the top or left part of the screen and Split 2 App is the bottom or right part.

The available application choices are those listed when you press ⓑ from the Page 2 mode screen or when you press [APPS]. You must have different applications in each screen unless you are in 2-graph mode.

Number of Graphs

Specifies whether both parts of a split screen can display graphs at the same time.

1	Only one part can display graphs.
2	Both parts can display an independent graph screen (Graph or Graph 2 setting) with independent settings.

Graph 2

Specifies the type of graphs that you can plot for the second graph on a two-graph split screen. This is active only when Number of Graphs = 2. In this two-graph setting, Graph sets the type of graph for the top or left part of the split screen, and Graph 2 sets the bottom or right part. The available choices are the same as for Graph.

Split Screen Ratio (Voyage™ 200 only)

Specifies the proportional sizes of the two parts of a split screen.

1:1	The screen is split evenly.
1:2	The bottom or right part is approximately twice the size of the top or left part.
2:1	The top or left part is approximately twice the size of the bottom or right part.

Exact/Approx

Specifies how fractional and symbolic expressions are calculated and displayed. By retaining rational and symbolic forms in the EXACT setting, the TI-89 Titanium / Voyage™ 200 increases precision by eliminating most numeric rounding errors.

1:AUTO	Uses EXACT setting in most cases. However, uses APPROXIMATE if the entry contains a decimal point.
2:EXACT	Displays non-whole-number results in their rational or symbolic form.
3:APPROXIMATE	Displays numeric results in floating-point form.

Note: For a complete description of these settings, refer to “Formats of Displayed Results” in the *Operating the Calculator* module.

Base

Lets you perform calculations by entering numbers in decimal, binary, or hexadecimal form.

1:DEC	Decimal numbers use 0 - 9 in the base 10 format
2:HEX	Hexadecimal numbers use 0 - 9 and A - F in the base 16 format.
3:BIN	Binary numbers use 0 and 1 in the base 2 format.

Unit System

Lets you enter a unit for values in an expression, such as 6_m * 4_m or 23_m/_s * 10_s, convert values from one unit to another within the same category, and create your own user-defined units.

1:SI	Select SI for the metric system of measurements
2:ENG/US	Select ENG/US for the non-metric system of measurements
3:CUSTOM	Allows you to select custom defaults.

Custom Units

Lets you select custom defaults. This mode is dimmed until you select Unit System, 3:CUSTOM.

Language

Lets you localize the TI-89 Titanium / Voyage™ 200 into one of several languages, depending on which language Flash applications are installed.

1:English	Default language included with the TI-89 Titanium / Voyage™ 200 operating system (OS).
2: — (language Flash applications)	Alternate languages are available only if the respective language Flash applications have been installed.

Apps Desktop

Lets you turn the display of the Apps desktop on or off.

ON	Displays the navigable Apps desktop. The Apps desktop appears when you: • Press [APPS]. • Turn the unit on after it has been turned off by pressing [2nd] [OFF]. • Press [2nd] [QUIT] from an App that is displayed in full screen mode.
OFF	Does not display the navigable Apps desktop. The unit defaults to the calculator Home screen. The calculator Home screen displays when you press [2nd] [QUIT]. The APPLICATIONS menu displays when you press [APPS].

TI-89 Titanium / Voyage™ 200 Character Codes

The **char()** function lets you refer to any character by its numeric character code. For example, to display ◆ on the Program I/O screen, use `Disp char(127)`. You can use **ord()** to find the numeric code of a character. For example, `ord("A")` returns 65.

1.	SOH	38.	&	75.	K	112.	p	149.	ᴇ	186.	º	223.	ß
2.	STX	39.	'	76.	L	113.	q	150.	*e*	187.	»	224.	à
3.	ETX	40.	(	77.	M	114.	r	151.	***i***	188.	*d*	225.	á
4.	EOT	41.	)	78.	N	115.	s	152.	^{r}	189.	∫	226.	â
5.	ENQ	42.	*	79.	O	116.	t	153.	ᴛ	190.	∞	227.	ã
6.	ACK	43.	+	80.	P	117.	u	154.	x̄	191.	¿	228.	ä
7.	BELL	44.	,	81.	Q	118.	v	155.	ȳ	192.	À	229.	å
8.	BS	45.	−	82.	R	119.	w	156.	≤	193.	Á	230.	æ
9.	TAB	46.	.	83.	S	120.	x	157.	≠	194.	Â	231.	ç
10.	LF	47.	/	84.	T	121.	y	158.	≥	195.	Ã	232.	è
11.	✎	48.	0	85.	U	122.	z	159.	∠	196.	Ä	233.	é
12.	FF	49.	1	86.	V	123.	{	160.	…	197.	Å	234.	ê
13.	CR	50.	2	87.	W	124.	|	161.	¡	198.	Æ	235.	ë
14.	🔒	51.	3	88.	X	125.	}	162.	¢	199.	Ç	236.	ì
15.	✓	52.	4	89.	Y	126.	~	163.	£	200.	È	237.	í
16.	▪	53.	5	90.	Z	127.	◆	164.	¤	201.	É	238.	î
17.	◄	54.	6	91.	[	128.	α	165.	¥	202.	Ê	239.	ï
18.	►	55.	7	92.	\	129.	β	166.	¦	203.	Ë	240.	ð
19.	▲	56.	8	93.	]	130.	Γ	167.	§	204.	Ì	241.	ñ
20.	▼	57.	9	94.	^	131.	γ	168.	√	205.	Í	242.	ò
21.	←	58.	:	95.	_	132.	Δ	169.	©	206.	Î	243.	ó
22.	→	59.	;	96.	`	133.	δ	170.	ª	207.	Ï	244.	ô
23.	↑	60.	<	97.	a	134.	ε	171.	«	208.	Ð	245.	õ
24.	↓	61.	=	98.	b	135.	ζ	172.	¬	209.	Ñ	246.	ö
25.	◀	62.	>	99.	c	136.	θ	173.	-	210.	Ò	247.	÷
26.	▶	63.	?	100.	d	137.	λ	174.	®	211.	Ó	248.	ø
27.	⬆	64.	@	101.	e	138.	ξ	175.	¯	212.	Ô	249.	ù
28.	∪	65.	A	102.	f	139.	Π	176.	°	213.	Õ	250.	ú
29.	∩	66.	B	103.	g	140.	π	177.	±	214.	Ö	251.	û
30.	⊂	67.	C	104.	h	141.	ρ	178.	^{2}	215.	×	252.	ü
31.	∈	68.	D	105.	i	142.	Σ	179.	^{3}	216.	Ø	253.	ý
32.	SPACE	69.	E	106.	j	143.	σ	180.	$^{-1}$	217.	Ù	254.	þ
33.	!	70.	F	107.	k	144.	τ	181.	µ	218.	Ú	255.	ÿ
34.	"	71.	G	108.	l	145.	ϕ	182.	¶	219.	Û		
35.	#	72.	H	109.	m	146.	ψ	183.	·	220.	Ü		
36.	$	73.	I	110.	n	147.	Ω	184.	$^+$	221.	Ý		
37.	%	74.	J	111.	o	148.	ω	185.	^{1}	222.	Þ		

TI-89 Titanium Key Codes

The **getKey()** function returns a value that corresponds to the last key pressed, according to the tables shown in this section. For example, if your program contains a **getKey()** function, pressing [2nd] [F6] will return a value of 273.

Table 1: Key Codes for Primary Keys

Key	Modifier									
	None		[↑]		[2nd]		[◆]		[alpha]	
	Assoc.	Value	Assoc.	Value	Assoc.	Value	Assoc.	Value	Assoc.	Value
[F1]	F1	268	F1	268	F6	273	Y=	8460	F1	268
[F2]	F2	269	F2	269	F7	274	WINDOW	8461	F2	269
[F3]	F3	270	F3	270	F8	275	GRAPH	8462	F3	270
[F4]	F4	271	F4	271	F4	271	TblSet	8463	F4	271
[F5]	F5	272	F5	272	F5	272	TABLE	8464	F5	272
[◆]			COPY	24576	CUT	12288				
[alpha]					a-lock					
[ESC]	ESC	264	ESC	264	QUIT	4360	PASTE	8456	ESC	264
[APPS]	APPS	265	APPS	265	SWITCH	4361		8457	APPS	265
[HOME]	HOME	277	HOME	277	CUST	4373	HOME	277	HOME	277
[MODE]	MODE	266	MODE	266	▸	18	_	95	MODE	266
[CATALOG]	CATLG	278	CATLG	278	*i*	151	∞	190	CATLG	278
[←]	BS	257	BS	257	INS	4353	DEL	8449	BS	257
[CLEAR]	CLEAR	263	CLEAR	263	CLEAR	263		8455	CLEAR	263
[X]	x	120	X	88	LN	4184	e^x	8280	x	120
[Y]	y	121	Y	89	SIN	4185	SIN^{-1}	8281	y	121
[Z]	z	122	Z	90	COS	4186	COS^{-1}	8282	z	122
[T]	t	116	T	84	TAN	4180	TAN^{-1}	8276	t	116
[^]	^	94	^	94	π	140	θ	136	^	94
[|]	|	124	F	70	°	176	Format d/b	8316	f	102
[(]	(	40	B	66	{	123			b	98
[)]	)	41	C	67	}	125	◉	169	c	99
[,]	,	44	D	68	[	91		8236	d	100
[÷]	/	47	E	69	]	93	!	33	e	101
[×]	*	42	J	74	√	4138	&	38	j	106
[−]	-	45	O	79	VAR-LNK	4141	Contr. -		o	111
[+]	+	43	U	85	CHAR	4139	Contr. +		u	117

Table 1: Key Codes for Primary Keys (Continued)

Key	Modifier									
	None		[↑]		[2nd]		[◆]		[alpha]	
	Assoc.	**Value**	**Assoc.**	**Value**	**Assoc.**	**Value**	**Assoc.**	**Value**	**Assoc.**	**Value**
[ENTER]	CR	13	CR	13	ENTRY	4109	APPROX	8205	CR	13
[STO▸]	STO▸	258	P	80	RCL	4354	@	64	p	112
[=]	=	61	A	65	'	39	≠	157	a	97
[EE]	EE	149	K	75	∠	159	SYMB	8341	k	107
[(-)]	-	173	SPACE	32	ANS	4372		8365	SPACE	32
[.]	.	46	W	87	>	62	≥	158	w	119
[0]	0	48	V	86	<	60	≤	156	v	118
[1]	1	49	Q	81	"	34		8241	q	113
[2]	2	50	R	82	\	92		8242	r	114
[3]	3	51	S	83	UNITS	4147		8243	s	115
[4]	4	52	L	76	:	58		8244	l	108
[5]	5	53	M	77	MATH	4149		8245	m	109
[6]	6	54	N	78	MEM	4150		8246	n	110
[7]	7	55	G	71	∫	4151		8247	g	103
[8]	8	56	H	72	d	4152		8248	h	104
[9]	9	57	I	73	;	59		8249	i	105

Table 2: Arrow Keys (including diagonal movement)

Key	Normal	[↑]	[2nd]	[◆]	[alpha]
⊙ up	338	16722	4434	8530	33106
⊙ right	340	16724	4436	8532	33108
⊙ down	344	16728	4440	8536	33112
⊙ left	337	16721	4433	8529	33105
up and left	339	16723	4435	8531	33107
up and right	342	16726	4438	8534	33110
down and left	345	16729	4441	8537	33113
down and right	348	16732	4444	8540	33116

Table 3: Greek Letters (prefixed by [◆] [(])

Keys		Second modifier			
		[alpha]		[↑]	
		Assoc.	**Value**	**Assoc.**	**Value**
[=]	[A]	α	128		
[(]	[B]	β	129		
[,]	[D]	δ	133	Δ	132
[÷]	[E]	ε	134		
[I]	[F]	φ	145		
[7]	[G]	γ	131	Γ	130
[4]	[L]	λ	137		
[5]	[M]	μ	181		
[STO▸]	[P]	π	140	Π	139
[2]	[R]	ρ	141		
[3]	[S]	σ	143	Σ	142
[T]	[T]	τ	144		
[.]	[W]	ω	148	Ω	147
[X]		ξ	138		
[Y]		ψ	146		
[Z]		ζ	135		

Voyage™ 200 Key Codes

The **getKey()** function returns a value that corresponds to the last key pressed, according to the tables shown in this section. For example, if your program contains a **getKey()** function, pressing [2nd] [F1] will return a value of 268.

Table 1: Key Codes for Primary Keys

Key	Modifier							
	None		[↑]		[2nd]		[◆]	
	Assoc.	**Value**	**Assoc.**	**Value**	**Assoc.**	**Value**	**Assoc.**	**Value**
[F1]	F1	268	F1	268	F1	268		8460
[F2]	F2	269	F2	269	F2	269		8461
[F3]	F3	270	F3	270	F3	270		8462
[F4]	F4	271	F4	271	F4	271		8463
[F5]	F5	272	F5	272	F5	272		8464
[F6]	F6	273	F6	273	F6	273		8465
[F7]	F7	274	F7	274	F7	274		8466
[F8]	F8	275	F8	275	F8	275		8467
[MODE]	MODE	266	MODE	266	MODE	266		8458
[CLEAR]	CLEAR	263	CLEAR	263	CLEAR	263		8455
[LN]	LN	262	LN	262	e^x	4358		8454
[ESC]	ESC	264	ESC	264	QUIT	4360		8456
[APPS]	APPS	265	APPS	265	SWITCH	4361		8457
[ENTER]	CR	13	CR	13	ENTRY	4109	APPROX	8205
[SIN]	SIN	259	SIN	259	SIN^{-1}	4355		8451
[COS]	COS	260	COS	260	COS^{-1}	4356		8452
[TAN]	TAN	261	TAN	261	TAN^{-1}	4357		8453
[^]	^	94	^	94	π	140		8286
[(]	(	40	(	40	{	123		8232
[)]	)	41	)	41	}	125		8233
[,]	,	44	,	44	[	91		8236
[÷]	/	47	/	47	]	93		8239
[×]	*	42	*	42	√	4138		8234
[−]	-	45	-	45	VAR-LNK	4141	Contrast −	
[+]	+	43	+	43	CHAR	4139	Contrast +	
[STO▸]	STO▸	258	STO▸	258	RCL	4354		8450
SPACE		32		32		32		8224
[=]	=	61	=	61	\	92		8253
[←]	BS	257	BS	257	INS	4353	DEL	8449
[θ]	θ	136	θ	136	:	58		8328
[(-)]	-	173	-	173	ANS	4372		8365
[.]	.	46	.	46	>	62		8238

Table 1: Key Codes for Primary Keys (Continued)

Key	Modifier							
	None		↑		2nd		◆	
	Assoc.	Value	Assoc.	Value	Assoc.	Value	Assoc.	Value
0	0	48	0	48	<	60		8240
1	1	49	1	49	E	149		8241
2	2	50	2	50	CATALOG	4146		8242
3	3	51	3	51	CUST	4147		8243
4	4	52	4	52	Σ	4148		8244
5	5	53	5	53	MATH	4149		8245
6	6	54	6	54	MEM	4150		8246
7	7	55	7	55	∫	4151		8247
8	8	56	8	56	d	4152		8248
9	9	57	9	57	x^{-1}	4153		8249
A	a	97	A	65	Table 3			8257
B	b	98	B	66	‘	39		8258
C	c	99	C	67	Table 4		COPY	8259
D	d	100	D	68	°	176		8260
E	e	101	E	69	Table 5		WINDOW	8261
F	f	102	F	70	∠	159	FORMAT	8262
G	g	103	G	71	Table 6			8263
H	h	104	H	72	&	38		8264
I	i	105	I	73	i	151		8265
J		106	J	74	∞	190		8266
K	k	107	K	75	|	124	KEY	8267
L	l	108	L	76	“	34		8268
M	m	109	M	77	;	59		8269
N	n	110	N	78	Table 7		NEW	8270
O	o	111	O	79	Table 8		OPEN	8271
P	p	112	P	80	_	95	UNITS	8272
Q	q	113	Q	81	?	63	CALC HOME	8273
R	r	114	R	82	@	64	GRAPH	8274
S	s	115	S	83	β	223	SAVE	8275
T	t	116	T	84	#	35	TBLSET	8276
U	u	117	U	85	Table 9			8277
V	v	118	V	86	≠	157	PASTE	8278
W	w	119	W	87	!	33	Y=	8279
X	x	120	X	88	◉	169	CUT	8280
Y	y	121	Y	89	▸	18	TABLE	8281
Z	z	122	Z	90	CAPS			8282

Table 2: Arrow Keys (including diagonal movement)

Key	Normal	⇧	[2nd]	◆	✋
↑	338	16722	4434	8530	33106
→	340	16724	4436	8532	33108
↓	344	16728	4440	8536	33112
←	337	16721	4433	8529	33105
↑ and ←	339	16723	4435	8531	33107
↑ and →	342	16726	4438	8534	33110
↓ and ←	345	16729	4441	8537	33113
↓ and →	348	16732	4444	8540	33116

Note: The Grab (✋) modifier only affects the arrow keys.

Table 3: Grave Accent Letters (prefixed by [2nd] A)

Key	Assoc.	Normal	⇧
A	à	224	192
E	è	232	200
I	ì	236	204
O	ò	242	210
U	ù	249	217

Table 4: Cedilla Letters (prefixed by [2nd] C)

Key	Assoc.	Normal	⇧
C	ç	231	199

Table 5: Acute Accent Letters (prefixed by [2nd] E)

Key	Assoc.	Normal	⇧
A	á	225	193
E	é	233	201
I	í	237	205
O	ó	243	211
U	ú	250	218
Y	ý	253	221

Table 6: Greek Letters (prefixed by [2nd] G)

Key	Assoc.	Normal	[↑]
A	α	128	
B	β	129	
D	δ	133	132
E	ε	134	
F	φ	145	
G	γ	131	130
L	λ	137	
M	μ	181	
P	π	140	139
R	ρ	141	
S	σ	143	142
T	τ	144	
W	ω	148	147
X	ξ	138	
Y	ψ	146	
Z	ζ	135	

Table 7: Tilde Letters (prefixed by [2nd] N)

Key	Assoc.	Normal	[↑]
N	ñ	241	209
O	õ	245	

Table 8: Caret Letters (prefixed by [2nd] O)

Key	Assoc.	Normal	[↑]
A	â	226	194
E	ê	234	202
I	î	238	206
O	ô	244	212
U	û	251	219

Table 9: Umlaut Letters (prefixed by [2nd] U)

Key	Assoc.	Normal	[↑]
A	ä	228	196
E	ë	235	203
I	ï	239	207
O	ö	246	214
U	ü	252	220
Y	ÿ	255	

Entering Complex Numbers

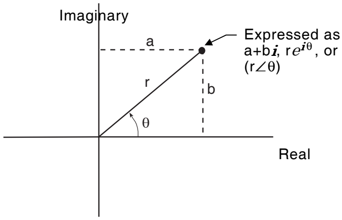
You can enter complex numbers in the polar form (r∠θ), where r is the magnitude and θ is the angle, or polar form $re^{i\theta}$. You can also enter complex numbers in rectangular form a+bi

Overview of Complex Numbers

A complex number has real and imaginary components that identify a point in the complex plane. These components are measured along the real and imaginary axes, which are similar to the x and y axes in the real plane.

The point can be expressed in rectangular form or in either of two polar forms.

The i symbol represents the imaginary number $\sqrt{-1}$.

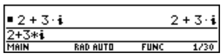


As shown below, the form that you can enter depends on the current Angle mode.

You can use the form:	When the Angle mode setting is:
a+bi	Radian, Degree or Gradian
r$e^{i\theta}$	Radian only (In Degree or Gradian angle mode, this form causes a Domain error.)
(r∠θ)	Radian, Degree or Gradian

Use the following methods to enter a complex number.

Note: *To get the i symbol, press [2nd] [i], do not simply type an alphabetic i.*

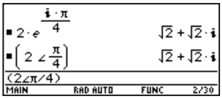
To enter the:	Do this:
Rectangular form a+bi	Substitute the applicable values or variable names for a and b. a [+] b [2nd] [i] For example: 

Important: Do not use the $re^{i\theta}$ polar form in Degree angle mode. It will cause a Domain error.

Note: To get the *e* symbol, press:
TI-89 Titanium: [◆] [e^x].
Voyage™ 200: [2nd] [e^x]
Do not simply type an alphabetic e.

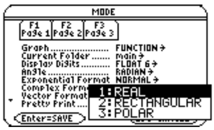
Tip: To get the ∠ symbol, press [2nd] [∠].

Tip: To enter θ in degrees for (r∠θ), you can type a ° symbol (such as 45°). To get the ° symbol, press [2nd] [°]. You should not use degrees or Gradian for $re^{i\theta}$.

To enter the:	Do this:
Polar form $re^{i\theta}$ – or – (r∠θ) Parentheses are required for the (r∠θ) form.	Substitute the applicable values or variable names for r and θ, where θ is interpreted according to the Angle mode setting. **TI-89 Titanium:** [alpha] [R] [◆] [e^x] [2nd] [i] [◆] [θ] [)] – or – [(] [alpha] [R] [2nd] [∠] [◆] [θ] [)] **Voyage™ 200:** R [2nd] [e^x] [2nd] [i] [θ] [)] – or – [(] R [2nd] [∠] [θ] [)] For example:  Results are shown in rectangular form, but you can select polar form.

Complex Format Mode for Displaying Results

Use [MODE] to set the Complex Format mode to one of three settings.



You can enter a complex number at any time, regardless of the Complex Format mode setting. However, the mode setting determines how results are displayed.

Note: You can enter complex numbers in any form (or a mixture of all forms) depending on the Angle mode.

If Complex Format is:	The TI-89 Titanium / Voyage™ 200:
REAL	Will not display complex results unless you: • Enter a complex number. – or – • Use a complex function such as **cFactor()**, **cSolve()**, or **cZeros()**. If complex results are displayed, they will be shown in either a+bi or $re^{i\theta}$ form.
RECTANGULAR	Displays complex results as a+bi.
POLAR	Displays complex results as: • $re^{i\theta}$ if the Angle mode = Radian – or – • (r∠θ) if the Angle mode = Degree or Gradian

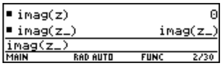
Using Complex Variables in Symbolic Calculations

Regardless of the Complex Format mode setting, variables that have no stored value and that do not end with an underscore (_) are treated as real numbers. To perform complex symbolic analysis, you can use either of the following methods to set up a complex variable.

Method 1: Use an underscore _ (**TI-89 Titanium:** [◆] [_] **Voyage™ 200** [2nd] [_]) as the last character in the variable name to designate a complex variable. For example:

Note:** For best results in calculations such as* **cSolve()** *and* **cZeros(), use Method 1.*

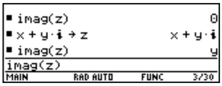
z_ is treated as a complex variable if it does not have a stored value.



Method 2: Store an unreal value into any variable. For example:

x+y*i*→z

Then z is treated as a complex variable.



Complex Numbers and Degree Mode

Radian angle mode is recommended for complex number calculations. Internally, the TI-89 Titanium / Voyage™ 200 converts all entered trig values to radians, but it does not convert values for exponential, logarithmic, or hyperbolic functions.

***Note:** If you use* Degree or Gradian *angle mode, you must make polar entries in the form (r∠θ). In* Degree or Gradian *angle mode, an* $re^{i\theta}$ *entry causes an error.*

In Degree and Gradian angle modes, complex identities such as *e*^(*i*θ) = cos(θ) + *i* sin(θ) are not generally true because the values for cos and sin are converted to radians, while those for *e*^( ) are not. For example, *e*^(*i45*) = cos(45) + *i* sin(45) is treated internally as *e*^(*i*45) = cos(π/4) + *i* sin(π/4). Complex identities are always true in Radian angle mode.

Accuracy Information

To maximize accuracy, the TI-89 Titanium / Voyage™ 200 carries more digits internally than it displays.

Computational Accuracy

Floating-point (decimal) values in memory are stored using up to 14 digits with a 3-digit exponent.

- For min and max Window variables (xmin, xmax, ymin, ymax, etc.), you can store values using up to 12 digits. Other Window variables use 14 digits.
- When a floating-point value is displayed, the displayed value is rounded as specified by the applicable mode settings (Display Digits, Exponential Format, etc.), with a maximum of 12 digits and a 3-digit exponent.
- RegEQ displays up to 14-digit coefficients.

Integer values in memory are stored using up to 614 digits.

Graphing Accuracy

Note: *For a table that lists the number of pixels in a full screen or split screen, refer to “Setting and Exiting the Split Screen Mode” in Split Screens.*

The Window variable xmin is the center of the leftmost pixel used, and xmax is the center of the rightmost pixel used. Δx is the distance between the centers of two horizontally adjacent pixels.

- Δx is calculated as (xmax − xmin) / (# of x pixels − 1).
- If Δx is entered from the Home screen or a program, xmax is calculated as xmin + Δx ∗ (# of x pixels − 1).

The Window variable ymin is the center of the bottom pixel used, and ymax is the center of the top pixel used. Δy is the distance between the centers of two vertically adjacent pixels.

- Δy is calculated as (ymax − ymin) / (# of y pixels − 1).
- If Δy is entered from the Home screen or a program, ymax is calculated as ymin + Δy ∗ (# of y pixels − 1).

Cursor coordinates are displayed as eight characters (which may include a negative sign, decimal point, and exponent). The coordinate values (xc, yc, zc, etc.) are updated with a maximum of 12-digit accuracy.

System Variables and Reserved Names

This section lists the names of system variables and reserved function names that are used by the TI-89 Titanium / Voyage™ 200. Only those system variables and reserved function names that are identified by an asterisk (*) can be deleted by using DelVar *var* on the entry line.

Graph	y1(x)–y99(x)*	y1'(t)–y99'(t)*	yi1–yi99*	r1(θ)–r99(θ)*
	xt1(t)–xt99(t)*	yt1(t)– yt99(t)*	z1(x,y)–z99(x,y)*	u1(n)–u99(n)*
	ui1–ui99*	xc	yc	zc
	tc	rc	θc	nc
	xfact	yfact	zfact	xmin
	xmax	xscl	xgrid	ymin
	ymax	yscl	ygrid	xres
	Δx	Δy	zmin	zmax
	zscl	eyeθ	eyeϕ	eyeψ
	ncontour	θmin	θmax	θstep
	tmin	tmax	tstep	t0
	tplot	ncurves	diftol	dtime
	Estep	fldpic	fldres	nmin
	nmax	plotStrt	plotStep	sysMath
Graph Zoom	zxmin	zxmax	zxscl	zxgrid
	zymin	zymax	zyscl	zygrid
	zxres	zθmin	zθmax	zθstep
	ztmin	ztmax	ztstep	zt0de
	ztmaxde	ztstepde	ztplotde	zzmin
	zzmax	zzscl	zeyeθ	zeyeϕ
	zeyeψ	znmin	znmax	zpltstrt
	zpltstep			
Statistics	$\bar{x}$	$\bar{y}$	Σx	σx
	Σx^2	Σxy	Σy	σy
	Σy^2	corr	maxX	maxY
	medStat	medx1	medx2	medx3
	medy1	medy2	medy3	minX
	minY	nStat	q1	q3
	regCoef*	regEq(x)*	seed1	seed2
	Sx	Sy	R^2	
Table	tblStart	Δtbl	tblInput	
Data/Matrix	c1–c99	sysData*		
Miscellaneous	main	ok	errornum	
Solver	eqn*	exp*		

EOS (Equation Operating System) Hierarchy

This section describes the Equation Operating System (EOS™) that is used by the TI-89 Titanium / Voyage™ 200. Numbers, variables, and functions are entered in a simple, straightforward sequence. EOS evaluates expressions and equations using parenthetical grouping and according to the priorities described below.

Order of Evaluation

Level	Operator
1	Parentheses ( ), brackets [ ], braces { }
2	Indirection (#)
3	Function calls
4	Post operators: degrees-minutes-seconds (° ,',"), factorial (!), percentage (%), radian (ʳ), subscript ([ ]), transpose (ᵀ)
5	Exponentiation, power operator (^)
6	Negation (⁻)
7	String concatenation (&)
8	Multiplication (∗), division (/)
9	Addition (+), subtraction (−)
10	Equality relations: equal (=), not equal (≠ or /=), less than (<), less than or equal (≤ or <=), greater than (>), greater than or equal (≥ or >=)
11	Logical **not**
12	Logical **and**
13	Logical **or**, exclusive logical **xor**
14	Constraint “with” operator (|)
15	Store (→)

Parentheses, Brackets, and Braces

All calculations inside a pair of parentheses, brackets, or braces are evaluated first. For example, in the expression 4(1+2), EOS first evaluates the portion of the expression inside the parentheses, 1+2, and then multiplies the result, 3, by 4.

The number of opening and closing parentheses, brackets, and braces must be the same within an expression or equation. If not, an error message is displayed that indicates the missing element. For example, (1+2)/(3+4 will display the error message “Missing).”

Note: Because the TI-89 Titanium / Voyage™ 200 allows you to define your own functions, a variable name followed by an expression in parentheses is considered a “function call” instead of implied multiplication. For example a(b+c) is the function a evaluated by b+c. To multiply the expression b+c by the variable a, use explicit multiplication: a∗(b+c).

Indirection

The indirection operator (#) converts a string to a variable or function name. For example, #(“x”&”y”&”z”) creates the variable name xyz. Indirection also allows the creation and modification of variables from inside a program. For example, if 10→r and “r”→s1, then #s1=10.

Post Operators

Post operators are operators that come directly after an argument, such as 5!, 25%, or 60° 15' 45". Arguments followed by a post operator are evaluated at the fourth priority level. For example, in the expression 4^3!, 3! is evaluated first. The result, 6, then becomes the exponent of 4 to yield 4096.

Exponentiation

Exponentiation (^) and element-by-element exponentiation (.^) are evaluated from right to left. For example, the expression 2^3^2 is evaluated the same as 2^(3^2) to produce 512. This is different from (2^3)^2, which is 64.

Negation

To enter a negative number, press [(-)] followed by the number. Post operations and exponentiation are performed before negation. For example, the result of $^{-}x^{2}$ is a negative number, and $^{-}9^{2} = {}^{-}81$. Use parentheses to square a negative number such as $({}^{-}9)^{2}$ to produce 81. Note also that negative 5 ($^{-}5$) is different from minus 5 (−5), and $^{-}3!$ evaluates as $^{-}(3!)$.

Constraint (|)

The argument following the “with” (|) operator provides a set of constraints that affect the evaluation of the argument preceding the “with” operator.

Regression Formulas

This section describes how the statistical regressions are calculated.

Least-Squares Algorithm

Most of the regressions use non-linear recursive least-squares techniques to optimize the following cost function, which is the sum of the squares of the residual errors:

$$J = \sum_{i=1}^{N} [residualExpression]^2$$

where: *residualExpression* is in terms of x_i and y_i
x_i is the independent variable list
y_i is the dependent variable list
N is the dimension of the lists

This technique attempts to recursively estimate the constants in the model expression to make *J* as small as possible.

For example, *y=a sin(bx+c)+d* is the model equation for **SinReg**. So its residual expression is:

$a\ sin(bx_i+c)+d-y_i$

For **SinReg**, therefore, the least-squares algorithm finds the constants *a*, *b*, *c*, and *d* that minimize the function:

$$J = \sum_{i=1}^{N} [a\ sin(bx_i + c) + d - y_i]^2$$

Regressions

Regression	Description
CubicReg	Uses the least-squares algorithm to fit the third-order polynomial: $y=ax^3+bx^2+cx+d$ For four data points, the equation is a polynomial fit; for five or more, it is a polynomial regression. At least four data points are required.
ExpReg	Uses the least-squares algorithm and transformed values *x* and ln(*y*) to fit the model equation: $y=ab^x$
LinReg	Uses the least-squares algorithm to fit the model equation: $y=ax+b$ where *a* is the slope and *b* is the y-intercept.

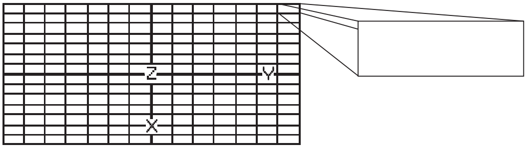
Regression	Description
LnReg	Uses the least-squares algorithm and transformed values ln(*x*) and *y* to fit the model equation: $y=a+b\ln(x)$
Logistic	Uses the least-squares algorithm to fit the model equation: $y=a/(1+b*e$^$(c*x))+d$
MedMed	Uses the median-median line (resistant line) technique to calculate summary points x1, y1, x2, y2, x3, and y3, and fits the model equation: $y=ax+b$ where *a* is the slope and *b* is the y-intercept.
PowerReg	Uses the least-squares algorithm and transformed values ln(*x*) and ln(*y*) to fit the model equation: $y=ax^b$
QuadReg	Uses the least-squares algorithm to fit the second-order polynomial: $y=ax^2+bx+c$ For three data points, the equation is a polynomial fit; for four or more, it is a polynomial regression. At least three data points are required.
QuartReg	Uses the least-squares algorithm to fit the fourth-order polynomial: $y=ax^4+bx^3+cx^2+dx+e$ For five data points, the equation is a polynomial fit; for six or more, it is a polynomial regression. At least five data points are required.
SinReg	Uses the least-squares algorithm to fit the model equation: $y=a\ sin(bx+c)+d$

Contour Levels and Implicit Plot Algorithm

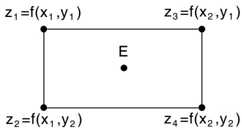
Contours are calculated and plotted by the following method. An implicit plot is the same as a contour, except that an implicit plot is for the z=0 contour only.

Algorithm

Based on your x and y Window variables, the distance between xmin and xmax and between ymin and ymax is divided into a number of grid lines specified by xgrid and ygrid. These grid lines intersect to form a series of rectangles.



For each rectangle, the equation is evaluated at each of the four corners (also called vertices or grid points) and an average value (E) is calculated:

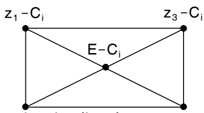


$$E = \frac{z_1 + z_2 + z_3 + z_4}{4}$$

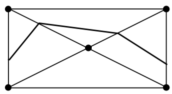
The E value is treated as the value of the equation at the center of the rectangle.

For each specified contour value (C_i):

- At each of the five points shown to the right, the difference between the point's z value and the contour value is calculated.



- A sign change between any two adjacent points implies that a contour crosses the line that joins those two points. Linear interpolation is used to approximate where the zero crosses the line.
- Within the rectangle, any zero crossings are connected with straight lines.
- This process is repeated for each contour value.



Each rectangle in the grid is treated similarly.

Runge-Kutta Method

For Runge-Kutta integrations of ordinary differential equations, the TI-89 Titanium / Voyage™ 200 uses the Bogacki-Shampine 3(2) formula as found in the journal *Applied Math Letters*, 2 (1989), pp. 1–9.

Bogacki-Shampine 3(2) Formula

The Bogacki-Shampine 3(2) formula provides a result of 3rd-order accuracy and an error estimate based on an embedded 2nd-order formula. For a problem of the form:

$$y' = f(x, y)$$

and a given step size *h*, the Bogacki-Shampine formula can be written:

$$F_1 = f(x_n, y_n)$$

$$F_2 = f\left(x_n + h\frac{1}{2}, y_n + h\frac{1}{2}F_1\right)$$

$$F_3 = f\left(x_n + h\frac{3}{4}, y_n + h\frac{3}{4}F_2\right)$$

$$y_{n+1} = y_n + h\left(\frac{2}{9}F_1 + \frac{1}{3}F_2 + \frac{4}{9}F_3\right)$$

$$x_{n+1} = x_n + h$$

$$F_4 = f(x_{n+1}, y_{n+1})$$

$$errest = h\left(\frac{5}{72}F_1 - \frac{1}{12}F_2 - \frac{1}{9}F_3 + \frac{1}{8}F_4\right)$$

The error estimate *errest* is used to control the step size automatically. For a thorough discussion of how this can be done, refer to *Numerical Solution of Ordinary Differential Equations* by L. F. Shampine (New York: Chapman & Hall, 1994).

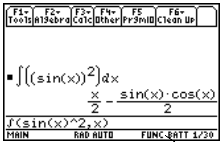
The TI-89 Titanium / Voyage™ 200 software does not adjust the step size to land on particular output points. Rather, it takes the biggest steps that it can (based on the error tolerance diftol) and obtains results for $x_n \le x \le x_{n+1}$ using the cubic interpolating polynomial passing through the point (x_n, y_n) with slope F_1 and through (x_{n+1}, y_{n+1}) with slope F_4. The interpolant is efficient and provides results throughout the step that are just as accurate as the results at the ends of the step.

Battery Information

The TI-89 Titanium / Voyage™ 200 uses two types of batteries: four alkaline batteries, and one button cell battery as a backup for retaining memory while you change the alkaline batteries.

When to Replace the Batteries

As the alkaline batteries run down, the display will begin to dim (especially during calculations). To compensate for this, you will need to adjust the contrast to a higher setting. If you find it necessary to increase the contrast setting frequently, you will need to replace the alkaline batteries. To assist you, a BATT indicator (BATT) will display in the status line area when the batteries have drained down to the point when you should replace them soon. When the BATT indicator is displayed in reverse text (BATT), you must replace the alkaline batteries immediately.



BATT indicator

Note: *To avoid loss of information stored in memory, the TI-89 Titanium / Voyage™ 200 must be off. Do not remove the alkaline batteries and the button cell backup battery at the same time.*

To avoid loss of data, do not remove the button cell backup battery unless four fresh alkaline batteries are installed.

Effects of Replacing the Batteries

If you do not remove both types of batteries at the same time or allow them to run down completely, you can change either type of battery without losing anything in memory.

Battery Precautions

Take these precautions when replacing batteries:

- Do not leave batteries within the reach of children.
- Do not mix new and used batteries. Do not mix brands (or types within brands) of batteries.
- Do not mix rechargeable and non-rechargeable batteries.
- Install batteries according to polarity (+ and -) diagrams.
- Do not place non-rechargeable batteries in a battery recharger.
- Properly dispose of used batteries immediately.
- Do not incinerate or dismantle batteries.

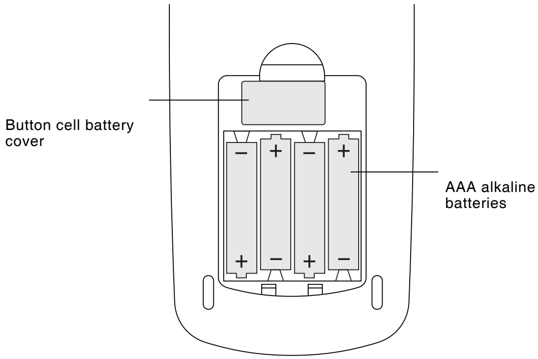
Replacing the Alkaline Batteries in the TI-89 Titanium

1. If the TI-89 Titanium is on, turn it off (press [2nd] [OFF]) to avoid loss of information stored in memory.
2. Slide the protective cover over the keyboard and place the device face down.
3. Push down on the battery cover latch, and then pull up to remove the cover.
4. Remove all four discharged AAA batteries.
5. Install four new AAA alkaline batteries, arranged according to the polarity (+ and -) diagram inside the battery compartment.
6. Replace the battery cover by inserting the two prongs into the two slots at the bottom of the battery compartment, and then push the cover until the latch snaps closed.

Replacing the button cell backup battery in the TI-89 Titanium

To replace the button cell backup battery, remove the battery cover and unscrew the tiny screw holding the BACK UP BATTERY cover in place.

Remove the old battery and install a new battery, positive (+) side up. Replace the cover and the screw.



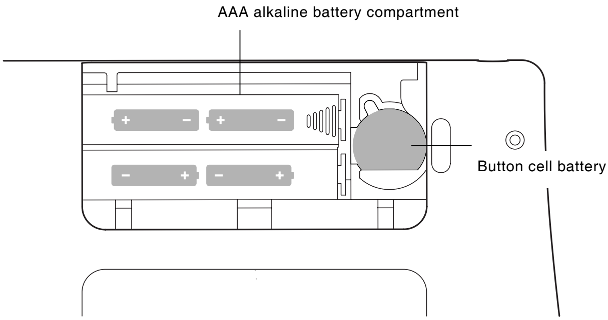
Replacing the Alkaline Batteries in the Voyage 200

1. If the Voyage™ 200 is on, turn it off (press [2nd] [OFF]) to avoid loss of information stored in memory.
2. Slide the protective cover over the keyboard and place the device face down.
3. Press the notched battery cover and slide it off, away from the device.
4. Remove all four discharged AAA batteries.
5. Install four new AAA alkaline batteries, arranged according to the polarity (+ and -) diagram inside the battery compartment.
6. Slide the battery cover onto the device, prong side first. Gently push the cover until the prongs snap into place.

Replacing the button cell battery in the Voyage 200

To replace the button cell backup battery, remove the battery cover. Insert a blunt object into the circular indentation next to the battery. Gently place a finger on the battery and pry it out.

Slide in a new battery, positive (+) side up. Press firmly to snap the new button cell battery into place.



In Case of Difficulty

If you have difficulty operating the TI-89 Titanium / Voyage™ 200, the following suggestions may help you correct the problem.

Suggestions

If:	Suggested action:
You cannot see anything on the display.	Press ◆ + to darken or ◆ − to lighten the display contrast.
The BATT indicator is displayed.	Replace the batteries. If BATT is displayed in reverse text (BATT), replace the batteries as soon as possible.
The BUSY indicator is displayed.	A calculation is in progress. If you want to stop the calculation, press ON.
The PAUSE indicator is displayed.	A graph or program is paused and the TI-89 Titanium / Voyage 200 is waiting for input; press ENTER.
An error message is displayed.	Refer to the list of error messages in this module. Press ESC to clear.
The TI-89 Titanium / Voyage 200 does not appear to be working properly.	Press ESC several times to exit any menu or dialog box and to return the cursor to the entry line. — or — Be sure that the batteries are installed properly and that they are fresh.

Note: *Correcting a "lock up" will reset your TI-89 Titanium / Voyage 200 and clear its memory.*

If:	Suggested action:
The TI-89 Titanium appears to be "locked up" and will not respond to keyboard input.	The following action clears RAM. This erases all data, programs, and user-defined variables, functions, or folders. Press and hold [◄], [►], and [2nd]. Then press and release [ON]. The following action clears RAM *and* Flash ROM. This erases all data, programs, user-defined variables, functions, folders, Flash applications, and the user data archive. 1. Remove one of the four AAA batteries. 2. Press and hold [(-)] and [)] as you reinstall the battery. 3. Continue holding [(-)] and [)] for five seconds before releasing.
The Voyage™ 200 appears to be "locked up" and will not respond to keyboard input.	The following action clears RAM. This erases all data, programs, and user-defined variables, functions, or folders. Press and hold [2nd] and [hand]. Then press and release [ON]. The following action clears RAM *and* Flash ROM. This erases all data, programs, user-defined variables, functions, folders, Flash applications, and the user data archive. 1. Remove one of the four AAA batteries. 2. Press and hold [(-)] and [)] as you reinstall the battery. 3. Continue holding [(-)] and [)] for five seconds before releasing.

Texas Instruments Support and Service

For general information	**Home Page:**	education.ti.com
	KnowledgeBase and e-mail inquiries:	education.ti.com/support
	Phone:	(800) TI-CARES; (800) 842- 2737 For U.S., Canada, Mexico, Puerto Rico, and Virgin Islands only
	International Information:	education.ti.com/international
For technical support	**KnowledgeBase and support by e-mail:**	education.ti.com/support
	Phone (not toll-free):	(972) 917-8324

For product (hardware) service

Customers in the U.S., Canada, Mexico, Puerto Rico and Virgin Islands: Always contact Texas Instruments Customer Support before returning a product for service.

All other customers: Refer to the leaflet enclosed with this product (hardware) or contact your local Texas Instruments retailer/distributor.

Index

Symbols

Numerics

A

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

U

X

Y

Z