TEXAS INSTRUMENTS

TI-89 Titanium

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.

Table of Contents

Introduction

The TI-89 Titanium graphing calculator

This guidebook provides information about a powerful, advanced graphing calculator available from Texas Instruments: the TI-89 Titanium.

Your TI-89 Titanium comes equipped with a variety of preinstalled software applications (Apps) that have features relevant to many different subjects and curriculums.

Using Flash read-only memory (ROM) for the TI-89 Titanium (4 megabytes [MB] available), you can install additional Apps and increase the capabilities of your calculator. Installing Apps and operating system (OS) upgrades is like installing software on a computer. All you need is TI Connect™ software and a TI Connectivity Cable.

The TI-89 Titanium graphical user interface (GUI) and configurable Apps desktop make it easy to organize Apps into categories that you create.

Extend the reach of your TI-89 Titanium with accessories, such as the Calculator-Based Laboratory™ (CBL 2™) systems, Calculator-Based Ranger™ (CBR™) system, TI-Presenter™ video adapter, and TI ViewScreen™ overhead panel.

The CBL 2 and CBR systems offer static and real-world data collection. Use the TI-Presenter video adapter to connect the TI-89 Titanium to video display/recording devices, such as TVs, VCRs, video cameras, and computer projectors. The TI ViewScreen overhead panel lets you project an enlarged image of the TI-89 Titanium display so an entire class can view it.

How to use this guidebook

We've added conventions to make it easier for you to get the most out of this guidebook.

- Key symbols appear throughout the guidebook.
- Many keys can perform more than one function. To use secondary functions, which are printed above the keys, you must first press 2nd, alpha, or ◆. These extra functions are printed within brackets in this guidebook.

  For example, a procedure might include this key sequence used to display a menu of special characters:

  Press 2nd [CHAR]. (Press and release the 2nd key, then press the [CHAR] key, which is the second function of the + key.)

- Key commands that require you to press and hold two keys at the same time are indicated by the phrase *press and hold*. For example, the instruction to darken the display contrast is:

  Press and hold [◆] and tap [+].

- Your graphing calculator uses menus to access many operations. Menu options often can be selected using one of two methods. For example,

  Press [F2] **9:Trig**

  means that you can choose the **Trig** option first by pressing [F2] and then either pressing the **9** key or pressing ⌄ as many times as required to select **Trig,** and then pressing [ENTER].

The chapters in this guidebook include:

Getting Started – Offers students and educators in all curriculums a quick overview of the basic operations of the TI-89 Titanium.

Previews – A set of short examples that include step-by-step procedures, actual keystrokes, and sample displays.

Activities – A set of longer examples that shows how to solve, analyze, and visualize actual mathematical problems.

Connectivity – How to link your graphing calculator to another calculator or to a computer using either the USB or the I/O port, with details about how to transmit variables and applications, and how to upgrade the operating system.

Memory and Variable Management – How to manage variables stored in your graphing calculator’s memory and in the data archive, a protected area of memory separate from RAM (random access memory).

Technical Reference – Includes the syntax and action of each function and instruction included in the operating system, an alphabetical listing of operations, error messages, and other reference information.

The remainder of the product information is available in electronic form. This comprehensive set of electronic chapters is included on the CD-ROM that came with your TI-89 Titanium. This same information is also available online as a free download at:
education.ti.com/guides

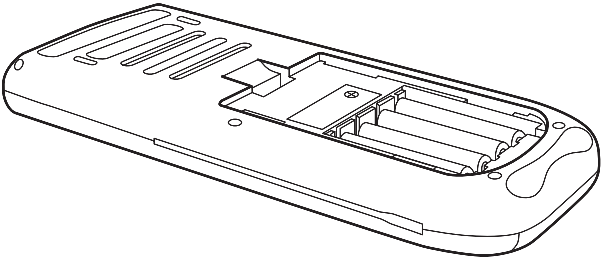
Getting Started

Initial start-up

Installing the AAA Batteries

The TI-89 Titanium uses four AAA alkaline batteries and a backup silver oxide battery (SR44SW or 303). The backup batteries are 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

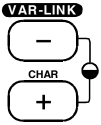
After installing the batteries included with the calculator, press [ON]. The Apps desktop appears.

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

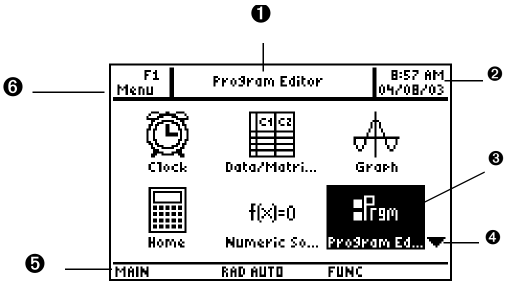
- To lighten the display, press and hold ◆ and tap −.
- To darken the display, press and hold ◆ and tap +.



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 Titanium on again, it will be exactly as you left it.

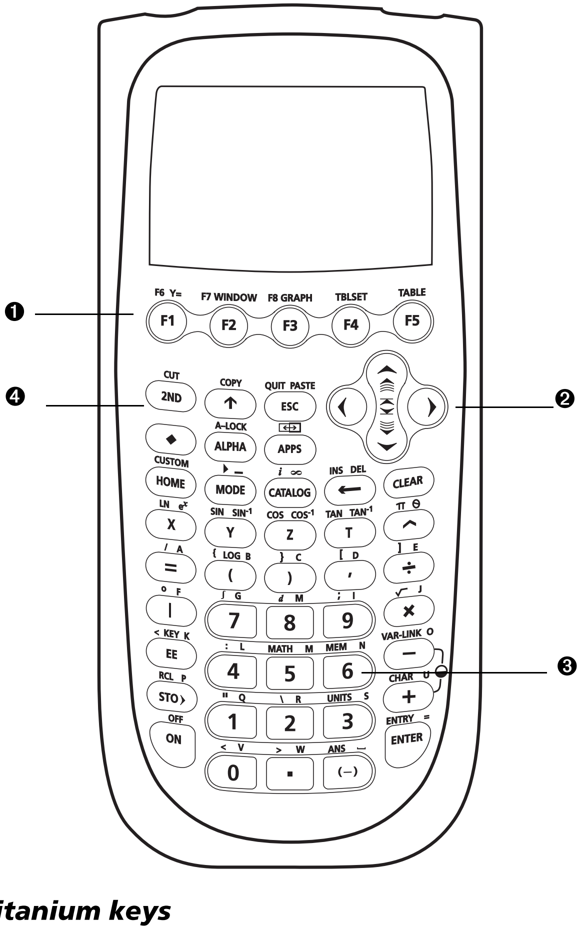
Note: [OFF] is the second function of the [ON] key.

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 function when a calculation or program is in progress, unless a pause is specified in the calculation or program.

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.

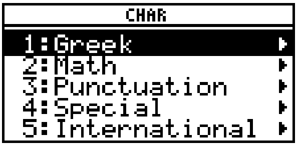
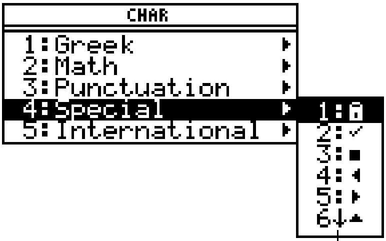
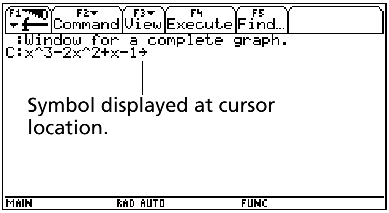
Entering special characters

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].

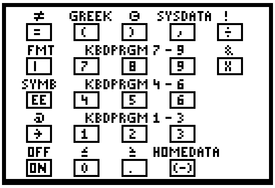
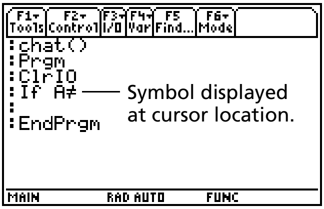
Example: Enter the right arrow symbol (→) in the Text Editor.

Press	Result
[2nd] [CHAR]	 CHAR 1:Greek ▸ 2:Math ▸ 3:Punctuation ▸ 4:Special ▸ 5:International ▸
4	 CHAR 1:Greek ▸ 2:Math ▸ 3:Punctuation ▸ 4:Special ▸ 5:International ▸ 1:🔒 2:✓ 3:■ 4:◂ 5:▸ 6↓▴ Scroll down for more characters.
9 – or – Press ⊙ repeatedly to select **9:**→ and press [ENTER]	 F1 F2▾ Command F3▾ View F4 Execute F5 Find... MAIN RAD AUTO FUNC :Window for a complete graph. C:x^3-2x^2+x-1→ Symbol displayed at cursor location.

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

To type most characters, press [2nd] 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]	 ≠ GREEK Θ SYSDATA ! = ( ) , ÷ FMT KBDPRGM 7 - 9 & | 7 8 9 × SYMB KBDPRGM 4 - 6 EE 4 5 6 @ KBDPRGM 1 - 3 → 1 2 3 OFF ≤ ≥ HOMEDATA ON 0 . (-)
[◆] [=]	 F1 Tools F2 Control F3 I/O F4 Var F5 Find... F6 Mode :chat() :Prgm :ClrIO :If A≠ : :EndPrgm Symbol displayed at cursor location.

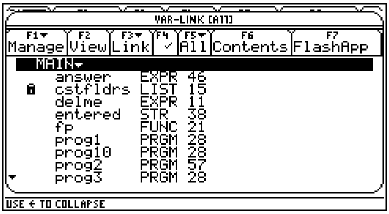
Modifier keys

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.

Keys	Description
[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]	

```
   answer     EXPR  46
🔒 cstfldrs   LIST  15
   delme      EXPR  11
   entered    STR   38
   fp         FUNC  21
   prog1      PRGM  28
   prog10     PRGM  28
   prog2      PRGM  57
▾  prog3      PRGM  28
USE ← TO COLLAPSE
```

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.

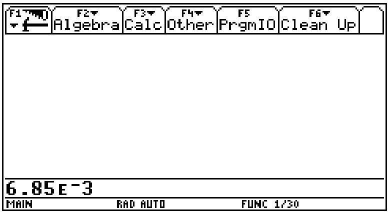
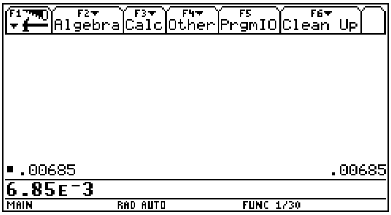
To enter a negative number, press [(-)] before typing the number.

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 [2nd] [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**	 F1 F2 Algebra F3 Calc F4 Other F5 PrgmIO F6 Clean Up MAIN RAD AUTO FUNC 1/30 6.85E-3
[ENTER]	 F1 F2 Algebra F3 Calc F4 Other F5 PrgmIO F6 Clean Up MAIN RAD AUTO FUNC 1/30 ■ .00685 .00685 6.85E-3

Other important keys

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

Key Command	Description
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.
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

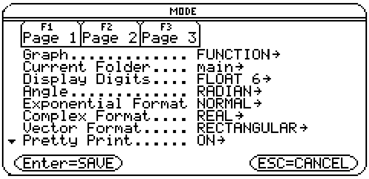
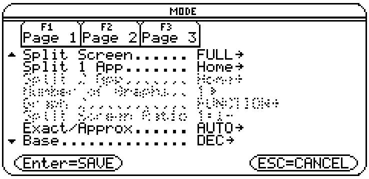
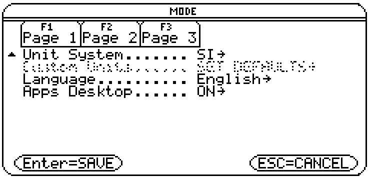
Modes control how the TI-89 Titanium displays and interprets information. All numbers, including elements of matrices and lists, are displayed according to the current mode settings. When the TI-89 Titanium is turned off, the Constant Memory™ feature retains all of the mode settings you have selected.

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.

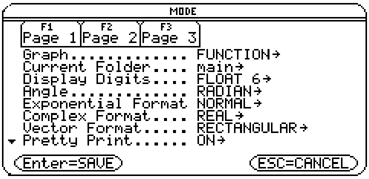
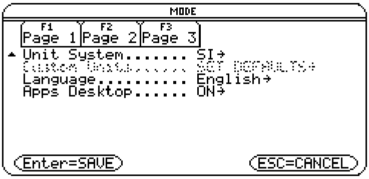
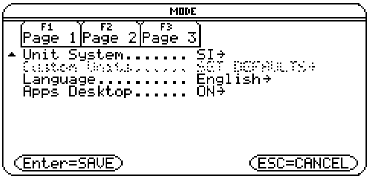
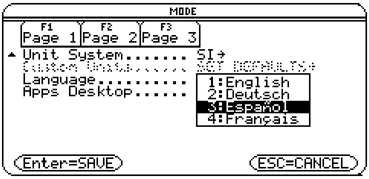
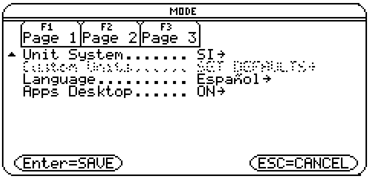
Note: Modes that are grayed out are available only if other required mode settings are selected. For example, the Custom Units mode listed on Page 3 is available only if the Unit System mode is set to CUSTOM.

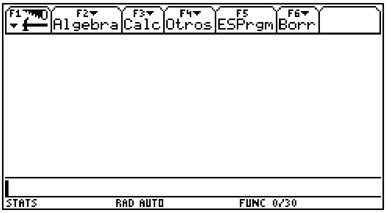
Viewing mode settings

Press	Result
[MODE]	 Graph............. FUNCTION→ Current Folder.... main→ Display Digits.... FLOAT 6→ Angle............. RADIAN→ Exponential Format NORMAL→ Complex Format.... REAL→ Vector Format..... RECTANGULAR→ Pretty Print...... ON→
[F2]	 Split Screen...... FULL→ Split 1 App....... Home→ Split 2 App....... Home→ Number of Graphs.. 1→ Graph 2........... FUNCTION→ Split Screen Ratio 1:1→ Exact/Approx...... AUTO→ Base.............. DEC→
[F3]	 Unit System....... SI→ Custom Units...... SET DEFAULTS→ Language.......... English→ Apps Desktop...... ON→

Changing mode settings

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

Press	Result
[MODE]	 MODE F1 Page 1 F2 Page 2 F3 Page 3 Enter=SAVE ESC=CANCEL Graph............ FUNCTION→ Current Folder.... main→ Display Digits.... FLOAT 6→ Angle............ RADIAN→ Exponential Format NORMAL→ Complex Format.... REAL→ Vector Format..... RECTANGULAR→ Pretty Print...... ON→
[F3]	 MODE F1 Page 1 F2 Page 2 F3 Page 3 Enter=SAVE ESC=CANCEL Unit System....... SI→ Custom Units...... SET DEFAULTS→ Language.......... English→ Apps Desktop...... ON→
Scroll down to the Language field. ▼	 MODE F1 Page 1 F2 Page 2 F3 Page 3 Enter=SAVE ESC=CANCEL Unit System....... SI→ Custom Units...... SET DEFAULTS→ Language.......... English→ Apps Desktop...... ON→
Press ▶ and then press ▼ until **3:Español** is highlighted. **Note:** Your menu list might vary, depending on the languages installed.	 MODE F1 Page 1 F2 Page 2 F3 Page 3 1:English 2:Deutsch 3:Español 4:Français Enter=SAVE ESC=CANCEL Unit System....... SI→ Custom Units...... SET DEFAULTS→ Language.......... Apps Desktop......
[ENTER]	 MODE F1 Page 1 F2 Page 2 F3 Page 3 Enter=SAVE ESC=CANCEL Unit System....... SI→ Custom Units...... SET DEFAULTS→ Language.......... Español→ Apps Desktop...... ON→

Press	Result
[ENTER] **Note:** The previous open App appears (in this example, the calculator Home screen).	 F1 F2 Algebra F3 Calc F4 Otros F5 ESPrgm F6 Borr STATS RAD AUTO FUNC 0/30

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

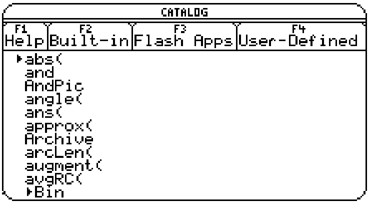
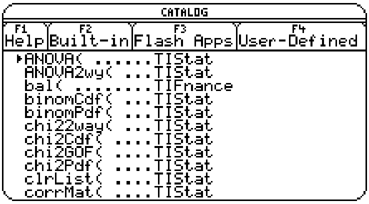
Using the Catalog to access commands

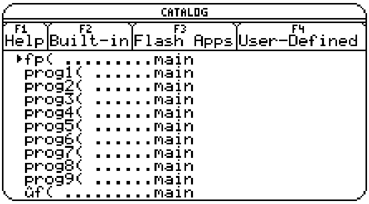
Use the Catalog to access a list of TI-89 Titanium commands, including functions, instructions, and user-defined programs. Commands are listed alphabetically. Commands not beginning with a letter are found at the end of the list (&, /, +, –, etc.).

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)	 CATALOG F1 Help F2 Built-in F3 Flash Apps F4 User-Defined ▸abs(and AndPic angle(ans(approx(Archive arcLen(augment(avgRC(▸Bin
[F3] (displays Flash Apps commands, if any)	 CATALOG F1 Help F2 Built-in F3 Flash Apps F4 User-Defined ▸ANOVA(......TIStat ANOVA2wy(...TIStat bal(........TIFnance binomCdf(...TIStat binomPdf(...TIStat chi22way(...TIStat chi2Cdf(....TIStat chi2GOF(....TIStat chi2Pdf(....TIStat clrList(....TIStat corrMat(....TIStat

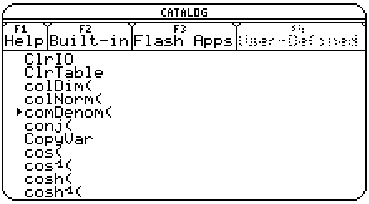
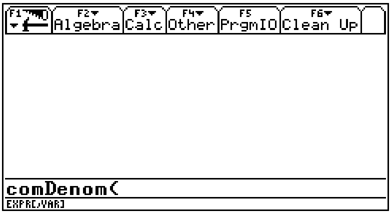
Press	Result
F4 (displays User-Defined commands, if any)	 CATALOG F1 Help F2 Built-in F3 Flash Apps F4 User-Defined ▸fp(.........main prog1(......main prog2(......main prog3(......main prog4(......main prog5(......main prog6(......main prog7(......main prog8(......main prog9(......main úf(.........main

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.

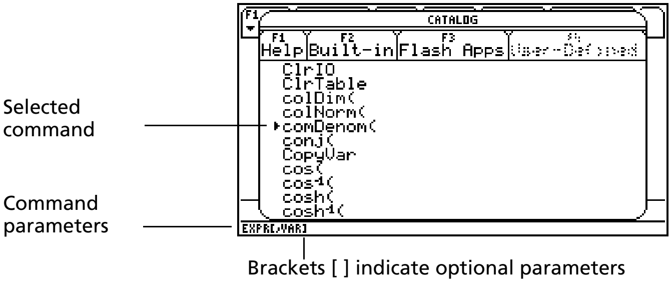
Note: Before selecting a command, position the cursor where you want the command to appear.

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

Press	Result
CATALOG alpha **C** 2nd ▼ Then press ▼ until the pointer is at the **comDenom(** function.	 CATALOG F1 Help F2 Built-in F3 Flash Apps F4 User-Defined ClrIO ClrTable colDim(colNorm(▸comDenom(conj(CopyVar cos(cos^{-1}(cosh(cosh^{-1}(
ENTER	 F1 F2 Algebra F3 Calc F4 Other F5 PrgmIO F6 Clean Up comDenom(EXPR[,VAR]

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

Note: Pressing F1 also displays the parameters for the selected command.



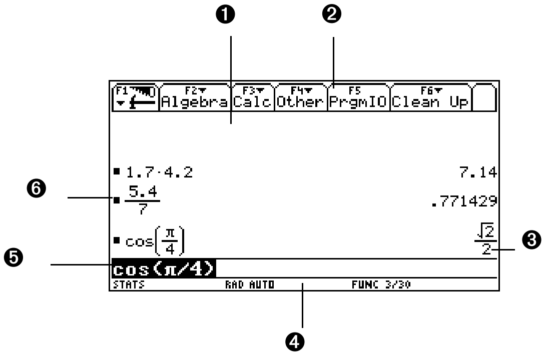
To exit the Catalog without selecting a command, press ESC.

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 last entry is displayed here.

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

About the history area

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 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:

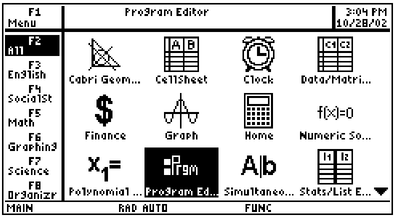
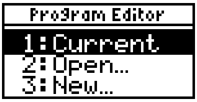
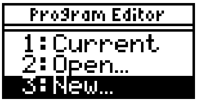
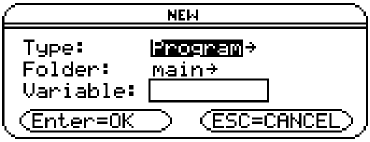
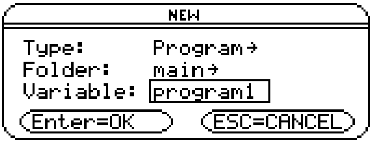
Note: The TI-89 Titanium uses the general term *variable* to refer to the App data files that you create.

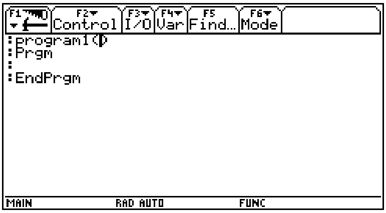
Option	**Description**
Current	Returns the screen displayed when you last viewed the App. If no current App variable exists, the New dialog box appears.
Open	Lets you open an existing file.

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 Program Ed...	 F1 Menu Program Editor 3:04 PM 10/28/02 F2 All F3 English F4 SocialSt F5 Math F6 Graphing F7 Science F8 Organizr Cabri Geom... CellSheet Clock Data/Matri... Finance Graph Home f(x)=0 Numeric So... Polynomial ... Program Ed... Simultaneo... Stats/List E... MAIN RAD AUTO FUNC
[ENTER]	
3	
[ENTER]	 Type: Program→ Folder: main→ Variable:
▼▼ **program1**	 Type: Program→ Folder: main→ Variable: program1

Press	Result
[ENTER] [ENTER]	 F1 F2 Control F3 I/O F4 Var F5 Find... F6 Mode MAIN RAD AUTO FUNC :program1() :Prgm : :EndPrgm

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

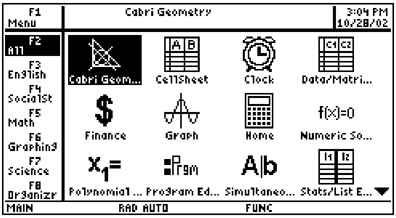
On the TI-89 Titanium, the Apps category names appear only in the **F1** Menu. To select an Apps category, press [F1] **2:Select Category** and use the cursor keys to highlight an Apps category, and then press [ENTER] to select the highlighted category. You can also use the function key shortcuts to 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.
[2nd] [F6] Graphing	Customizable category. Graphing is the default.

Key	Description
[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]	 F1 Menu Cabri Geometry 3:04 PM 10/28/02 F2 All F3 English F4 SocialSt F5 Math F6 Graphing F7 Science F8 Organizr Cabri Geom... CellSheet Clock Data/Matri... Finance Graph Home Numeric So... Polynomial ... Program Ed... Simultaneo... Stats/List E... MAIN RAD AUTO FUNC

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 19.)

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

Customizing the Apps categories

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.)

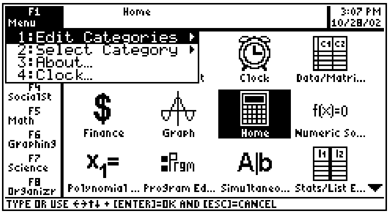
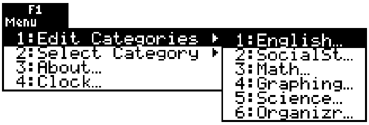
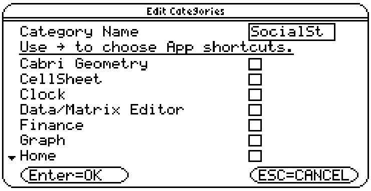
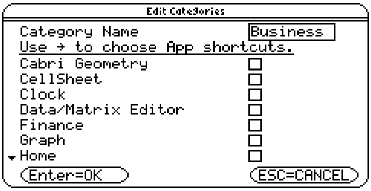
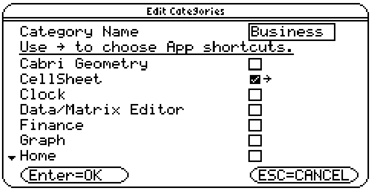
To customize the [F3] through [2nd] [F8] Apps categories:

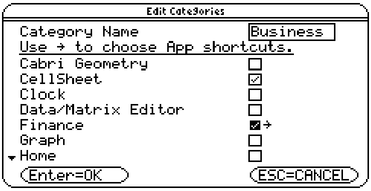
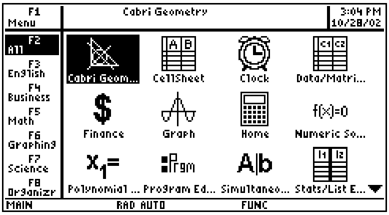
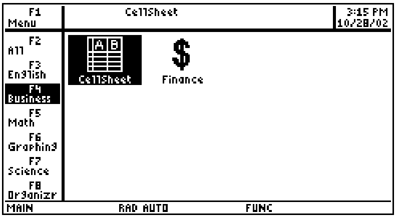
1. Select [F1] **1:Edit Categories**. A submenu displays the six customizable Apps category names. (The All category is not listed.)
2. Highlight an Apps category and press [ENTER]. The Edit Categories dialog box appears with a list of installed Apps and a text box with the category name highlighted.
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]	 F1 Menu Home 3:07 PM 10/28/02 1:Edit Categories ▸ 2:Select Category ▸ 3:About… 4:Clock… F4 SocialSt F5 Math F6 Graphing F7 Science F8 Organizr Clock Data/Matri… Finance Graph Home Numeric So… Polynomial … Program Ed… Simultaneo… Stats/List E… TYPE OR USE ←→↑↓ + [ENTER]=OK AND [ESC]=CANCEL
⊳	 F1 Menu 1:Edit Categories ▸ 2:Select Category ▸ 3:About… 4:Clock… 1:English… 2:SocialSt… 3:Math… 4:Graphing… 5:Science… 6:Organizr…
2 – or – ⊽ [ENTER]	 Edit Categories Category Name SocialSt Use → to choose App shortcuts. Cabri Geometry □ CellSheet □ Clock □ Data/Matrix Editor □ Finance □ Graph □ Home □ Enter=OK ESC=CANCEL
[2nd] [a-lock] [↑] **Business**	 Edit Categories Category Name Business Use → to choose App shortcuts. Cabri Geometry □ CellSheet □ Clock □ Data/Matrix Editor □ Finance □ Graph □ Home □ Enter=OK ESC=CANCEL
⊽ ⋮ ⊳	 Edit Categories Category Name Business Use → to choose App shortcuts. Cabri Geometry □ CellSheet ☒→ Clock □ Data/Matrix Editor □ Finance □ Graph □ Home □ Enter=OK ESC=CANCEL

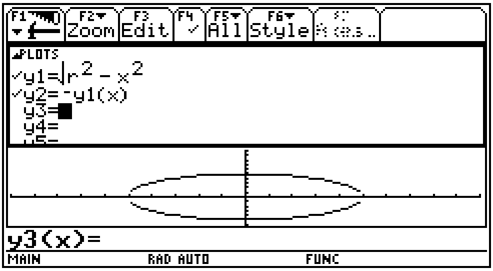
Press	Result
▼ ⋮ ▶	 Edit Categories Category Name Business Use → to choose App shortcuts. Cabri Geometry □ CellSheet ☑ Clock □ Data/Matrix Editor □ Finance ☑→ Graph □ ▾Home □ Enter=OK ESC=CANCEL
ENTER	 F1 Menu · Cabri Geometry · 3:04 PM 10/28/02 F2 All · F3 English · F4 Business · F5 Math · F6 Graphing · F7 Science · F8 Organizr Cabri Geom… · CellSheet · Clock · Data/Matri… · Finance · Graph · Home · Numeric So… · Polynomial … · Program Ed… · Simultaneo… · Stats/List E… MAIN RAD AUTO FUNC
F4	 F1 Menu · CellSheet · 3:15 PM 10/28/02 F2 All · F3 English · F4 Business · F5 Math · F6 Graphing · F7 Science · F8 Organizr CellSheet · Finance MAIN RAD AUTO FUNC

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

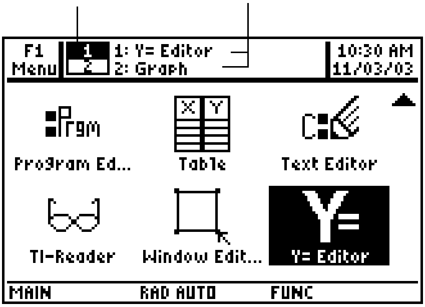


PLOTS
✓y1=$\sqrt{r^2 - x^2}$
✓y2=−y1(x)
y3=
y4=
y5=

y3(x)=
MAIN RAD AUTO FUNC

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.



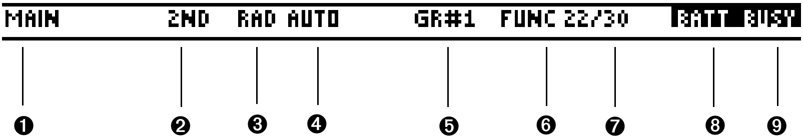
Split-screen status (highlight indicates the portion where the next App selected will open.)

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.



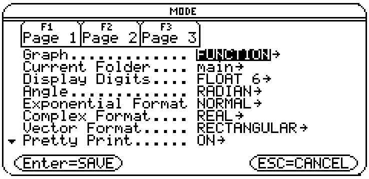
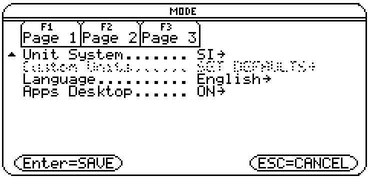
Indicator	Meaning
❶ Current folder	Name of the selected folder (MAIN is the default folder.)
❷ Modifier key	Selected modifier key ([2nd], [♦], [↑]), if any.
❸ Angle mode	Selected units in which angle values are displayed and interpreted (RAD, DEG)
❹ Exact/Approx mode	Mode in which answers are calculated and displayed (AUTO, EXACT, APPROX)

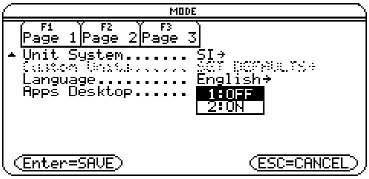
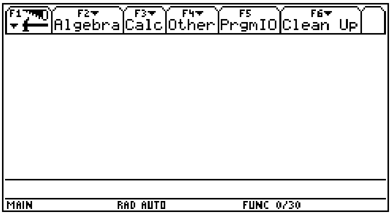
Indicator	Meaning
❺ Graph number	Active of two independent graphs in split-screen mode (GR#1, GR#2)
❻ Graph mode	Selected type of graph that can be plotted (FUNC, PAR, POL, SEQ, 3D, DE)
❼ Entry/Answer pairs	22/30–Number of entry/answer pairs (default is 30, maximum is 99) in the history area of the calculator 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]	 MODE F1 Page 1 | F2 Page 2 | F3 Page 3 Graph............. FUNCTION→ Current Folder.... main→ Display Digits.... FLOAT 6→ Angle............. RADIAN→ Exponential Format NORMAL→ Complex Format.... REAL→ Vector Format..... RECTANGULAR→ Pretty Print...... ON→ Enter=SAVE ESC=CANCEL
[F3]	 MODE F1 Page 1 | F2 Page 2 | F3 Page 3 Unit System....... SI→ Custom Units...... SET DEFAULTS→ Language.......... English→ Apps Desktop...... ON→ Enter=SAVE ESC=CANCEL

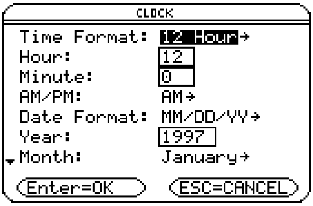
Press	Result
▼ ▼ ▶ ▲	 MODE F1 Page 1 F2 Page 2 F3 Page 3 Unit System....... SI→ Custom Units...... SET DEFAULTS→ Language.......... English→ Apps Desktop...... 1:OFF 2:ON Enter=SAVE ESC=CANCEL
ENTER ENTER **Note:** The previous open App appears (in this example, the calculator Home screen).	 F1 F2 Algebra F3 Calc F4 Other F5 PrgmIO F6 Clean Up MAIN RAD AUTO FUNC 0/30

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.



CLOCK

Time Format: 12 Hour→
Hour: 12
Minute: 0
AM/PM: AM→
Date Format: MM/DD/YY→
Year: 1997
Month: January→
Enter=OK ESC=CANCEL

▼ 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*).

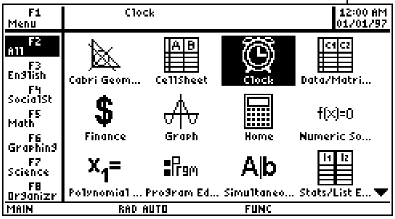
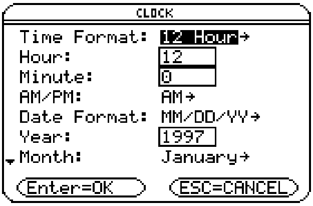
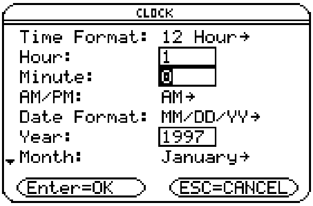
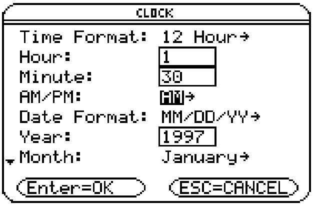
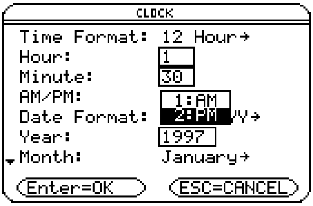
   — or —

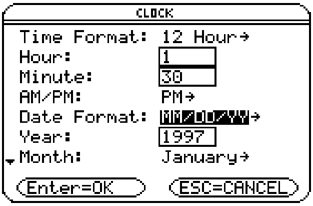
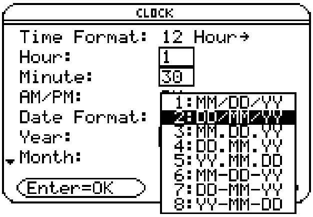
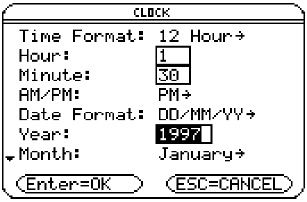
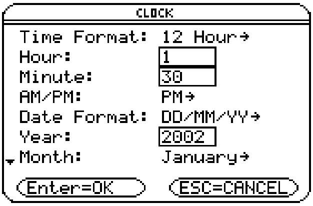
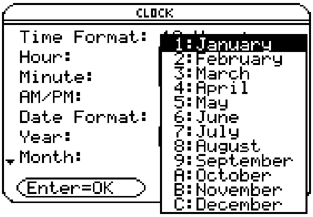
   To save your settings and exit, press ENTER. The time is updated in the top right corner of the Apps desktop.

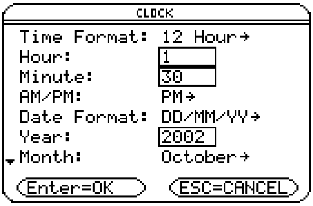
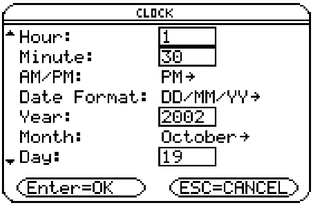
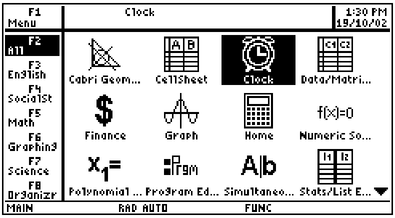
Setting the date

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 Clock	Time and date  F1 Menu Clock 12:00 AM 01/01/97 F2 All F3 English F4 SocialSt F5 Math F6 Graphing F7 Science F8 Organizr Cabri Geom... CellSheet Clock Data/Matri... Finance Graph Home f(x)=0 Numeric So... Polynomial ... Program Ed... Simultaneo... Stats/List E... MAIN RAD AUTO FUNC
ENTER	 CLOCK Time Format: 12 Hour→ Hour: 12 Minute: 0 AM/PM: AM→ Date Format: MM/DD/YY→ Year: 1997 Month: January→ Enter=OK ESC=CANCEL
↓ **1** ↓	 CLOCK Time Format: 12 Hour→ Hour: 1 Minute: 0 AM/PM: AM→ Date Format: MM/DD/YY→ Year: 1997 Month: January→ Enter=OK ESC=CANCEL
3 0 ↓	 CLOCK Time Format: 12 Hour→ Hour: 1 Minute: 30 AM/PM: AM→ Date Format: MM/DD/YY→ Year: 1997 Month: January→ Enter=OK ESC=CANCEL
→ ↓	 CLOCK Time Format: 12 Hour→ Hour: 1 Minute: 30 AM/PM: 1:AM Date Format: 2:PM YY→ Year: 1997 Month: January→ Enter=OK ESC=CANCEL

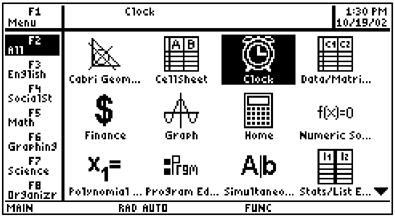
Press	Result
ENTER ▼	 CLOCK Time Format: 12 Hour→ Hour: 1 Minute: 30 AM/PM: PM→ Date Format: MM/DD/YY→ Year: 1997 Month: January→ Enter=OK ESC=CANCEL
▶ ▼	 CLOCK Time Format: 12 Hour→ Hour: 1 Minute: 30 AM/PM: Date Format: Year: Month: Enter=OK 1:MM/DD/YY 2:DD/MM/YY 3:MM.DD.YY 4:DD.MM.YY 5:YY.MM.DD 6:MM-DD-YY 7:DD-MM-YY 8:YY-MM-DD
ENTER ▼	 CLOCK Time Format: 12 Hour→ Hour: 1 Minute: 30 AM/PM: PM→ Date Format: DD/MM/YY→ Year: 1997 Month: January→ Enter=OK ESC=CANCEL
2 0 0 2	 CLOCK Time Format: 12 Hour→ Hour: 1 Minute: 30 AM/PM: PM→ Date Format: DD/MM/YY→ Year: 2002 Month: January→ Enter=OK ESC=CANCEL
▼ ▶	 CLOCK Time Format: Hour: Minute: AM/PM: Date Format: Year: Month: Enter=OK 1:January 2:February 3:March 4:April 5:May 6:June 7:July 8:August 9:September A:October B:November C:December

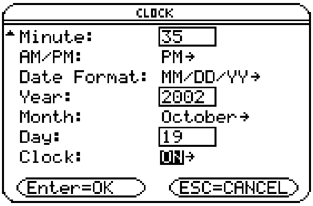
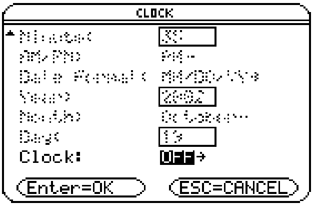
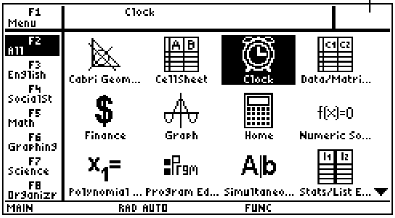
Press	Result
Scroll down to October and press [ENTER]	 CLOCK Time Format: 12 Hour→ Hour: 1 Minute: 30 AM/PM: PM→ Date Format: DD/MM/YY→ Year: 2002 Month: October→
⊽ **1 9**	 CLOCK Hour: 1 Minute: 30 AM/PM: PM→ Date Format: DD/MM/YY→ Year: 2002 Month: October→ Day: 19
[ENTER] [ENTER]	Revised time and date  F1 Menu Clock 1:30 PM 19/10/02 F2 All F3 English F4 SocialSt F5 Math F6 Graphing F7 Science F8 Organizr Cabri Geom... CellSheet Clock Data/Matri... Finance Graph Home Numeric So... Polynomial ... Program Ed... Simultaneo... Stats/List E... MAIN RAD AUTO FUNC

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	Clock on  F1 Menu Clock 1:30 PM 10/19/02 F2 All F3 English F4 SocialSt F5 Math F6 Graphing F7 Science F8 Organizr Cabri Geom... CellSheet Clock Data/Matri... Finance Graph Home Numeric So... Polynomial ... Program Ed... Simultaneo... Stats/List E... MAIN RAD AUTO FUNC

Press	Result
ENTER Scroll down to the Clock field.	 CLOCK Minute: 35 AM/PM: PM→ Date Format: MM/DD/YY→ Year: 2002 Month: October→ Day: 19 Clock: ON→ Enter=OK ESC=CANCEL
▶ ▲ ENTER	 CLOCK Minute: 35 AM/PM: PM→ Date Format: MM/DD/YY→ Year: 2002 Month: October→ Day: 19 Clock: OFF→ Enter=OK ESC=CANCEL
ENTER	Clock off  F1 Menu F2 All F3 English F4 SocialSt F5 Math F6 Graphing F7 Science F8 Organizr Clock Cabri Geom... CellSheet Clock Data/Matri... Finance Graph Home f(x)=0 Numeric So... Polynomial ... Program Ed... Simultaneo... Stats/List E... MAIN RAD AUTO FUNC

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.
[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

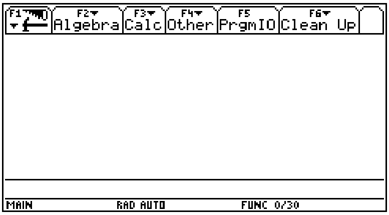
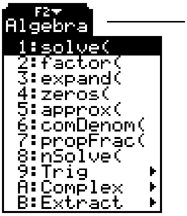
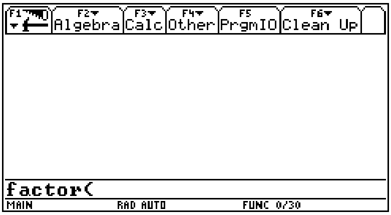
- Press the number or letter to the left of the option you want to select.

  — or —

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

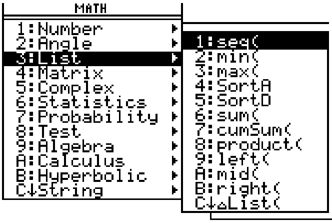
Note: If the first menu option is selected, press ▲ to select the last option on the menu. If the last menu option is selected, press ▼ to select the first option on the menu.

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 Home and press ENTER	 F1▼ F2▼ Algebra F3▼ Calc F4▼ Other F5 PrgmIO F6▼ Clean Up MAIN RAD AUTO FUNC 0/30
F2	 F2▼ Algebra 1:solve(2:factor(3:expand(4:zeros(5:approx(6:comDenom(7:propFrac(8:nSolve(9:Trig ▸ A:Complex ▸ B:Extract ▸ ▼ indicates Algebra menu will open when you press F2.
2 – or – ⊙ ENTER	 F1▼ F2▼ Algebra F3▼ Calc F4▼ Other F5 PrgmIO F6▼ Clean Up factor(MAIN RAD AUTO FUNC 0/30

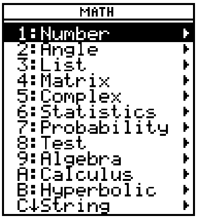
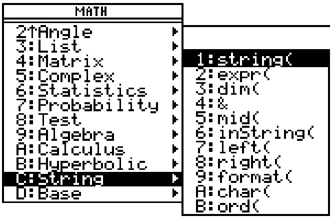
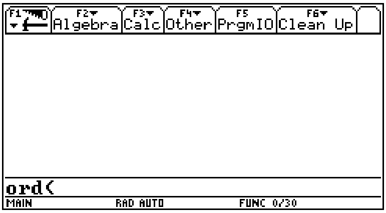
Selecting submenu options

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



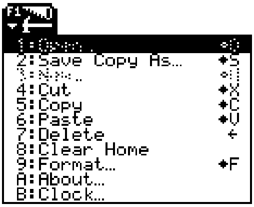
↓ points to additional options.

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

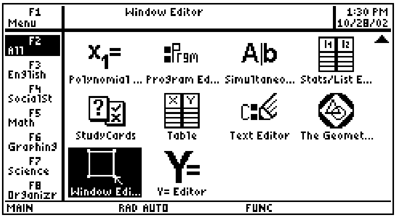
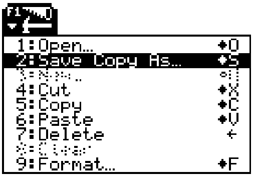
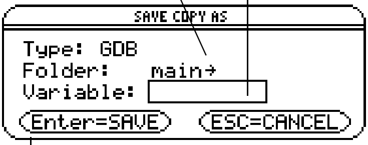
Press	Result
[2nd] [MATH]	 MATH 1:Number 2:Angle 3:List 4:Matrix 5:Complex 6:Statistics 7:Probability 8:Test 9:Algebra A:Calculus B:Hyperbolic C↓String
C – or – ↑ ↑ →	 MATH 2↑Angle 3:List 4:Matrix 5:Complex 6:Statistics 7:Probability 8:Test 9:Algebra A:Calculus B:Hyperbolic C:String D:Base 1:string(2:expr(3:dim(4:& 5:mid(6:inString(7:left(8:right(9:format(A:char(B:ord(
B – or – ↑ [ENTER]	 F1 F2 Algebra F3 Calc F4 Other F5 PrgmIO F6 Clean Up MAIN RAD AUTO FUNC 0/30 ord(

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 Window Edi... and press [ENTER]	 F1 Menu Window Editor 1:30 PM 10/28/02 F2 All F3 English F4 SocialSt F5 Math F6 Graphing F7 Science F8 Organizr Polynomial ... Program Ed... Simultaneo... Stats/List E... StudyCards Table Text Editor The Geomet... Window Edi... Y= Editor MAIN RAD AUTO FUNC
[F1]	 1:Open... ◆O 2:Save Copy As... ◆S 3:New... ◆N 4:Cut ◆X 5:Copy ◆C 6:Paste ◆V 7:Delete ← 8:Clear 9:Format... ◆F
2 – or – ⊖ [ENTER] [ESC]	Press ⊳ to display a list of folders. Type the name of the variable. SAVE COPY AS Type: GDB Folder: main→ Variable: Press [ENTER] twice to save and then 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, and vice versa.

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.

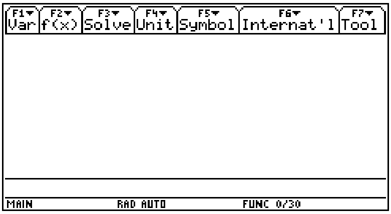
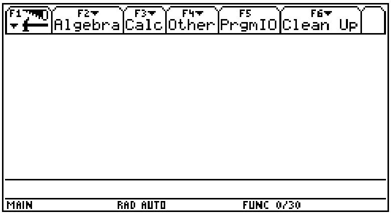
More information is available about menus. (See the electronic *Operating the Calculator* chapter.)

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.

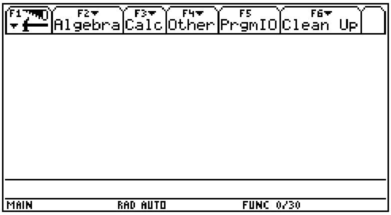
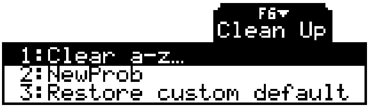
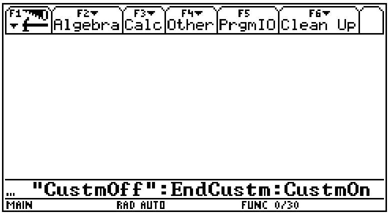
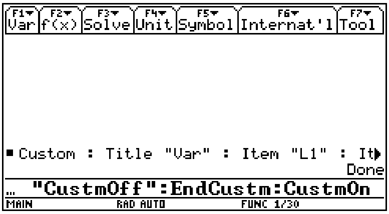
The custom menu replaces the standard toolbar menu on the calculator Home screen. (For details on creating a custom menu, see the electronic *Programming* chapter.) More information is available about custom menus. (See the electronic *Operating the Calculator* chapter.)

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

Press	Result
2nd [CUSTOM]	Default custom menu  F1 Var F2 f(x) F3 Solve F4 Unit F5 Symbol F6 Internat'l F7 Tool MAIN RAD AUTO FUNC 0/30
2nd [CUSTOM]	Normal toolbar menu  F1 F2 Algebra F3 Calc F4 Other F5 PrgmIO F6 Clean Up MAIN RAD AUTO FUNC 0/30

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.

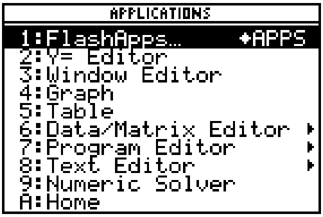
Press	Result
[2nd] [CUSTOM] (to turn off the custom menu and turn on the standard toolbar menu)	 F1 F2 Algebra F3 Calc F4 Other F5 PrgmIO F6 Clean Up MAIN RAD AUTO FUNC 0/30
[2nd] [F6]	 F6 Clean Up 1:Clear a-z… 2:NewProb 3:Restore custom default
3 – or – ▽ ▽ [ENTER]	 F1 F2 Algebra F3 Calc F4 Other F5 PrgmIO F6 Clean Up MAIN RAD AUTO FUNC 0/30 … "CustmOff":EndCustm:CustmOn
[ENTER]	 F1 Var F2 f(x) F3 Solve F4 Unit F5 Symbol F6 Internat'l F7 Tool MAIN RAD AUTO FUNC 1/30 ■Custom : Title "Var" : Item "L1" : It▸ Done … "CustmOff":EndCustm:CustmOn

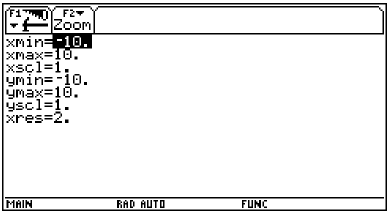
Opening Apps with the Apps desktop turned off

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]	 APPLICATIONS 1:FlashApps… ♦APPS 2:Y= Editor 3:Window Editor 4:Graph 5:Table 6:Data/Matrix Editor ▸ 7:Program Editor ▸ 8:Text Editor ▸ 9:Numeric Solver A:Home

3
– or –
▼ ▼ [ENTER]
	
	xmin=-10.
	xmax=10.
	xscl=1.
	ymin=-10.
	ymax=10.
	yscl=1.
	xres=2.

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

Using split screens

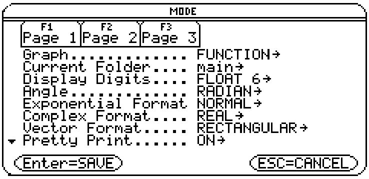
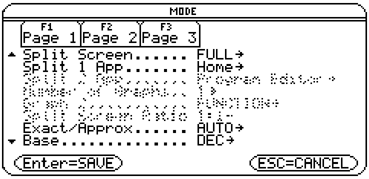
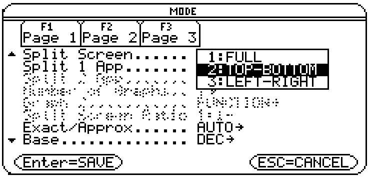
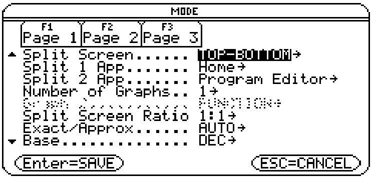
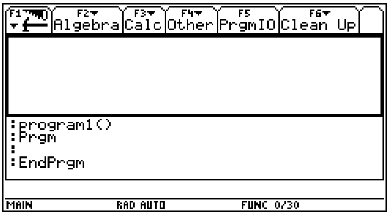
The TI-89 Titanium lets you split the screen to show two Apps at the same time. For example, display both the Y= Editor and Graph screens to compare the list of functions and how they are graphed.

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.

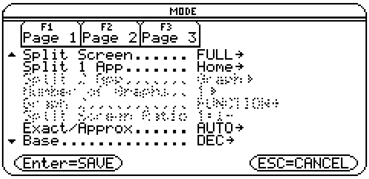
Example: Set split-screen mode to TOP-BOTTOM.

Press	Result
MODE	 MODE F1 Page 1 F2 Page 2 F3 Page 3 Graph............. FUNCTION→ Current Folder.... main→ Display Digits.... FLOAT 6→ Angle............. RADIAN→ Exponential Format NORMAL→ Complex Format.... REAL→ Vector Format..... RECTANGULAR→ Pretty Print...... ON→ Enter=SAVE ESC=CANCEL
F2	 MODE F1 Page 1 F2 Page 2 F3 Page 3 Split Screen...... FULL→ Split 1 App....... Home→ Split 2 App....... Program Editor→ Number of Graphs.. 1→ Graph 2.......... FUNCTION→ Split Screen Ratio 1:1→ Exact/Approx...... AUTO→ Base.............. DEC→ Enter=SAVE ESC=CANCEL
◐ ⊖	 MODE F1 Page 1 F2 Page 2 F3 Page 3 Split Screen...... 1:FULL Split 1 App....... 2:TOP-BOTTOM Split 2 App....... 3:LEFT-RIGHT Number of Graphs.. 1→ Graph 2.......... FUNCTION→ Split Screen Ratio 1:1→ Exact/Approx...... AUTO→ Base.............. DEC→ Enter=SAVE ESC=CANCEL
ENTER	 MODE F1 Page 1 F2 Page 2 F3 Page 3 Split Screen...... TOP-BOTTOM→ Split 1 App....... Home→ Split 2 App....... Program Editor→ Number of Graphs.. 1→ Graph 2.......... FUNCTION→ Split Screen Ratio 1:1→ Exact/Approx...... AUTO→ Base.............. DEC→ Enter=SAVE ESC=CANCEL
ENTER	 F1 F2 Algebra F3 Calc F4 Other F5 PrgmIO F6 Clean Up :program1() :Prgm : :EndPrgm MAIN RAD AUTO FUNC 0/30

Setting the initial Apps for split screen

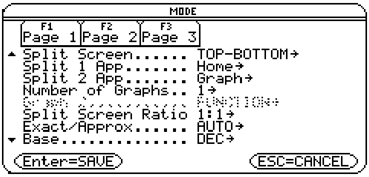
After you select either TOP-BOTTOM or LEFT-RIGHT split-screen mode, additional mode settings become available.

Full-screen mode



```
▲ Split Screen...... 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→
```

Split-screen mode



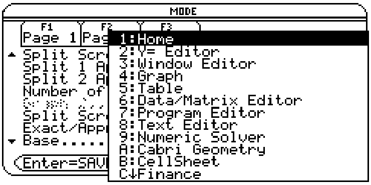
```
▲ Split Screen...... TOP-BOTTOM→
  Split 1 App....... Home→
  Split 2 App....... Graph→
  Number of Graphs.. 1→
  Graph 2........... FUNCTION→
  Split Screen Ratio 1:1→
  Exact/Approx...... AUTO→
▼ Base.............. DEC→
```

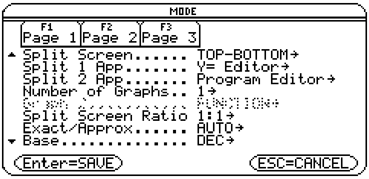
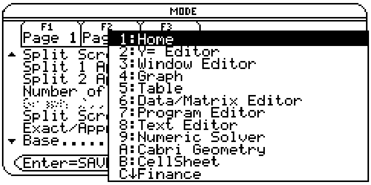
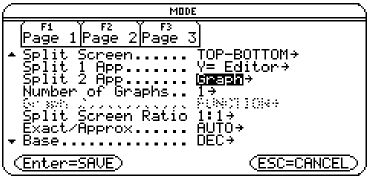
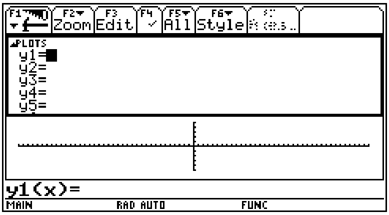
Mode	Description
Split 2 App	Lets you specify the App displayed in the bottom or right portion of the split screen. Works together with Split 1 App, which lets you specify the App displayed in the top or left portion of the split screen.
Number of Graphs	Lets you set up and display two independent graphs.

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 36.)
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
▽ ▷	 MODE 1:Home 2:Y= Editor 3:Window Editor 4:Graph 5:Table 6:Data/Matrix Editor 7:Program Editor 8:Text Editor 9:Numeric Solver A:Cabri Geometry B:CellSheet C↓Finance

Press	Result
2	 MODE F1 Page 1 F2 Page 2 F3 Page 3 Split Screen...... TOP-BOTTOM→ Split 1 App....... Y= Editor→ Split 2 App....... Program Editor→ Number of Graphs.. 1→ Graph............ FUNCTION→ Split Screen Ratio 1:1→ Exact/Approx...... AUTO→ Base.............. DEC→ Enter=SAVE ESC=CANCEL
▼ ▶	 MODE 1:Home 2:Y= Editor 3:Window Editor 4:Graph 5:Table 6:Data/Matrix Editor 7:Program Editor 8:Text Editor 9:Numeric Solver A:Cabri Geometry B:CellSheet C↓Finance
4	 MODE F1 Page 1 F2 Page 2 F3 Page 3 Split Screen...... TOP-BOTTOM→ Split 1 App....... Y= Editor→ Split 2 App....... Graph→ Number of Graphs.. 1→ Graph............ FUNCTION→ Split Screen Ratio 1:1→ Exact/Approx...... AUTO→ Base.............. DEC→ Enter=SAVE ESC=CANCEL
ENTER	 F2 Zoom F3 Edit F4 ✓ F5 All F6 Style PLOTS y1= y2= y3= y4= y5= y1(x)= MAIN RAD AUTO FUNC

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.

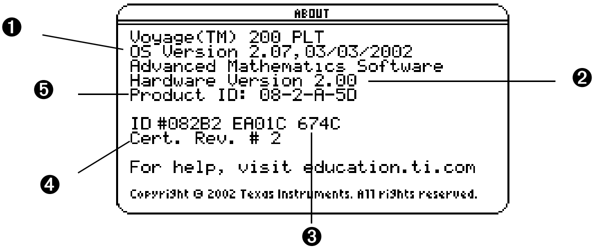
For system requirements and instructions to link to compatible calculators and download TI Connect software, Apps, and OS versions, see the TI E&PS Web site.

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:



```
ABOUT
Voyage(TM) 200 PLT
OS Version 2.07,03/03/2002
Advanced Mathematics Software
Hardware Version 2.00
Product ID: 08-2-A-5D

ID #082B2 EA01C 674C
Cert. Rev. # 2

For help, visit education.ti.com
Copyright © 2002 Texas Instruments. All rights reserved.
```

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

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

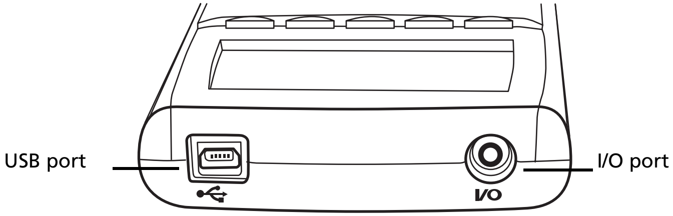
The TI-89 Titanium includes both a mini-USB port and a standard I/O port. Ports are used to link two compatible graphing calculators or connect to a computer or peripheral device.

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.

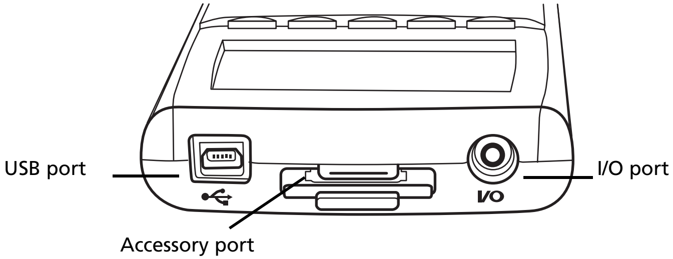
To connect your calculator to a computer – Connect your TI-89 Titanium using the USB port and the included USB computer cable.

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



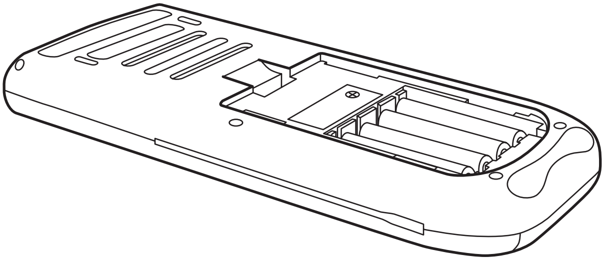
TI-89 Titanium ports (teacher model)

Batteries

The TI-89 Titanium uses four AAA alkaline batteries and a backup silver oxide battery (SR44SW or 303). 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.

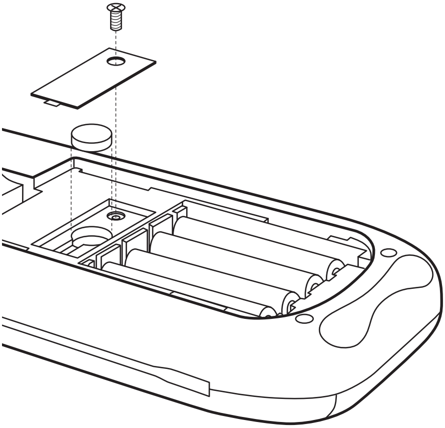
The status line also gives battery information.

Indicator	Meaning
BATT	Batteries are low.
BATT	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 (silver oxide) battery

1. To replace the silver oxide 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 SR44SW or 303 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.

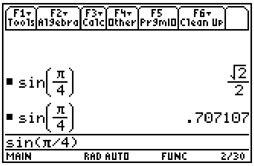
2

Previews

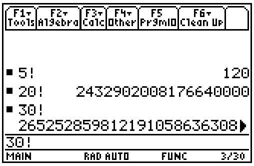
Performing Computations

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.

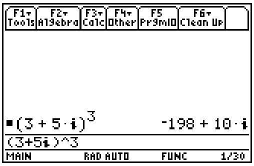
Showing Computations

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] [◆] [≈]	 F1 Tools F2 Algebra F3 Calc F4 Other F5 PrgmIO F6 Clean Up ■ sin(π/4) $\frac{\sqrt{2}}{2}$ ■ sin(π/4) .707107 sin(π/4) MAIN RAD AUTO FUNC 2/30

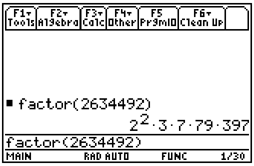
Finding the Factorial of Numbers

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]	 F1 Tools F2 Algebra F3 Calc F4 Other F5 PrgmIO F6 Clean Up ■ 5! 120 ■ 20! 2432902008176640000 ■ 30! 265252859812191058636308▸ 30! MAIN RAD AUTO FUNC 3/30

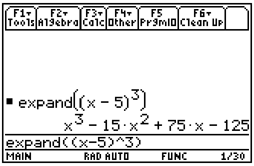
Expanding Complex Numbers

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	 ■ $(3+5\cdot i)^3$ $^{-}198+10\cdot i$ (3+5i)^3

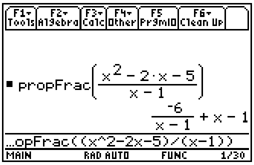
Finding Prime Factors

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.	 ■ factor(2634492) $2^2\cdot 3\cdot 7\cdot 79\cdot 397$ factor(2634492)

Expanding Expressions

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.	 ■ expand$\left((x-5)^3\right)$ $x^3-15\cdot x^2+75\cdot x-125$ expand((x-5)^3)

Reducing Expressions

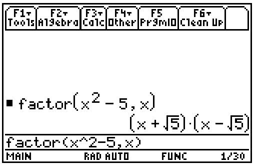
Steps and keystrokes	Display
Reduce the expression $(x^2-2x-5)/(x-1)$ to its simplest form. You can enter "propFrac" on the entry line by typing **PROPFRAC** on the keyboard, or by pressing F2 and selecting **7:propFrac(**. Press F2 7 (X ^ 2 − 2 X − 5) ÷ (X − 1)) ENTER	 ■ propFrac$\left(\frac{x^2-2\cdot x-5}{x-1}\right)$ $\frac{^{-}6}{x-1}+x-1$ ...opFrac((x^2-2x-5)/(x-1))

Factoring Polynomials

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



```
■ factor(x^2 - 5, x)
                 (x + √5)·(x - √5)
factor(x^2-5,x)
MAIN        RAD AUTO      FUNC      1/30
```

Solving Equations

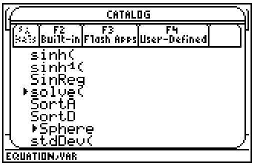
Steps and keystrokes	Display

Solve the equation x^2–2x–6=2 with respect to x.

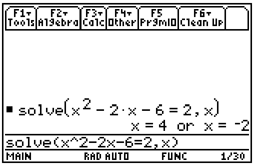
You can enter “**solve(**” on the entry line by selecting ***“*****solve(*****”*** from the Catalog menu, by typing **SOLVE(** on the keyboard, or by pressing F2 and selecting **1:solve(**.

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



```
 sinh(
 sinh⁻¹(
 SinReg
▸solve(
 SortA
 SortD
 ▸Sphere
 stdDev(
EQUATION,VAR
```



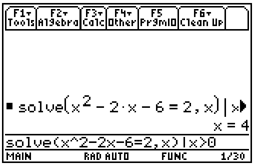
```
■ solve(x^2 - 2·x - 6 = 2, x)
                 x = 4  or  x = ⁻2
solve(x^2-2x-6=2,x)
MAIN        RAD AUTO      FUNC      1/30
```

Solving Equations with a Domain Constraint

Steps and keystrokes	Display

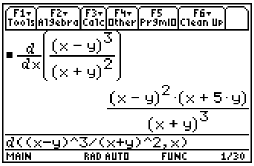
Solve the equation x^2–2x–6=2 with respect to x where x is greater than zero. The “*with*” (I) operator provides domain constraint.

F2 1 X ^ 2 - 2 X - 6 = 2 , X) I X 2nd [>] 0 ENTER

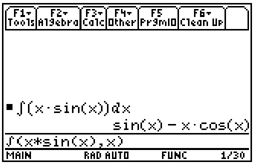


```
■ solve(x^2 - 2·x - 6 = 2, x)|x▸
                             x = 4
solve(x^2-2x-6=2,x)|x>0
MAIN        RAD AUTO      FUNC      1/30
```

Finding the Derivative of Functions

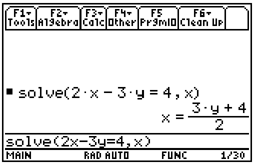
Steps and keystrokes	Display
Find the derivative of (x−y)^3/(x+y)^2 with respect to x. This example illustrates using the calculus differentiation function and how the function is displayed in “pretty print” in the history area. Press 2nd [d] (X − Y) ^ 3 ÷ (X + Y) ^ 2 , X) ENTER	 F1▾ Tools F2▾ Algebra F3▾ Calc F4▾ Other F5 PrgmIO F6▾ Clean Up ■ $\frac{d}{dx}\left(\frac{(x-y)^3}{(x+y)^2}\right)$ $\frac{(x-y)^2\cdot(x+5\cdot y)}{(x+y)^3}$ d((x-y)^3/(x+y)^2,x) MAIN RAD AUTO FUNC 1/30

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	 F1▾ Tools F2▾ Algebra F3▾ Calc F4▾ Other F5 PrgmIO F6▾ Clean Up ■ ∫(x·sin(x))dx sin(x) − x·cos(x) ∫(x*sin(x),x) MAIN RAD AUTO FUNC 1/30

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	 F1▾ Tools F2▾ Algebra F3▾ Calc F4▾ Other F5 PrgmIO F6▾ Clean Up ■ solve(2·x − 3·y = 4, x) x = $\frac{3\cdot y+4}{2}$ solve(2x-3y=4,x) MAIN RAD AUTO FUNC 1/30
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)	

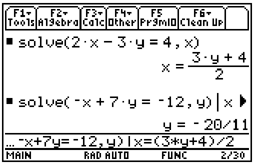
Steps and keystrokes **Display**

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]



■ solve(2·x − 3·y = 4, x) $x = \frac{3\cdot y + 4}{2}$

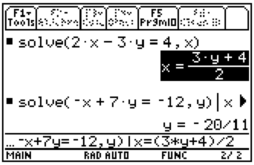
■ solve(⁻x + 7·y = ⁻12, y) | x ▸ y = ⁻20/11

…⁻x+7y=⁻12,y)|x=(3*y+4)/2

MAIN RAD AUTO FUNC 2/30

4. Highlight the equation for x in the history area.

   Press ▲ ▲ ▲



■ solve(2·x − 3·y = 4, x) $x = \frac{3\cdot y + 4}{2}$

■ solve(⁻x + 7·y = ⁻12, y) | x ▸ y = ⁻20/11

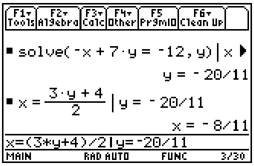
…⁻x+7y=⁻12,y)|x=(3*y+4)/2

MAIN RAD AUTO FUNC 2/2

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



■ solve(⁻x + 7·y = ⁻12, y) | x ▸ y = ⁻20/11

■ $x = \frac{3\cdot y + 4}{2}$ | y = ⁻20/11 x = ⁻8/11

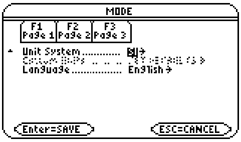
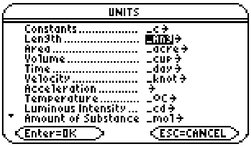
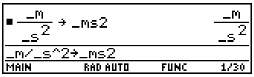
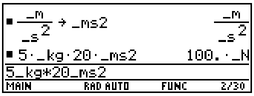
x=(3*y+4)/2|y=⁻20/11

MAIN RAD AUTO FUNC 3/30

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.

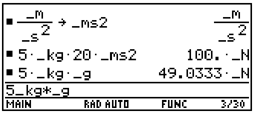
Steps and keystrokes	Display
1. Display the **MODE** dialog box, Page 3. For **Unit System** mode, select **SI** for the metric system of measurements. Results are displayed according to these default units. Press [MODE] [F3] ▶ 1 [ENTER]	 MODE F1 Page 1 · F2 Page 2 · F3 Page 3 Unit System............ SI → Language............... English → Enter=SAVE · ESC=CANCEL
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]	 UNITS Constants.............. _c → Length................. _Å → Area................... _acre → Volume................. _cup → Time................... _day → Velocity............... _knot → Acceleration........... → Temperature............ _°C → Luminous Intensity... _cd → Amount of Substance _mol → Enter=OK · ESC=CANCEL  ■ _m/_s^2 → _ms2 _m/_s^2 _m/_s^2→_ms2 MAIN RAD AUTO FUNC 1/30
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]	 ■ _m/_s^2 → _ms2 _m/_s^2 ■ 5·_kg·20·_ms2 100.·_N 5_kg*20_ms2 MAIN RAD AUTO FUNC 2/30

Steps and keystrokes **Display**

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

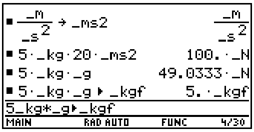


```
■ _m/_s^2 ▸ _ms2                    _m/_s^2
■ 5·_kg·20·_ms2                  100.·_N
■ 5·_kg·_g                    49.0333·_N
5_kg*_g
MAIN          RAD AUTO      FUNC      3/30
```

5. Convert to kilograms of force (_kgf).

   2nd [▸] displays the ▸ conversion operator.

   🖩 ▶ 2nd [▸] ♦ [_] 2nd [a-lock] KGF alpha ENTER



```
■ _m/_s^2 ▸ _ms2                    _m/_s^2
■ 5·_kg·20·_ms2                  100.·_N
■ 5·_kg·_g                    49.0333·_N
■ 5·_kg·_g ▸ _kgf              5.·_kgf
5_kg*_g▸_kgf
MAIN          RAD AUTO      FUNC      4/30
```

Basic Function Graphing I

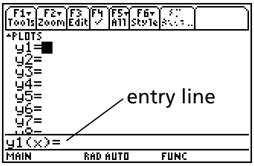
The example in this section demonstrates some of the graphing capabilities of the TI-89 Titanium keystrokes. It illustrates how to graph a function using the **Y= Editor**. You will learn how to enter a function, produce a graph of the function, trace a curve, find a minimum point, and transfer the minimum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

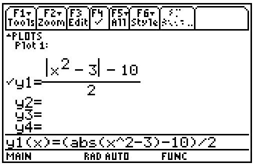


```
▴PLOTS
 y1=█
 y2=
 y3=
 y4=
 y5=
 y6=
 y7=
y1(x)=
MAIN          RAD AUTO      FUNC
```

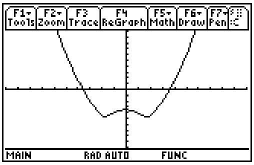
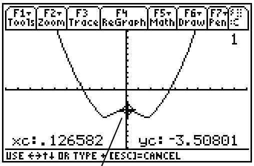
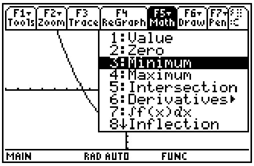
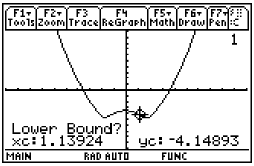
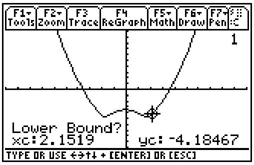
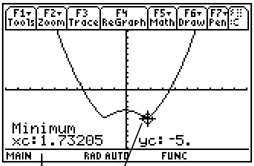
2. Enter the function **(abs(x^2−3)−10)/2**.

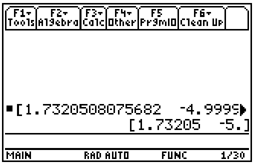
   The screen shot shows the “pretty print” display at **y1=**.

   🖩 (CATALOG A ENTER X ^ 2 − 3) − 1 0) ÷ 2 ENTER



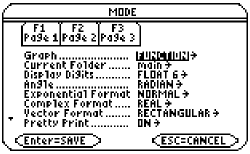
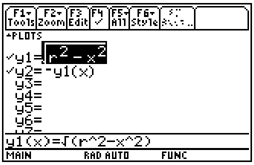
```
▴PLOTS
 Plot 1:
✓y1=(|x^2 - 3| - 10)/2
 y2=
 y3=
 y4=
y1(x)=(abs(x^2-3)-10)/2
MAIN          RAD AUTO      FUNC
```

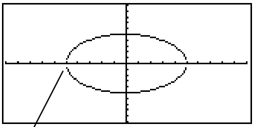
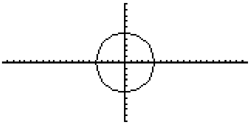
Steps and keystrokes	Display
3. Display the graph of the function. Select **6:ZoomStd** by pressing **6** or by moving the cursor to **6:ZoomStd** and pressing [ENTER]. Press [F2] 6	 MAIN RAD AUTO FUNC
4. Turn on **Trace**. The tracing cursor, and the x and y coordinates are displayed. Press [F3]	 F1 Tools F2 Zoom F3 Trace F4 ReGraph F5 Math F6 Draw F7 Pen 1 xc:.126582 yc:-3.50801 USE ←→↑↓ OR TYPE + [ESC]=CANCEL tracing cursor
5. Open the **MATH** menu and select **3:Minimum**. Press [F5] ▼ ▼ [ENTER]	 F1 Tools F2 Zoom F3 Trace F4 ReGraph F5 Math F6 Draw F7 Pen 1:Value 2:Zero 3:Minimum 4:Maximum 5:Intersection 6:Derivatives▸ 7:∫f(x)dx 8↓Inflection MAIN RAD AUTO FUNC
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]	 F1 Tools F2 Zoom F3 Trace F4 ReGraph F5 Math F6 Draw F7 Pen 1 Lower Bound? xc:1.13924 yc:-4.14893 MAIN RAD AUTO FUNC
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 ▶ ... ▶	 F1 Tools F2 Zoom F3 Trace F4 ReGraph F5 Math F6 Draw F7 Pen 1 Lower Bound? xc:2.1519 yc:-4.18467 TYPE OR USE ←→↑↓ + [ENTER] OR [ESC]
8. Find the minimum point on the graph between the lower and upper bounds. Press [ENTER]	 Minimum xc:1.73205 yc:-5. MAIN RAD AUTO FUNC minimum point minimum coordinates

Steps and keystrokes	Display
9. Transfer the result to the Home screen, and then display the Home screen. ◆ (-) HOME	 F1 Tools F2 Algebra F3 Calc F4 Other F5 PrgmIO F6 Clean Up ■[1.7320508075682 ⁻4.9999▸ [1.73205 ⁻5.] MAIN RAD AUTO FUNC 1/30

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.

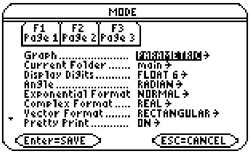
Steps and keystrokes	Display
1. Display the **MODE** dialog box. For **Graph** mode, select **FUNCTION**. Press MODE ⊳ 1 ENTER	 MODE F1 Page 1 F2 Page 2 F3 Page 3 Graph .... FUNCTION → Current Folder .... main → Display Digits .... FLOAT 6 → Angle .... RADIAN → Exponential Format NORMAL → Complex Format .... REAL → Vector Format .... RECTANGULAR → Pretty Print .... ON → Enter=SAVE ESC=CANCEL
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	 F1 Tools F2 Zoom F3 Edit F4 ✓ F5 All F6 Style ▸PLOTS ✓y1=$\sqrt{r^2 - x^2}$ ✓y2=⁻y1(x) y3= y4= y5= y6= y1(x)=√(r^2−x^2) MAIN RAD AUTO FUNC

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.

Parametric Graphing

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	 MODE F1 Page 1 F2 Page 2 F3 Page 3 Graph........................ PARAMETRIC → Current Folder .... main → Display Digits.......... FLOAT 6 → Angle .... RADIAN → Exponential Format NORMAL → Complex Format..... REAL → Vector Format........ RECTANGULAR → Pretty Print........... ON → Enter=SAVE ESC=CANCEL

Steps and keystrokes | **Display**

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.

   Display:

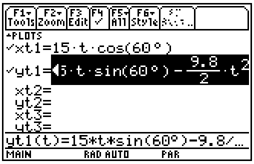
   xt1(t)=15t*cos(60°)

3. Define the vertical component yt1(t) = v_0t sin θ − (g/2)t^2.

   Enter values for v_0, θ, and g.

   [ENTER] 15T [×] [2nd] [SIN] 60 [2nd] [°] [)] [−] [(] 9.8 [÷] 2 [)] T [^] 2 [ENTER]

   Display:



   +PLOTS
   ✓xt1=15·t·cos(60°)
   ✓yt1=15·t·sin(60°) − 9.8/2·t²
   xt2=
   yt2=
   xt3=
   yt3=
   yt1(t)=15*t*sin(60°)-9.8/…

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

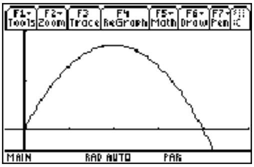
   Display:

   ```
   tmin=0.
   tmax=3.
   tstep=.02
   xmin=-2.
   xmax=25.
   xscl=5.
   ymin=-2.
   ymax=10.
   yscl=5.
   ```

5. Graph the parametric equations to model the path of the ball.

   Press [◆] [GRAPH]

   Display:

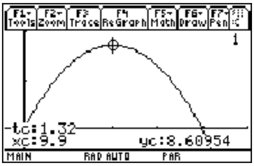


6. Select **Trace**. Then move the cursor along the path to find the:

   - y value at maximum height.
   - t value where the ball hits the ground.

   Press [F3] ▶ or ◀ as necessary

   Display:



   tc:1.32
   xc:9.9
   yc:8.60954

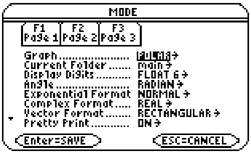
Polar Graphing

The graph of the polar equation r1(θ) = A sin B θ 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]

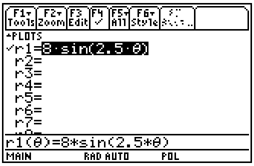


```
MODE
F1 Page 1 | F2 Page 2 | F3 Page 3
Graph.................. POLAR →
Current Folder....... main →
Display Digits.......... FLOAT 6 →
Angle.................... RADIAN →
Exponential Format NORMAL →
Complex Format..... REAL →
Vector Format....... RECTANGULAR →
Pretty Print........... ON →
Enter=SAVE          ESC=CANCEL
```

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]

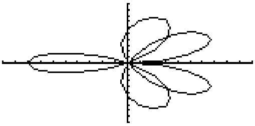


```
F1▾ Tools | F2▾ Zoom | F3 Edit | F4 ✓ | F5▾ All | F6▾ Style
▲PLOTS
✓r1=8·sin(2.5·θ)
 r2=
 r3=
 r4=
 r5=
 r6=
 r7=
r1(θ)=8*sin(2.5*θ)
MAIN          RAD AUTO        POL
```

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.

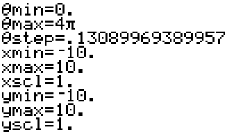
   Press [F2] 6



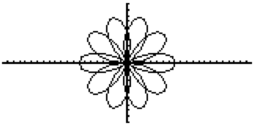
4. Display the **Window Editor**, and change θmax to 4π.

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

   Press [♦] [WINDOW] ▼ 4 [2nd] [π]



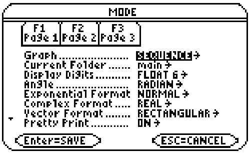
```
θmin=0.
θmax=4π
θstep=.13089969389957
xmin=⁻10.
xmax=10.
xscl=1.
ymin=⁻10.
ymax=10.
yscl=1.
```

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]	 MODE Graph .... SEQUENCE → Current Folder .... main → Display Digits .... FLOAT 6 → Angle .... RADIAN → Exponential Format NORMAL → Complex Format .... REAL → Vector Format .... RECTANGULAR → Pretty Print .... ON → F1 Page 1 F2 Page 2 F3 Page 3 Enter=SAVE ESC=CANCEL

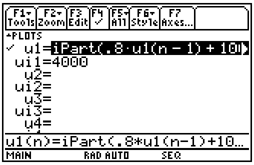
Steps and keystrokes | **Display**

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] 14.8 [alpha] U1 [(] [alpha] N [−] 1 [)] [+] 1000 [)] [ENTER]



```
▲PLOTS
✓ u1=iPart(.8·u1(n − 1) + 10▸
 ui1=4000
  u2=
 ui2=
  u3=
 ui3=
  u4=
u1(n)=iPart(.8*u1(n−1)+10...
MAIN          RAD AUTO       SEQ
```

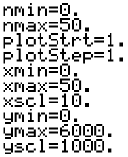
3. Define ui1 as the initial value that will be used as the first term.

   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 ▼



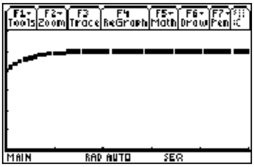
```
nmin=0.
nmax=50.
plotStrt=1.
plotStep=1.
xmin=0.
xmax=50.
xscl=10.
ymin=0.
ymax=6000.
yscl=1000.
```

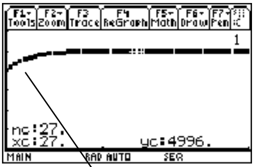
5. Set the x and y Window variables to appropriate values for this example.

   Press 0 ▼ 50 ▼ 10 ▼ 0 ▼ 6000 ▼ 1000

6. Display the Graph screen.

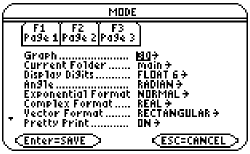
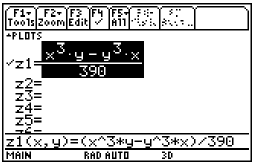
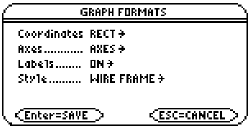
   Press [◆] [GRAPH]



Steps and keystrokes	**Display**
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	 F1 Tools F2 Zoom F3 Trace F4 ReGraph F5 Math F6 Draw F7 Pen 1 nc:27. xc:27. yc:4996. MAIN RAD AUTO SEQ By default, sequences use the Square display style.

3D Graphing

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]	MODE Graph .... 3D → Current Folder .... main → Display Digits .... FLOAT 6 → Angle .... RADIAN → Exponential Format NORMAL → Complex Format ..... REAL → Vector Format .... RECTANGULAR → Pretty Print .... ON →
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]	 PLOTS ✓z1=$\frac{x^3 \cdot y - y^3 \cdot x}{390}$ z2= z3= z4= z5= z1(x,y)=(x^3*y-y^3*x)/390 MAIN RAD AUTO 3D
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. 🖩 [◆] [|] ▼ ▶ 2 ▼ ▶ 2 ▼ ▶ 1 [ENTER]	GRAPH FORMATS Coordinates RECT → Axes .... AXES → Labels .... ON → Style .... WIRE FRAME →

Steps and keystrokes **Display**

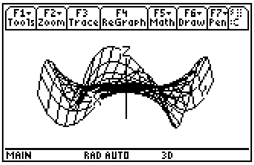
4. Select the **ZoomStd** viewing cube, which automatically graphs the equation.

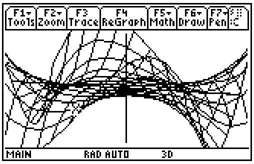
   As the equation is evaluated (before it is graphed), "evaluation percentages" are shown in the upper-left part of the screen.

   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)



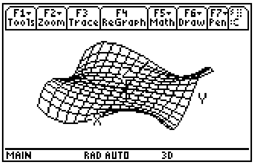


5. Animate the graph by decreasing the eyeϕ Window variable value.

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

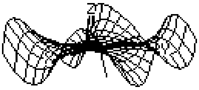
   To animate the graph continuously, press and hold the cursor for about 1 second and then release it. To stop, press ENTER.

   Press ▼ eight times



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) ◀ ◀ ◀



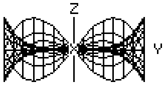
7. View the graph along the x axis, the y axis, and then the z axis.

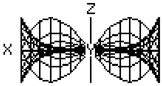
   Press X

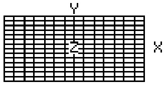
   This graph has the same shape along the y axis and x axis.

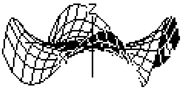
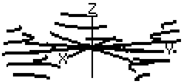
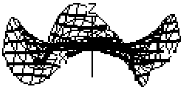
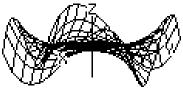
   Press Y

   Press Z





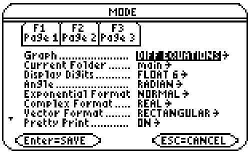


Steps and keystrokes	Display
8. Return to the initial orientation. Press 0 (zero)	
9. Display the graph in different graph format styles. 🖩 [I] (press [I] to switch from each style to the next)	 HIDDEN SURFACE  CONTOUR LEVELS (may require extra time to calculate contours)  WIRE AND CONTOUR  WIRE FRAME

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]	 MODE Graph.................. DIFF EQUATIONS → Current Folder....... main → Display Digits.......... FLOAT 6 → Angle...................... RADIAN → Exponential Format NORMAL → Complex Format..... REAL → Vector Format........ RECTANGULAR → Pretty Print........... ON →

Steps and keystrokes | **Display**

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

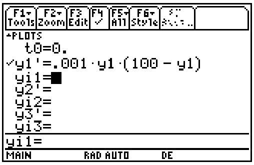
   y1'(t)=.001y1∗(100−y1)

   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

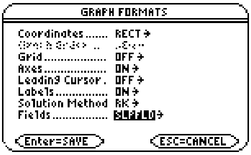


```
PLOTS
 t0=0.
✓y1'=.001·y1·(100 − y1)
 yi1=█
 y2'=
 yi2=
 y3'=
 yi3=
yi1=
```

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.

   ◆ | ▼ ▼ ▶ 2 ▼ ▼ ▶ 2 ▼ ▶ 1 ▼ ▶ 1 ENTER

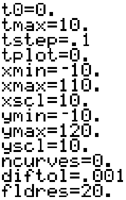


GRAPH FORMATS

Coordinates	RECT →
Graph Order	SEQ
Grid	OFF →
Axes	ON →
Leading Cursor	OFF →
Labels	ON →
Solution Method	RK →
Fields	SLPFLD →

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

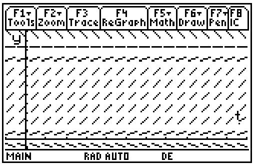


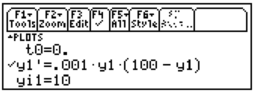
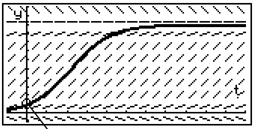
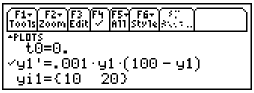
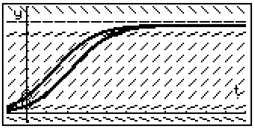
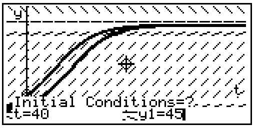
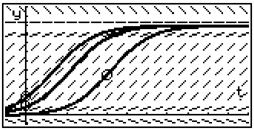
```
t0=0.
tmax=10.
tstep=.1
tplot=0.
xmin=-10.
xmax=110.
xscl=10.
ymin=-10.
ymax=120.
yscl=10.
ncurves=0.
diftol=.001
fldres=20.
```

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).

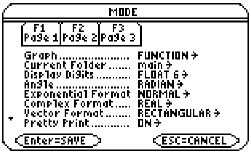
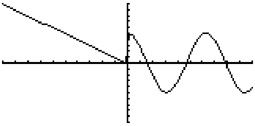
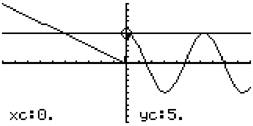
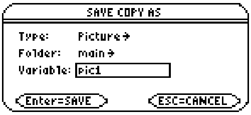
   Press ◆ [GRAPH]

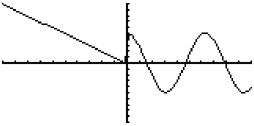
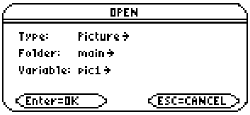
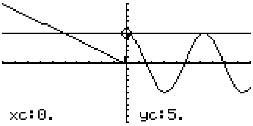


Steps and keystrokes	Display
6. Return to the **Y= Editor** and enter an initial condition: **yi1=10** Press [♦] [Y=] [ENTER] 10 [ENTER]	 F1▾ Tools F2▾ Zoom F3 Edit F4 ✓ F5▾ All F6▾ Style▸PLOTS t0=0. ✓y1'=.001·y1·(100 – y1) yi1=10
7. Return to the Graph screen. Initial conditions entered in the **Y= 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.
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]	 F1▾ Tools F2▾ Zoom F3 Edit F4 ✓ F5▾ All F6▾ Style▸PLOTS t0=0. ✓y1'=.001·y1·(100 – y1) yi1={10 20}
9. Return to the Graph screen. Press [♦] [GRAPH]	
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]	 Initial Conditions=? t=40 y1=45 

Additional Graphing Topics

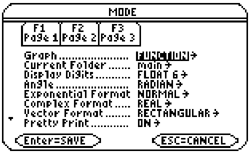
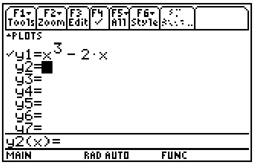
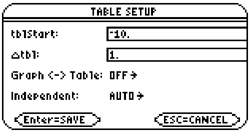
From the Home screen, graph the piecewise defined function: y = ⁻x when x < 0 and y = 5 cos(x) when x ≥ 0. Draw a horizontal line across the top of the cosine curve. Then save a picture of the displayed graph.

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]	 MODE F1 Page 1 · F2 Page 2 · F3 Page 3 Graph....................... FUNCTION → Current Folder....... main → Display Digits.......... FLOAT 6 → Angle....................... RADIAN → Exponential Format NORMAL → Complex Format..... REAL → Vector Format....... RECTANGULAR → Pretty Print........... ON →
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]	
4. Draw a horizontal line across the top of the cosine curve. The calculator remains in "line" 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]	 SAVE COPY AS Type: Picture → Folder: main → Variable: pic1

Steps and keystrokes	Display
6. Clear the drawn horizontal line. You can also press [F4] to regraph. [2nd] [F6] 1	
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]	 OPEN Type: Picture→ Folder: main→ Variable: pic1→ Enter=OK ESC=CANCEL  xc:0. yc:5.

Tables

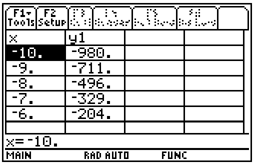
Evaluate the function y=x^3–2x at each integer between ⁻10 and 10. How many sign changes are there, and where do they occur?

Steps and keystrokes	Display
1. Display the **MODE** dialog box. For the **Graph** mode, select **FUNCTION**. Press [MODE] ▶ 1 [ENTER]	 MODE F1 Page 1 F2 Page 2 F3 Page 3 Graph.................. FUNCTION→ Current Folder....... main→ Display Digits........ FLOAT 6→ Angle.................. RADIAN→ Exponential Format NORMAL→ Complex Format..... REAL→ Vector Format....... RECTANGULAR→ Pretty Print.......... ON→ Enter=SAVE ESC=CANCEL
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]	 F1▾ Tools F2▾ Zoom F3 Edit F4 ✓ F5▾ All F6▾ Style F7 ▸PLOTS ✓y1=x^3 – 2·x y2= y3= y4= y5= y6= y7= y2(x)= MAIN RAD AUTO FUNC
3. Set the table parameters to: **tblStart = ⁻10** **∆tbl = 1** **Graph < - > Table = OFF** **Independent = AUTO** Press [◆] [TBLSET] [(-)] 10 ▼ 1 ▼ ▶ 1 ▼ ▶ 1 [ENTER]	 TABLE SETUP tblStart: ⁻10. ∆tbl: 1. Graph <-> Table: OFF→ Independent: AUTO→ Enter=SAVE ESC=CANCEL

Steps and keystrokes | **Display**

4. Display the Table screen.

   Press [◆] [TABLE]



x	y1
⁻10.	⁻980.
⁻9.	⁻711.
⁻8.	⁻496.
⁻7.	⁻329.
⁻6.	⁻204.

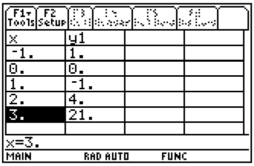
x=⁻10.

MAIN RAD AUTO FUNC

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



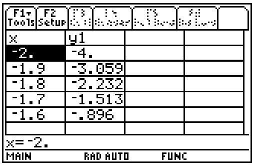
x	y1
⁻1.	1.
0.	0.
1.	⁻1.
2.	4.
3.	21.

x=3.

MAIN RAD AUTO FUNC

6. Zoom in on the sign change between x = ⁻2 and x = ⁻1 by changing the table parameters to:
   tblStart = ⁻2
   ∆tbl = .1

   Press [F2] [(-)] 2 ▼ .1 [ENTER] [ENTER]



x	y1
⁻2.	⁻4.
⁻1.9	⁻3.059
⁻1.8	⁻2.232
⁻1.7	⁻1.513
⁻1.6	⁻.896

x=⁻2.

MAIN RAD AUTO FUNC

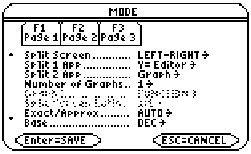
Split Screens

Split the screen to show the **Y= Editor** and the Graph screen. Then 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]



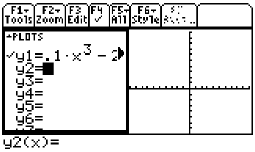
MODE

Split Screen........... LEFT-RIGHT →
Split 1 App.............. Y= Editor →
Split 2 App.............. Graph →
Number of Graphs.. 1 →
Exact/Approx........ AUTO →
Base........................ DEC →

2. Clear the **Y= Editor** and turn off any stat data plots. Then define y1(x) = .1x^3−2x+6.

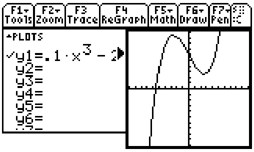
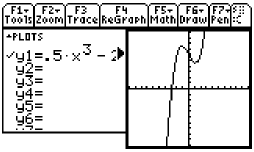
   A thick border around the **Y= Editor** indicates it is active. When active, its entry line goes all the way across the display.

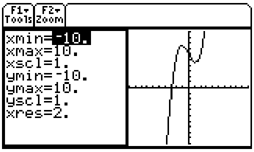
   Press [F1] 8 [ENTER] [F5] 5 [ENTER] .1 X [^] 3 [−] 2 X [+] 6 [ENTER]



▲PLOTS
✓y1=.1·x^3 − 2▶
y2=█
y3=
y4=
y5=
y6=

y2(x)=

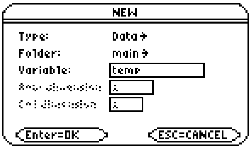
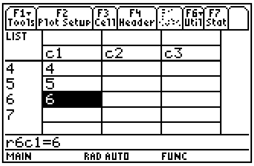
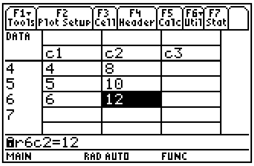
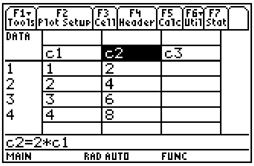
Steps and keystrokes	Display
3. Select the **ZoomStd** viewing window, which switches to the Graph screen and graphs the function. The thick border is now around the Graph screen. Press F2 6	 F1 Tools F2 Zoom F3 Trace F4 ReGraph F5 Math F6 Draw F7 Pen PLOTS ✓y1=.1·x^3 − 2 y2= y3= y4= y5= y6=
4. Switch to the **Y= Editor**. Then 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	
5. Switch to the Graph screen, which regraphs the edited function. The thick border is around the Graph screen. Press 2nd [⊞]	 F1 Tools F2 Zoom F3 Trace F4 ReGraph F5 Math F6 Draw F7 Pen PLOTS ✓y1=.5·x^3 − 2 y2= y3= y4= y5= y6=

6. Switch to the **Y= Editor**. Then open the **Window Editor** in its place.
Press 2nd [⊞] ◆ [WINDOW]
	
	xmin=-10.
	xmax=10.
	xscl=1.
	ymin=-10.
	ymax=10.
	yscl=1.
	xres=2.

7. Open the Home screen. Then exit to a full-sized Home screen. Press 2nd [QUIT] twice.	

Data/Matrix Editor

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).

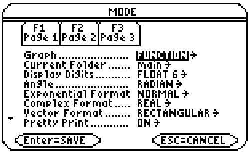
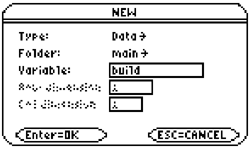
Steps and keystrokes	**Display**
1. Use [APPS] to display the **Data/Matrix Editor.** Create a new list variable named **TEMP**. Press 3 ▶ 3 ▼ ▼ TEMP [ENTER] [ENTER]	 NEW Type: Data→ Folder: main→ Variable: temp Row dimension: Col dimension: Enter=OK ESC=CANCEL
2. Enter a column of numbers. Then move 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] ▲	 LIST c1 c2 c3 4 4 5 5 6 6 7 r6c1=6 MAIN RAD AUTO FUNC
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]	 DATA c1 c2 c3 4 4 8 5 5 10 6 6 12 7 🔒r6c2=12 MAIN RAD AUTO FUNC 🔒 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] ▲ ▲	 DATA c1 c2 c3 1 1 2 2 2 4 3 3 6 4 4 8 c2=2*c1 MAIN RAD AUTO FUNC

Steps and keystrokes	Display
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 creating a new variable each time, which uses up memory.

Statistics and Data Plots

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.

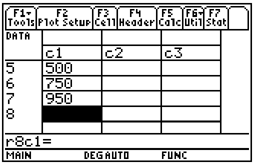
Steps and keystrokes	Display
1. Display the **MODE** dialog box. For **Graph** mode, select **FUNCTION**. Press [MODE] ⓡ 1 [ENTER]	 MODE F1 Page 1 · F2 Page 2 · F3 Page 3 Graph.................. FUNCTION → Current Folder....... main → Display Digits.......... FLOAT 6 → Angle...................... RADIAN → Exponential Format NORMAL → Complex Format..... REAL → Vector Format........ RECTANGULAR → Pretty Print........... ON → Enter=SAVE · ESC=CANCEL
2. Use [APPS] to display the **Data/Matrix Editor**. Create a new data variable named **BUILD**. Press 3 ⓓ ⓓ BUILD [ENTER] [ENTER]	 NEW Type: Data → Folder: main → Variable: build Row dimension: 1 Col dimension: 1 Enter=OK · ESC=CANCEL

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]



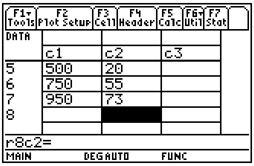
	c1	c2	c3
5	500		
6	750		
7	950		
8			

r8c1=

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]



	c1	c2	c3
5	500	20	
6	750	55	
7	950	73	
8			

r8c2=

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

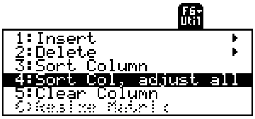
This sorts column 1 and then adjusts all other columns so that they retain the same order as column 1. This is critical for maintaining the relationships between columns of data.

To sort column 1, the cursor can be anywhere in column 1. This example has you press

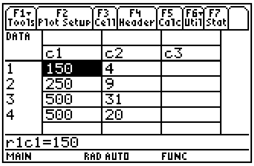
🖩 [◆] ▲

so that you can see the first four rows.

🖩 ◀ [◆] ▲ [2nd] [F6] 4



1:Insert ▸
2:Delete ▸
3:Sort Column
4:Sort Col, adjust all
5:Clear Column
6:Resize Matrix



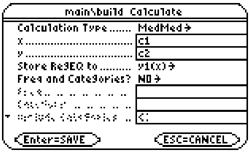
	c1	c2	c3
1	150	4	
2	250	9	
3	500	31	
4	500	20	

r1c1=150

Steps and keystrokes **Display**

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]

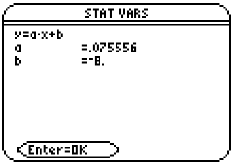


```
main\build Calculate
Calculation Type....... MedMed→
x...................... c1
y...................... c2
Store RegEQ to......... y1(x)→
Freq and Categories?    NO→
Enter=SAVE              ESC=CANCEL
```

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]



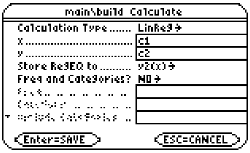
```
STAT VARS
y=a·x+b
a          =.075556
b          =⁻8.
Enter=OK
```

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]

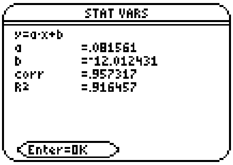


```
main\build Calculate
Calculation Type....... LinReg→
x...................... c1
y...................... c2
Store RegEQ to......... y2(x)→
Freq and Categories?    NO→
Enter=SAVE              ESC=CANCEL
```

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

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

Press [ENTER]



```
STAT VARS
y=a·x+b
a          =.081561
b          =⁻12.012431
corr       =.957317
R²         =.916457
Enter=OK
```

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

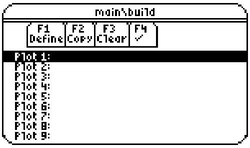
Press [ENTER]

12. Display the Plot Setup screen.

Plot 1 is highlighted by default.

[F3] lets you clear highlighted Plot settings.

Press [F2]



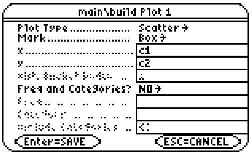
```
main\build
F1      F2    F3    F4
Define  Copy  Clear ✓
Plot 1:
Plot 2:
Plot 3:
Plot 4:
Plot 5:
Plot 6:
Plot 7:
Plot 8:
Plot 9:
```

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

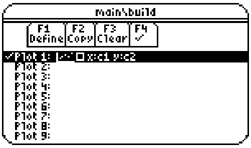


main\build Plot 1
Plot Type .... Scatter →
Mark .... Box →
x c1
y c2
Freq and Categories? NO →

14. Save the plot definition and return to the Plot Setup screen.

Notice the shorthand notation for **Plot 1's** definition.

Press [ENTER] twice



main\build
✓Plot 1: □ x:c1 y:c2
Plot 2:
Plot 3:
Plot 4:
Plot 5:
Plot 6:
Plot 7:
Plot 8:
Plot 9:

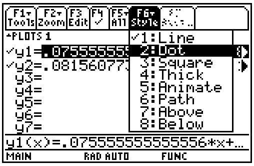
15. Display the **Y= Editor**. For **y1(x)**, the MedMed regression equation, set the display style to **Dot**.

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

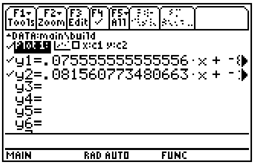


PLOTS 1
✓y1=.07555555
✓y2=.08156077
y3=
y4=
y5=
y6=
y7=
y1(x)=.0755555555555556*x+...
MAIN RAD AUTO FUNC

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

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

Press ↑

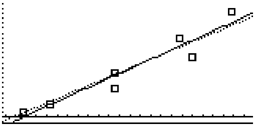


DATA:main\build
✓Plot 1 □ x:c1 y:c2
✓y1=.0755555555555556 · x + -
✓y2=.081560773480663 · x + -
y3=
y4=
y5=
y6=
MAIN RAD AUTO FUNC

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



Steps and keystrokes **Display**

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

    Press [2nd][⊞]

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

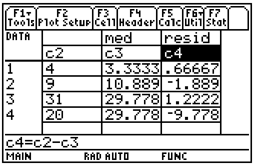
    To enter a title, the cursor must highlight the title cell at the very top of the column.

    [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]

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]



DATA		med	resid
	c2	c3	c4
1	4	3.3333	.66667
2	9	10.889	-1.889
3	31	29.778	1.2222
4	20	29.778	-9.778

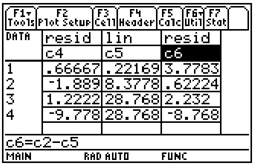
c4=c2-c3

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]



DATA	resid	lin	resid
	c4	c5	c6
1	.66667	.22169	3.7783
2	-1.889	8.3778	.62224
3	1.2222	28.768	2.232
4	-9.778	28.768	-8.768

c6=c2-c5

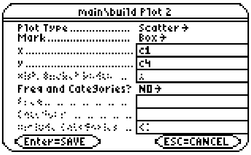
23. Display the Plot Setup screen and deselect **Plot 1**.

    Press [F2] [F4]

Steps and keystrokes | **Display**

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]

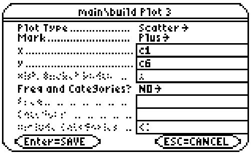


main\build Plot 2

Plot Type .... Scatter →
Mark .... Box →
x c1
y c4
Freq and Categories? NO →

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]



main\build Plot 3

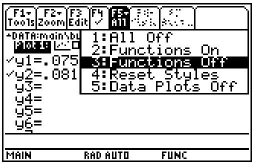
Plot Type .... Scatter →
Mark .... Plus →
x c1
y c6
Freq and Categories? NO →

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



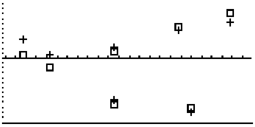
▲DATA:main\bu
✓y1=.075
✓y2=.081
y3=
y4=
y5=
y6=

MAIN RAD AUTO FUNC

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

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

Press [F2] 9



28. Display the Home screen.

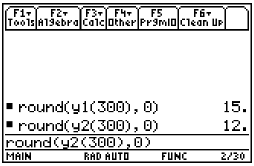
🖩 [HOME]

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**.

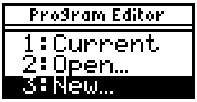
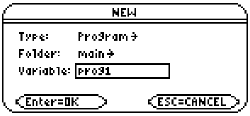
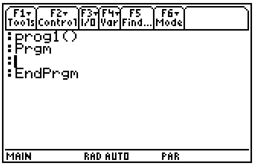
Press [2nd] [MATH] 1 3 Y1 [(] 300 [)] [,] 0 [)] [ENTER] ▶ ◀ (eight times) [←] 2 [ENTER]

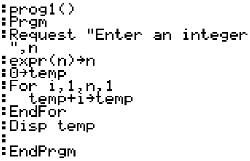


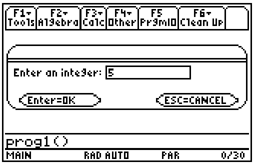
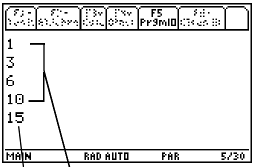
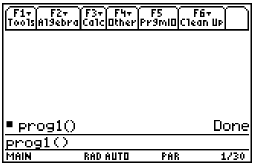
```
■ round(y1(300),0)          15.
■ round(y2(300),0)          12.
round(y2(300),0)
MAIN     RAD AUTO     FUNC     2/30
```

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.

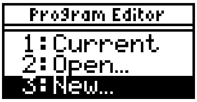
Steps and keystrokes	Display
1. Use [APPS] to display the **Program Editor**. Create a new program. Press 3	
2. Type **PROG1** (with no spaces) as the name of the new program variable. 🖩 ⏷ ⏷ PROG [alpha] 1	 Type: Program → Folder: main → Variable: prog1
3. Display the "template" for a new program. The 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	 F1 Tools F2 Control F3 I/O F4 Var F5 Find... F6 Mode :prog1() :Prgm : :EndPrgm

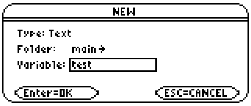
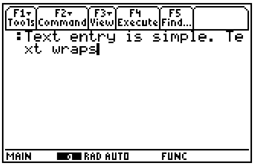
Steps and keystrokes	Display
4. Type the following program lines. **Request "Enter an integer",n** Displays a dialog box that prompts "Enter an integer", waits for the user to enter a value, and stores it (as a string) to variable n. **expr(n)→n** Converts the string to a numeric expression. **0→temp** 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.	 :prog1() :Prgm :Request "Enter an integer ",n :expr(n)→n :0→temp :For i,1,n,1 : temp+i→temp :EndFor :Disp temp : :EndPrgm
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()

Steps and keystrokes	Display
6. Type 5 in the displayed dialog box. Press 5	 F1▾ Tools F2▾ Algebra F3▾ Calc F4▾ Other F5 PrgmIO F6▾ Clean Up Enter an integer: 5 prog1() MAIN RAD AUTO PAR 0/30
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	 1 3 6 10 15 MAIN RAD AUTO PAR 5/30 Output from other programs may still be on the screen. Result of integer 5
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]	 F1▾ Tools F2▾ Algebra F3▾ Calc F4▾ Other F5 PrgmIO F6▾ Clean Up ■ prog1() Done prog1() MAIN RAD AUTO PAR 1/30

Text Operations

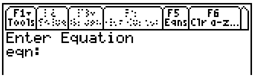
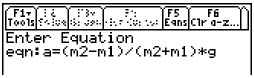
Start a new **Text Editor** session. Then practice using the **Text Editor** by typing whatever text you want. As you type, practice moving the text cursor and correcting any typos you may enter.

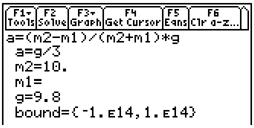
Steps and keystrokes	Display
1. Start a new session of the **Text Editor**. Press 3	 Program Editor 1:Current 2:Open… 3:New…

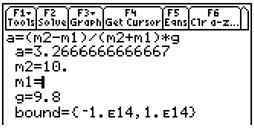
Steps and keystrokes	Display
2. Create a text variable called **TEST**, which will automatically store any text you enter in the new session. Use the **MAIN** folder, shown as the default on the **NEW** dialog box. After typing in an input box such as **Variable**, you must press [ENTER] twice. Press ▼ TEST [ENTER] [ENTER]	 NEW Type: Text Folder: main→ Variable: test
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	 F1▾ Tools F2▾ Command F3▾ View F4 Execute F5 Find... :Text entry is simple. Te xt wraps
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]	
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][⊞]	

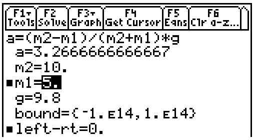
Numeric Solver

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.

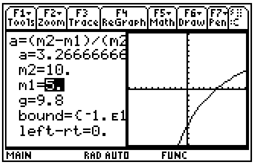
Steps and keystrokes		Display
1.	Use APPS to display the **Numeric Solver**.	 Enter Equation eqn:
2.	Enter the equation. When you press ENTER or ▼, the screen lists the variables used in the equation. 🖩 alpha A = (alpha M2 − alpha M1) ÷ (alpha M2 + alpha M1) × alpha G ENTER	 Enter Equation eqn:a=(m2−m1)/(m2+m1)*g

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
		
		 F1 Tools F2 Solve F3 Graph F4 Get Cursor F5 Eqns F6 Clr a-z…
		a=(m2−m1)/(m2+m1)*g
		a=g/3
		m2=10.
		m1=
		g=9.8
		bound={⁻1.ᴇ14,1.ᴇ14}

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 ▼ ▼
		
		 F1 Tools F2 Solve F3 Graph F4 Get Cursor F5 Eqns F6 Clr a-z…
		a=(m2−m1)/(m2+m1)*g
		a=3.2666666666667
		m2=10.
		m1=
		g=9.8
		bound={⁻1.ᴇ14,1.ᴇ14}
		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
		
		 F1 Tools F2 Solve F3 Graph F4 Get Cursor F5 Eqns F6 Clr a-z…
		a=(m2−m1)/(m2+m1)*g
		a=3.2666666666667
		m2=10.
		■m1=5.
		g=9.8
		bound={⁻1.ᴇ14,1.ᴇ14}
		■left-rt=0.
		■ marks the calculated values.

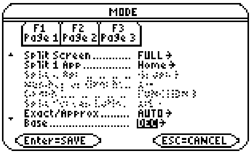
Steps and keystrokes	Display

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
	 F1 Tools
	F2 Zoom
	F3 Trace
	F4 ReGraph
	F5 Math
	F6 Draw
	F7 Pen
	a=(m2-m1)/(m2
	a=3.2666666
	m2=10.
	m1=5.
	g=9.8
	bound={-1.E1
	left-rt=0.

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]	 Split Screen............ FULL → Split 1 App.............. Home → [illegible] Exact/Approx......... AUTO → Base......................... DEC →

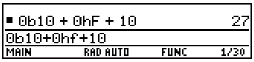
Steps and keystrokes | **Display**

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]



```
▪ 0b10 + 0hF + 10                27
0b10+0hf+10
MAIN        RAD AUTO    FUNC   1/30
```

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

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

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

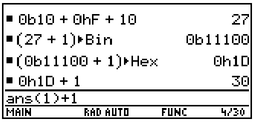
4. Add 1 to the result and convert it to hexadecimal.

   🖩 [+] 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]



```
▪ 0b10 + 0hF + 10                27
▪(27 + 1)▸Bin               0b11100
▪(0b11100 + 1)▸Hex             0h1D
▪ 0h1D + 1                       30
ans(1)+1
MAIN        RAD AUTO    FUNC   4/30
```

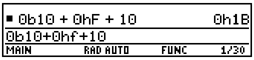
6. Change the **Base** mode to **HEX**.

   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.

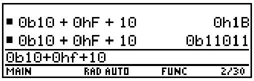
   🖩 0 [alpha] B 10 [+] 0 [2nd] [a-lock] HF [alpha] [+] 10 [ENTER]



```
▪ 0b10 + 0hF + 10              0h1B
0b10+0hf+10
MAIN        RAD AUTO    FUNC   1/30
```

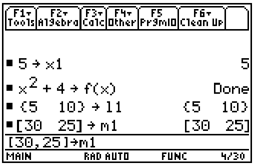
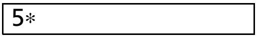
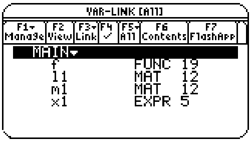
8. Change the **Base** mode to **BIN**.

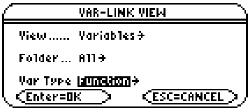
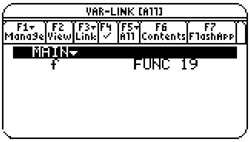
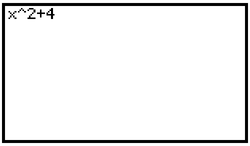
   Press [MODE] [F2] (use ⊖ to move to **Base** mode) ⊙ 3 [ENTER]

Steps and keystrokes	Display
9. Re-enter 0b10+0hF+10. Press ENTER	 ■ 0b10 + 0hF + 10 0h1B ■ 0b10 + 0hF + 10 0b11011 0b10+0hf+10 MAIN RAD AUTO FUNC 2/30

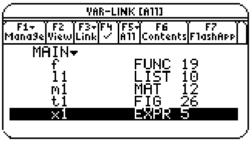
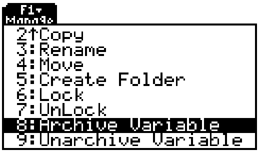
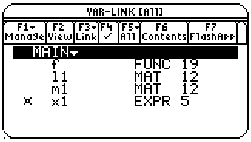
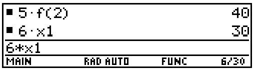
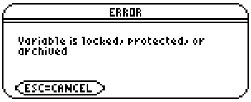
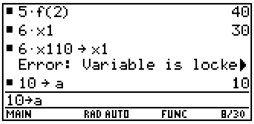
Memory and Variable Management

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.

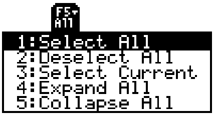
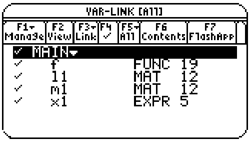
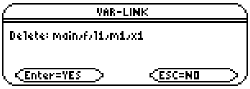
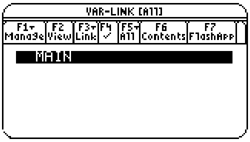
Steps and keystrokes	Display
1. From the Home screen, assign variables with the following variable types. Expression: 5 →x1 Function: x^2+4 →f(x) List: {5,10} →l1 Matrix: [30,25] →m1 🖩 HOME CLEAR 5 STO▶ X1 ENTER X ^ 2 + 4 STO▶ alpha F (X) ENTER 2nd [{] 5 , 10 2nd [}] STO▶ alpha L1 ENTER 2nd [[] 30 , 25 2nd []] STO▶ alpha M1 ENTER	 F1▾ Tools F2▾ Algebra F3▾ Calc F4▾ Other F5 PrgmIO F6▾ Clean Up ■ 5 → x1 5 ■ x^2 + 4 → f(x) Done ■ {5 10} → l1 {5 10} ■ [30 25] → m1 [30 25] [30,25]→m1 MAIN RAD AUTO FUNC 4/30
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]	 VAR-LINK [All] F1▾ Manage F2 View F3▾ Link F4 ✓ F5▾ All F6 Contents F7 FlashApp MAIN▾ f FUNC 19 l1 MAT 12 m1 MAT 12 x1 EXPR 5

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	 VAR-LINK VIEW View...... Variables→ Folder... All→ Var Type Function→ Enter=OK ESC=CANCEL  VAR-LINK [All] F1▾ Manage F2 View F3▾ Link F4 ✓ F5▾ All F6 Contents F7 FlashApp MAIN▾ f FUNC 19
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	 x^2+4
6. Close the Contents window. Press ESC	
7. With the **f** variable still highlighted, close **VAR-LINK** and paste the variable name to the entry line. Notice that “(” is pasted. Press ENTER	5∗f(
8. Complete the operation. Press 2) ENTER	5∗f(2)

Archiving a variable

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**)	 VAR-LINK [All] F1▾ Manage F2 View F3▾ Link F4 ✓ F5▾ All F6 Contents F7 FlashApp MAIN▾ f FUNC 19 l1 LIST 10 m1 MAT 12 t1 FIG 26 x1 EXPR 5
2. Use the [F1] **Manage** toolbar menu to archive the variable. ¤ indicates the variable is archived. Press [F1] 8	 F1▾ Manage 2↑Copy 3:Rename 4:Move 5:Create Folder 6:Lock 7:UnLock 8:Archive Variable 9:Unarchive Variable  VAR-LINK [All] F1▾ Manage F2 View F3▾ Link F4 ✓ F5▾ All F6 Contents F7 FlashApp MAIN▾ f FUNC 19 l1 MAT 12 m1 MAT 12 ¤ x1 EXPR 5
3. Return to the Home screen and use the archived variable in a calculation. [HOME] 6 [×] X1 [ENTER]	■ 5·f(2) 40 ■ 6·x1 30 6*x1 MAIN RAD AUTO FUNC 6/30
4. Attempt to store a different value to the archived variable. Press 10 [STO▸] X1 [ENTER]	ERROR Variable is locked, protected, or archived
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]	■ 5·f(2) 40 ■ 6·x1 30 ■ 6·x110 → x1 Error: Variable is locke▸ ■ 10 → a 10 10→a MAIN RAD AUTO FUNC 8/30

Deleting variables

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	 F5▾ All 1:Select All 2:Deselect All 3:Select Current 4:Expand All 5:Collapse All  VAR-LINK [All] F1▾ Manage F2 View F3▾ Link F4 ✓ F5▾ All F6 Contents F7 FlashApp ✓ MAIN▾ ✓ f FUNC 19 ✓ l1 MAT 12 ✓ m1 MAT 12 ✓ x1 EXPR 5
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	 Delete: main/f/l1/m1/x1
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	 VAR-LINK [All] F1▾ Manage F2 View F3▾ Link F4 ✓ F5▾ All F6 Contents F7 FlashApp MAIN
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	

3

Activities

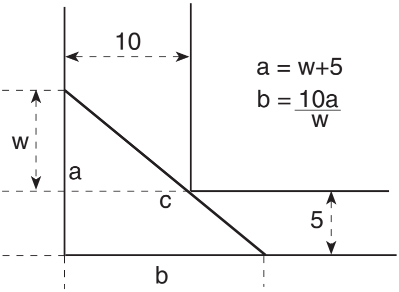
Analyzing the Pole-Corner Problem

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

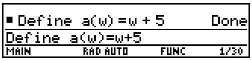
The maximum length of a pole **c** is the shortest line segment touching the interior corner and opposite sides of the two hallways as shown in the diagram below.

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.



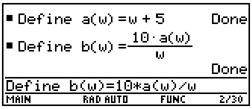
1. Define the expression for side **a** in terms of **w** and store it in **a(w)**.

   Note: When you want to define a function, use multiple character names as you build the definition.



```
■ Define a(w) = w + 5        Done
Define a(w)=w+5
```

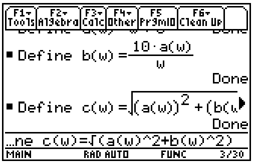
2. Define the expression for side **b** in terms of **w** and store it in **b(w).**



```
■ Define a(w) = w + 5        Done
■ Define b(w) = 10·a(w)/w
                             Done
Define b(w)=10*a(w)/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)**



■ Define $b(w) = \frac{10 \cdot a(w)}{w}$ Done

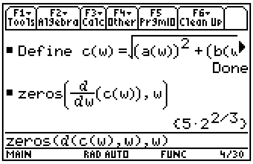
■ Define $c(w) = \sqrt{(a(w))^2 + (b(w}$ Done

...ne c(w)=√(a(w)^2+b(w)^2)

MAIN RAD AUTO FUNC 3/30

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)**.



■ Define $c(w) = \sqrt{(a(w))^2 + (b(w}$ Done

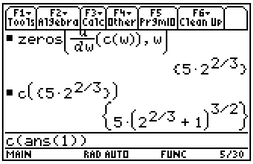
■ $\mathrm{zeros}\left(\frac{d}{dw}(c(w)), w\right)$ $\{5 \cdot 2^{2/3}\}$

zeros(d(c(w),w),w)

MAIN RAD AUTO FUNC 4/30

5. Compute the exact maximum length of the pole.

   Enter: **c** ([2nd] [ANS])



■ $\mathrm{zeros}\left(\frac{d}{dw}(c(w)), w\right)$ $\{5 \cdot 2^{2/3}\}$

■ $c(\{5 \cdot 2^{2/3}\})$ $\left\{5 \cdot \left(2^{2/3} + 1\right)^{3/2}\right\}$

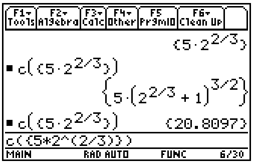
c(ans(1))

MAIN RAD AUTO FUNC 5/30

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].



$\{5 \cdot 2^{2/3}\}$

■ $c(\{5 \cdot 2^{2/3}\})$ $\left\{5 \cdot \left(2^{2/3} + 1\right)^{3/2}\right\}$

■ $c(\{5 \cdot 2^{2/3}\})$ {20.8097}

c({5*2^(2/3)})

MAIN RAD AUTO FUNC 6/30

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

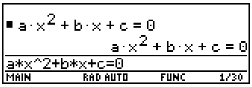
Perform the following steps to derive the quadratic formula by completing the square of the generalized quadratic equation.

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**.

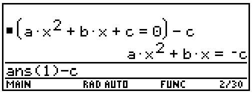


▪ $a\cdot x^2+b\cdot x+c=0$ $\quad a\cdot x^2+b\cdot x+c=0$

```
a*x^2+b*x+c=0
MAIN          RAD AUTO      FUNC      1/30
```

3. Subtract c from both sides of the equation.

   [2nd] [ANS] [−] [alpha] **C**

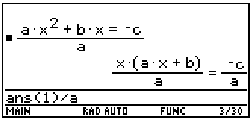


▪ $(a\cdot x^2+b\cdot x+c=0)-c$ $\quad a\cdot x^2+b\cdot x={}^-c$

```
ans(1)-c
MAIN          RAD AUTO      FUNC      2/30
```

   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**.

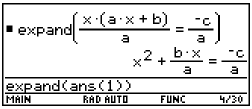


▪ $\frac{a\cdot x^2+b\cdot x={}^-c}{a}$ $\quad \frac{x\cdot(a\cdot x+b)}{a}=\frac{{}^-c}{a}$

```
ans(1)/a
MAIN          RAD AUTO      FUNC      3/30
```

   Note: Continue to use the last answer ([2nd] [ANS]) as in step 3 in steps 4 through 9.

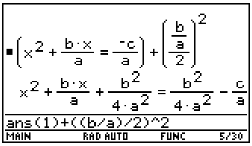
5. Use the **expand( )** function to expand the result of the last answer.



▪ $\mathrm{expand}\left(\frac{x\cdot(a\cdot x+b)}{a}=\frac{{}^-c}{a}\right)$ $\quad x^2+\frac{b\cdot x}{a}=\frac{{}^-c}{a}$

```
expand(ans(1))
MAIN          RAD AUTO      FUNC      4/30
```

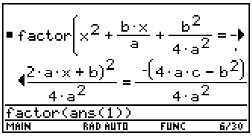
6. Complete the square by adding **((b/a)/2)^2** to both sides of the equation.



▪ $\left(x^2+\frac{b\cdot x}{a}=\frac{{}^-c}{a}\right)+\left(\frac{\frac{b}{a}}{2}\right)^2$ $\quad x^2+\frac{b\cdot x}{a}+\frac{b^2}{4\cdot a^2}=\frac{b^2}{4\cdot a^2}-\frac{c}{a}$

```
ans(1)+((b/a)/2)^2
MAIN          RAD AUTO      FUNC      5/30
```

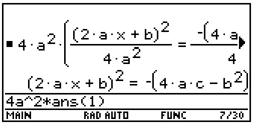
7. Factor the result using the **factor( )** function.



▪ $\mathrm{factor}\left(x^2+\frac{b\cdot x}{a}+\frac{b^2}{4\cdot a^2}=\right.$ ▸ $\quad$ ◂ $\frac{(2\cdot a\cdot x+b)^2}{4\cdot a^2}=\frac{-(4\cdot a\cdot c-b^2)}{4\cdot a^2}$

```
factor(ans(1))
MAIN          RAD AUTO      FUNC      6/30
```

8. Multiply both sides of the equation by **4a²**.

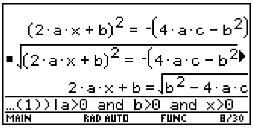


$\blacksquare\ 4\cdot a^2\cdot\left(\frac{(2\cdot a\cdot x+b)^2}{4\cdot a^2}=\frac{-(4\cdot a}{4}\right.$ ▸

$(2\cdot a\cdot x+b)^2=-(4\cdot a\cdot c-b^2)$

```
4a^2*ans(1)
MAIN          RAD AUTO      FUNC        7/30
```

9. Take the square root of both sides of the equation with the constraint that **a>0** and **b>0** and **x>0**.



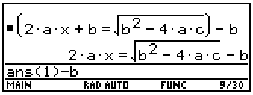
$(2\cdot a\cdot x+b)^2=-(4\cdot a\cdot c-b^2)$

$\blacksquare\ \sqrt{(2\cdot a\cdot x+b)^2=-(4\cdot a\cdot c-b^2}$ ▸

$2\cdot a\cdot x+b=\sqrt{b^2-4\cdot a\cdot c}$

```
…(1))|a>0 and b>0 and x>0
MAIN          RAD AUTO      FUNC        8/30
```

10. Solve for **x** by subtracting **b** from both sides and then dividing by **2a**.

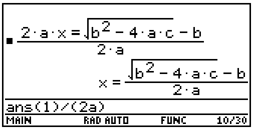


$\blacksquare\ (2\cdot a\cdot x+b=\sqrt{b^2-4\cdot a\cdot c})-b$

$2\cdot a\cdot x=\sqrt{b^2-4\cdot a\cdot c}-b$

```
ans(1)-b
MAIN          RAD AUTO      FUNC        9/30
```

Note: This is only one of the two general quadratic solutions due to the constraint in step 9.



$\blacksquare\ \frac{2\cdot a\cdot x=\sqrt{b^2-4\cdot a\cdot c}-b}{2\cdot a}$

$x=\frac{\sqrt{b^2-4\cdot a\cdot c}-b}{2\cdot a}$

```
ans(1)/(2a)
MAIN          RAD AUTO      FUNC       10/30
```

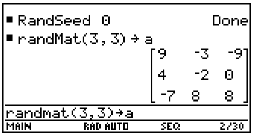
Exploring a Matrix

This activity shows you how to perform several matrix operations.

Exploring a 3x3 Matrix

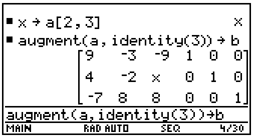
Perform these steps to generate a random matrix, augment and find the identity matrix, and then solve to find an invalid value of the inverse.

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**.



```
■ RandSeed 0                    Done
■ randMat(3,3) → a
                          [ 9  -3  -9 ]
                          [ 4  -2   0 ]
                          [-7   8   8 ]
randmat(3,3)→a
MAIN          RAD AUTO      SEQ         2/30
```

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**.

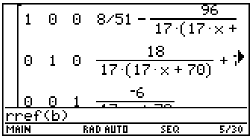


```
■ x → a[2,3]                           x
■ augment(a,identity(3)) → b
                 [ 9  -3  -9   1  0  0 ]
                 [ 4  -2   x   0  1  0 ]
                 [-7   8   8   0  0  1 ]
augment(a,identity(3))→b
MAIN          RAD AUTO      SEQ         4/30
```

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.



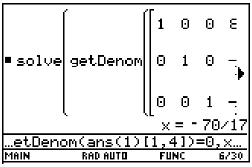
```
1  0  0  8/51 - 96/(17·(17·x +
0  1  0  18/(17·(17·x + 70)) +
0  0  1  -6/
rref(b)
MAIN        RAD AUTO      SEQ      5/30
```

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.



```
■ solve  getDenom  1  0  0  ε
                   0  1  0  -
                   0  0  1  -
                          x = -70/17
…etDenom(ans(1)[1,4])=0,x…
MAIN        RAD AUTO      FUNC     6/30
```

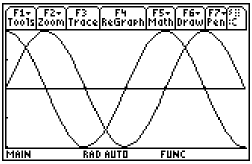
Exploring cos(x) = sin(x)

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

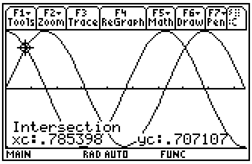
Perform the following steps to observe where the graphs of the functions **y1(x)=cos(x)** and **y2(x)=sin(x)** intersect.

1. In the **Y= Editor**, set **y1(x)=cos(x)** and **2(x)=sin(x)**.
2. In the **Window Editor**, set **xmin=0** and **xmax=3**π.
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**.



```
Intersection
xc:.785398      yc:.707107
```

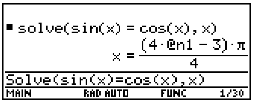
5. Note the **x** and **y** coordinates. (Repeat steps 4 and 5 to find the other intersections.)

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)**.

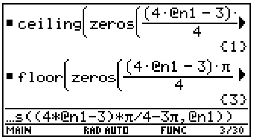
   The solution for **x** is where **@n1** is any integer.



```
■ solve(sin(x) = cos(x), x)
                 x = (4·@n1 − 3)·π / 4
Solve(sin(x)=cos(x),x)
MAIN        RAD AUTO      FUNC      1/30
```

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.

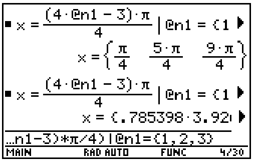


```
■ ceiling(zeros((4·@n1 − 3)· / 4 ▶
                                  {1}
■ floor(zeros((4·@n1 − 3)·π / 4 ▶
                                  {3}
…s((4*@n1-3)*π/4-3π,@n1))
MAIN        RAD AUTO      FUNC      3/30
```

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: 🖩 I



```
■ x = (4·@n1 − 3)·π / 4 | @n1 = {1 ▶
              x = {π/4  5·π/4  9·π/4}
■ x = (4·@n1 − 3)·π / 4 | @n1 = {1 ▶
              x = {.785398 3.92 ▶
…n1-3)*π/4)|@n1={1,2,3}
MAIN        RAD AUTO      FUNC      4/30
```

Finding Minimum Surface Area of a Parallelepiped

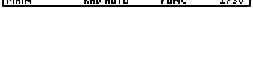
This activity shows you how to find the minimum surface area of a parallelepiped having a constant volume **V**. Detailed information about the steps used in this example can be found in *Symbolic Manipulation* and *3D Graphing*.

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.

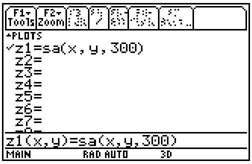
1. On the Home screen, define the function **sa(x,y,v)** for the surface area of a parallelepiped.

   Enter: **define sa(x,y,v)=2∗x∗y + 2v/x+2v/y**



```
■ Define sa(x, y, v) = 2·x·y +▶
                            Done
…ine sa(x,y,v)=2*x*y+2v/y
MAIN        RAD AUTO      FUNC      1/30
```

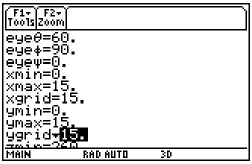
2. Select the 3D Graph mode. Then enter the function for **z1(x,y)** as shown in this example with volume **v=300**.



```
▸PLOTS
✓z1=sa(x,y,300)
 z2=
 z3=
 z4=
 z5=
 z6=
 z7=
z1(x,y)=sa(x,y,300)
MAIN        RAD AUTO      3D
```

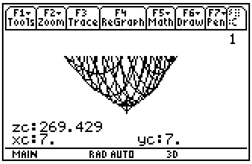
3. Set the Window variables to:

   eye=   [60,90,0]
   x=   [0,15,15]
   y=   [0,15,15]
   z=   [260,300]
   ncontour=   [5]



```
eyeθ=60.
eyeϕ=90.
eyeψ=0.
xmin=0.
xmax=15.
xgrid=15.
ymin=0.
ymax=15.
ygrid=15.
MAIN        RAD AUTO      3D
```

4. Graph the function and use **Trace** to go to the point close to the minimum value of the surface area function.



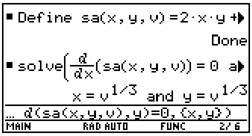
```
zc:269.429
xc:7.          yc:7.
MAIN        RAD AUTO      3D
```

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})**

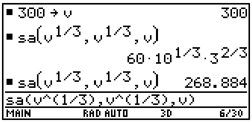


```
■ Define sa(x,y,v)=2·x·y+▸
                          Done
■ solve(d/dx(sa(x,y,v)) = 0 a▸
         x = v^(1/3) and y = v^(1/3)
… d(sa(x,y,v),y)=0,{x,y})
MAIN        RAD AUTO      FUNC     2/6
```

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.



```
■ 300 → v                      300
■ sa(v^(1/3), v^(1/3), v)
                 60·10^(1/3)·3^(2/3)
■ sa(v^(1/3), v^(1/3), v)   268.884
sa(v^(1/3),v^(1/3),v)
MAIN        RAD AUTO      3D      6/30
```

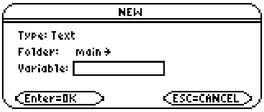
Running a Tutorial Script Using the Text Editor

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**.

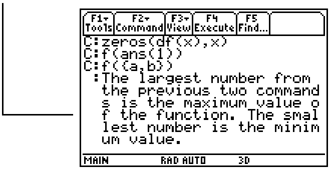


```
NEW
Type: Text
Folder:   main →
Variable: 
```

Note: The command symbol **C** is accessed from the F2 **1:Command** toolbar menu.

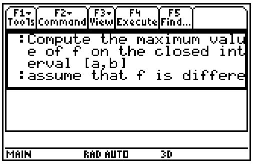
2. Type the following lines into the **Text Editor**.

```
   : 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.
```



```
C:zeros(df(x),x)
C:f(ans(1))
C:f({a,b})
 :The largest number from
  the previous two command
  s is the maximum value o
  f the function. The smal
  lest number is the minim
  um value.
```

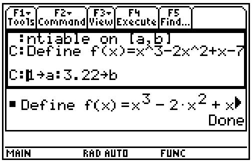
3. Press F3 and select **1:Script view** to show the **Text Editor** and the Home screen on a split-screen. Move the cursor to the first line in the **Text Editor**.



```
:Compute the maximum valu
 e of f on the closed int
 erval [a,b]
:assume that f is differe
```

4. Press [F4] repeatedly to execute each line in the script one at a time.

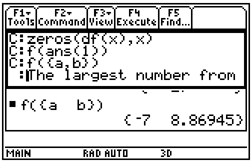
   Note: Press [F4] and select **2:Clear split** to go back to a full-sized **Text Editor** screen.



```
 :ntiable on [a,b]
C:Define f(x)=x^3-2x^2+x-7
C:1→a:3.22→b
```

■ Define f(x) = $x^3 - 2\cdot x^2 + x$▸ Done

MAIN RAD AUTO FUNC



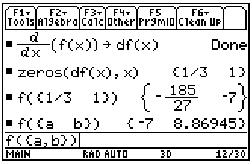
```
C:zeros(df(x),x)
C:f(ans(1))
C:f({a,b})
 :The largest number from
```

■ f({a b}) {-7 8.86945}

MAIN RAD AUTO 3D

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.



■ $\frac{d}{dx}$(f(x)) → df(x) Done

■ zeros(df(x), x) {1/3 1}

■ f({1/3 1}) $\left\{-\frac{185}{27} \quad -7\right\}$

■ f({a b}) {-7 8.86945}

f({a,b})

MAIN RAD AUTO 3D 12/30

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*.

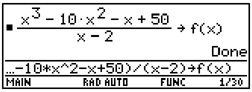
Decomposing a Rational Function

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.

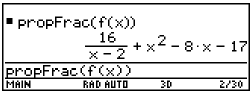


■ $\frac{x^3 - 10\cdot x^2 - x + 50}{x - 2}$ → f(x) Done

…-10*x^2-x+50)/(x-2)→f(x)

MAIN RAD AUTO FUNC 1/30

2. Use the proper fraction function (**propFrac**) to split the function into a quotient and remainder.



■ propFrac(f(x))

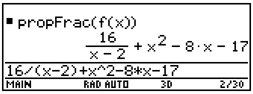
$\frac{16}{x - 2} + x^2 - 8\cdot x - 17$

propFrac(f(x))

MAIN RAD AUTO 3D 2/30

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.



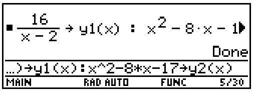
■ propFrac(f(x))

$\frac{16}{x-2} + x^2 - 8 \cdot x - 17$

16/(x-2)+x^2-8*x-17

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)**

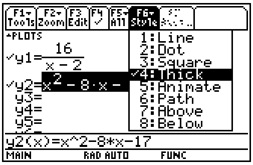


■ $\frac{16}{x-2}$ → y1(x) : $x^2 - 8 \cdot x - 1$▸

Done

...)→y1(x):x^2-8*x-17→y2(x)

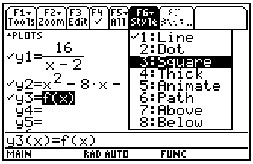
5. In the **Y= Editor**, select the thick graphing style for **y2(x)**.



▸PLOTS
✓y1=$\frac{16}{x-2}$
✓y2=$x^2 - 8 \cdot x -$
y3=
y4=
y5=

y2(x)=x^2-8*x-17

6. Add the original function **f(x)** to **y3(x)** and select the square graphing style.

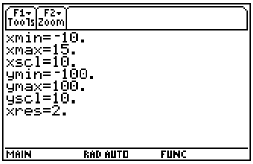


▸PLOTS
✓y1=$\frac{16}{x-2}$
✓y2=$x^2 - 8 \cdot x -$
✓y3=f(x)
y4=
y5=

y3(x)=f(x)

7. In the **Window Editor**, set the window variables to:

   x= **[$^{-}$10,15,10]**
   y= **[$^{-}$100,100,10]**



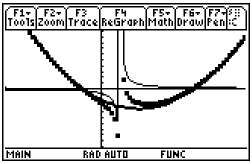
xmin=$^{-}$10.
xmax=15.
xscl=10.
ymin=$^{-}$100.
ymax=100.
yscl=10.
xres=2.

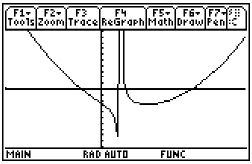
8. Draw the graph.

   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.

Filtering Data by Categories

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

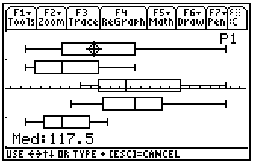
Category (C2)	Academic Year and Sex
1	Freshman boys
2	Freshman girls
3	Sophomore boys
4	Sophomore girls
5	Junior boys
6	Junior girls
7	Senior boys
8	Senior girls

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.

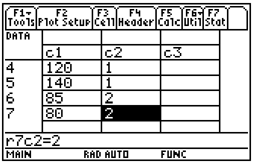
1. Start the **Data/Matrix Editor**, and create a new Data variable named **students**.



Med:117.5

USE ←→↑↓ OR TYPE + [ESC]=CANCEL

2. Enter the data and categories from Table 2 into columns **c1** and **c2**, respectively.



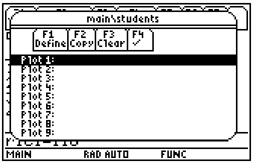
DATA			
	c1	c2	c3
4	120	1	
5	140	1	
6	85	2	
7	80	2	

r7c2=2

MAIN RAD AUTO FUNC

3. Open the F2 **Plot Setup** toolbar menu.

Note: Set up several box plots to compare different subsets of the entire data set.

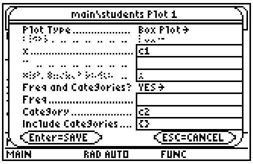


main\students

Plot 1:
Plot 2:
Plot 3:
Plot 4:
Plot 5:
Plot 6:
Plot 7:
Plot 8:
Plot 9:

MAIN RAD AUTO FUNC

4. Define the plot and filter parameters for **Plot 1** as shown in this screen.

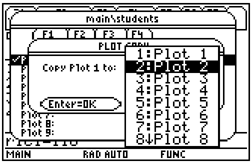


main\students Plot 1

Plot Type.................. Box Plot →
x.................................. c1
Freq and Categories? YES →
Freq............................
Category.................... c2
Include Categories.... {}

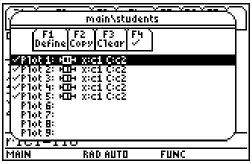
MAIN RAD AUTO FUNC

5. Copy **Plot 1** to **Plot 2**.



```
main\students
PLOT COPY
Copy Plot 1 to:
<Enter=OK>
1:Plot 1
2:Plot 2
3:Plot 3
4:Plot 4
5:Plot 5
6:Plot 6
7:Plot 7
8↓Plot 8
Plot 8:
Plot 9:
MAIN          RAD AUTO       FUNC
```

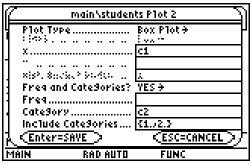
6. Repeat step 5 and copy **Plot 1** to **Plot 3**, **Plot 4**, and **Plot 5**.



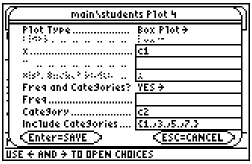
```
main\students
F1     F2   F3    F4
Define Copy Clear ✓
✓Plot 1: ⊢◫⊣ x:c1 C:c2
✓Plot 2: ⊢◫⊣ x:c1 C:c2
✓Plot 3: ⊢◫⊣ x:c1 C:c2
✓Plot 4: ⊢◫⊣ x:c1 C:c2
✓Plot 5: ⊢◫⊣ x:c1 C:c2
 Plot 6:
 Plot 7:
 Plot 8:
 Plot 9:
MAIN          RAD AUTO       FUNC
```

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)



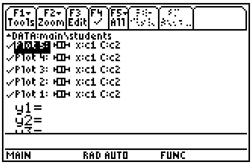
```
main\students Plot 2
Plot Type.................. Box Plot →
x.......................... c1
Freq and Categories?        YES →
Freq.......................
Category................... c2
Include Categories......... {1,2}
<Enter=SAVE>                <ESC=CANCEL>
MAIN          RAD AUTO       FUNC
```



```
main\students Plot 4
Plot Type.................. Box Plot →
x.......................... c1
Freq and Categories?        YES →
Freq.......................
Category................... c2
Include Categories......... {1,3,5,7}
<Enter=SAVE>                <ESC=CANCEL>
USE ← AND → TO OPEN CHOICES
```

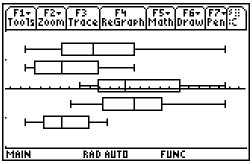
8. In the **Y= Editor**, deselect any functions that may be selected from a previous activity.

   Note: Only **Plot 1** through **Plot 5** should be selected.



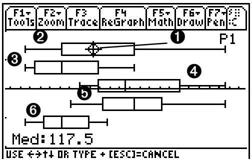
```
F1▾   F2▾  F3   F4 F5▾
Tools Zoom Edit ✓  All
▸DATA:main\students
✓Plot 5: ⊢◫⊣ x:c1 C:c2
✓Plot 4: ⊢◫⊣ x:c1 C:c2
✓Plot 3: ⊢◫⊣ x:c1 C:c2
✓Plot 2: ⊢◫⊣ x:c1 C:c2
✓Plot 1: ⊢◫⊣ x:c1 C:c2
 y1=
 y2=
MAIN          RAD AUTO       FUNC
```

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



Med:117.5

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, and is similar to the “Coffee To Go” experiment in the *CBL System Experiment Workbook*. 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.
: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.

Program Instruction	Description
: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.
: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.

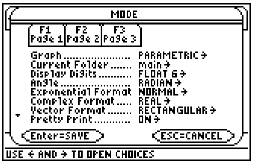
Studying the Flight of a Hit Baseball

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.

Setting Up a Parametric Graph and Table

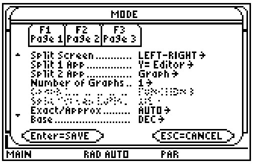
Perform the following steps to study the flight of a hit baseball that has an initial velocity of 95 feet per second and an initial angle of 32 degrees.

1. Set the modes for **Page 1** as shown in this screen.



```
MODE
F1 Page 1  F2 Page 2  F3 Page 3
Graph.................. PARAMETRIC →
Current Folder......... main →
Display Digits......... FLOAT 6 →
Angle.................. RADIAN →
Exponential Format..... NORMAL →
Complex Format......... REAL →
Vector Format.......... RECTANGULAR →
Pretty Print........... ON →
Enter=SAVE          ESC=CANCEL
USE ← AND → TO OPEN CHOICES
```

2. Set the modes for **Page 2** as shown in this screen.

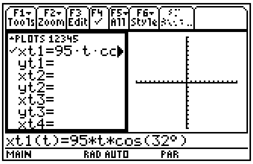


```
MODE
F1 Page 1  F2 Page 2  F3 Page 3
Split Screen........... LEFT-RIGHT →
Split 1 App............ Y= Editor →
Split 2 App............ Graph →
Number of Graphs....... 1 →
Exact/Approx........... AUTO →
Base................... DEC →
Enter=SAVE          ESC=CANCEL
MAIN          RAD AUTO        PAR
```

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°)

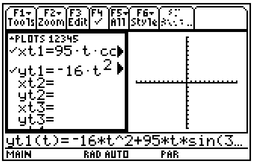
   Note: Press [2nd] [°] to obtain the degree symbol.



```
F1 Tools  F2 Zoom  F3 Edit  F4 ✓  F5 All  F6 Style
PLOTS 12345
✓xt1=95·t·cc
 yt1=
 xt2=
 yt2=
 xt3=
 yt3=
 xt4=
xt1(t)=95*t*cos(32°)
MAIN          RAD AUTO        PAR
```

4. In the **Y= Editor**, enter the equation for the height of the ball at time t for **yt1(t)**.

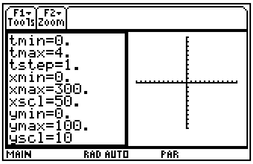
   yt1(t)=⁻16∗t^2+95∗t∗sin(32°)



```
F1 Tools  F2 Zoom  F3 Edit  F4 ✓  F5 All  F6 Style
PLOTS 12345
✓xt1=95·t·cc
✓yt1=-16·t²
 xt2=
 yt2=
 xt3=
 yt3=
yt1(t)=-16*t^2+95*t*sin(3…
MAIN          RAD AUTO        PAR
```

5. Set the Window variables to:

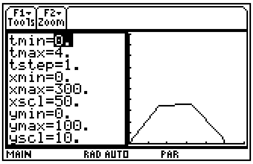
   t values= [0,4,.1]
   x values= [0,300,50]
   y values= [0,100,10]



```
F1 Tools  F2 Zoom
tmin=0.
tmax=4.
tstep=1.
xmin=0.
xmax=300.
xscl=50.
ymin=0.
ymax=100.
yscl=10
MAIN          RAD AUTO        PAR
```

6. Switch to the right side and display the graph.

   Note: Press [2nd] [⊞].



```
F1 Tools  F2 Zoom
tmin=0.
tmax=4.
tstep=1.
xmin=0.
xmax=300.
xscl=50.
ymin=0.
ymax=100.
yscl=10.
MAIN          RAD AUTO        PAR
```

7. Display the **TABLE SETUP** dialog box, and change **tblStart** to **0** and **Δtbl** to **0.1**.

   Note: Press [◆] [TBLSET].

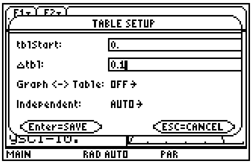


TABLE SETUP

tblStart: 0.

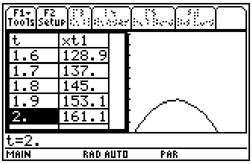
Δtbl: 0.1

Graph <-> Table: OFF →

Independent: AUTO →

8. Display the table in the left side and press ⊙ to highlight **t=2**.

   Note: Press [◆] [TABLE].

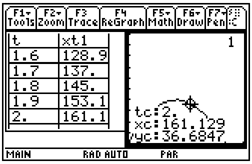


t	xt1
1.6	128.9
1.7	137.
1.8	145.
1.9	153.1
2.	161.1

t=2.

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**.



t	xt1
1.6	128.9
1.7	137.
1.8	145.
1.9	153.1
2.	161.1

tc:2.
xc:161.129
yc:36.6847

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.

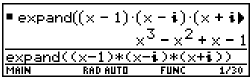
Visualizing Complex Zeros of a Cubic Polynomial

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.

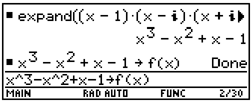
1. On the Home screen, use the **expand( )** function to expand the cubic expression **(x−1)(x−*i*)(x+*i*)** and see the first polynomial.



```
▪ expand((x − 1)·(x − i)·(x + i▶
                      x^3 − x^2 + x − 1
expand((x-1)*(x-i)*(x+i))
```

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.

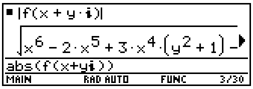


```
▪ expand((x − 1)·(x − i)·(x + i▶
                      x^3 − x^2 + x − 1
▪ x^3 − x^2 + x − 1 → f(x)        Done
x^3-x^2+x-1→f(x)
```

3. Use the **abs( )** function to find the absolute value of **f(x+y*i*)**.

   (This calculation may take about 2 minutes.)

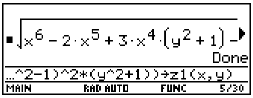
   Note: The absolute value of a function forces any roots to visually just touch rather than cross the **x** axis. Likewise, the absolute value of a function of two variables will force any roots to visually just touch the **xy** plane.



```
■ |f(x + y·i)|
  √(x^6 - 2·x^5 + 3·x^4·(y^2 + 1) -▸
abs(f(x+yi))
MAIN        RAD AUTO     FUNC     3/30
```

4. Copy and paste the last answer to the entry line and store it in the function **z1(x,y)**.

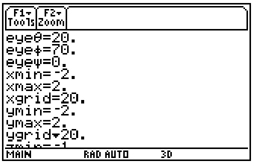
   Note: The graph of **z1(x,y)** will be the modulus surface.



```
■ √(x^6 - 2·x^5 + 3·x^4·(y^2 + 1) -▸
                                  Done
…^2-1)^2*(y^2+1))→z1(x,y)
MAIN        RAD AUTO     FUNC     5/30
```

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]

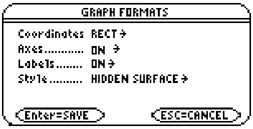


```
eyeθ=20.
eyeϕ=70.
eyeψ=0.
xmin=-2.
xmax=2.
xgrid=20.
ymin=-2.
ymax=2.
ygrid▾20.
MAIN        RAD AUTO     3D
```

6. In the **Y=Editor**, press:
   ◆ |
   and set the Graph Format variables to:

   Axes=   ON
   Labels= ON
   Style=   HIDDEN SURFACE

   Note: Calculating and drawing the graph takes about three minutes.

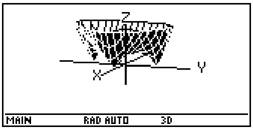


GRAPH FORMATS

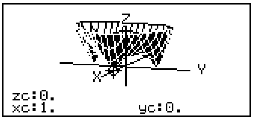
```
Coordinates  RECT→
Axes........ ON →
Labels...... ON→
Style....... HIDDEN SURFACE→
```

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.

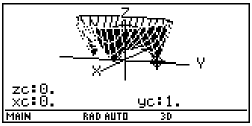


8. Use the Trace tool to explore the function values at **x=1** and **y=0**.



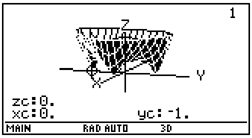
```
zc:0.
xc:1.            yc:0.
```

9. Use the Trace tool to explore the function values at **x=0** and **y=1**.



zc:0.
xc:0. yc:1.

10. Use the Trace tool to explore the function values at **x=0** and **y=⁻1**.



zc:0.
xc:0. yc: ⁻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 $\mathbf{x^3-x^2+x-1}$ can be visualized with the three points where the graph of the modulus surface touches the **xy** plane.

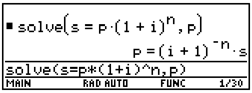
Solving a Standard Annuity Problem

This activity can be used to find the interest rate, starting principal, number of compounding periods, and future value of an annuity.

Finding the Interest Rate 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**.

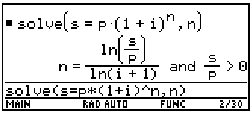


■ $\text{solve}\left(s = p\cdot(1+i)^n, p\right)$

$p = (i+1)^{-n}\cdot s$

```
solve(s=p*(1+i)^n,p)
```

2. Enter the equation to solve for **n**.



■ $\text{solve}\left(s = p\cdot(1+i)^n, n\right)$

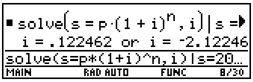
$n = \frac{\ln\left(\frac{s}{p}\right)}{\ln(i+1)} \text{ and } \frac{s}{p} > 0$

```
solve(s=p*(1+i)^n,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%.



```
▪ solve(s = p·(1 + i)ⁿ, i)| s =▸
  i = .122462 or i = ⁻2.12246
solve(s=p*(1+i)^n,i)|s=20…
MAIN        RAD AUTO      FUNC       8/30
```

   Note:

   - To enter the "*with*" (|) operator:
     ▦ [I]

   - Press [◆] [ENTER] to obtain a floating-point result.

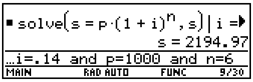
Finding the Future Value of an Annuity

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.



```
▪ solve(s = p·(1 + i)ⁿ, s)| i =▸
                      s = 2194.97
…i=.14 and p=1000 and n=6
MAIN        RAD AUTO      FUNC       9/30
```

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 *Programming*.

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).

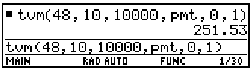
```
:tvm(temp1,temp2,temp3,temp4,temp5,temp6)
:Func
:Local tempi,tempfunc,tempstr1
:⁻temp3+(1+temp2/1200 temp6) temp4 ((1−(1+temp2/1200)^
    (⁻temp1))/(temp2/1200))−temp5 (1+temp2/1200)^(⁻temp1)
    →tempfunc
:For tempi,1,5,1
:"temp"&exact(string(tempi))→tempstr1
:If when(#tempstr1=0,false,false,true) 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**.



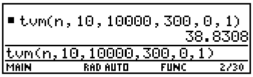
```
■ tvm(48,10,10000,pmt,0,1)
                      251.53
tvm(48,10,10000,pmt,0,1)
MAIN      RAD AUTO      FUNC      1/30
```

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**.



```
▪ tvm(n, 10, 10000, 300, 0, 1)
                       38.8308
tvm(n,10,10000,300,0,1)
```

Result: The number of payments is 38.8308.

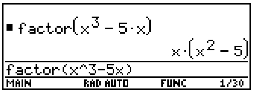
Finding Rational, Real, and Complex Factors

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*.

Finding Factors

Enter the expressions shown below on the Home screen.

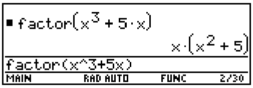
1. **factor(x^3−5x)** [ENTER] displays a rational result.



▪ $\text{factor}(x^3 - 5\cdot x)$ $\quad x\cdot(x^2 - 5)$

```
factor(x^3-5x)
```

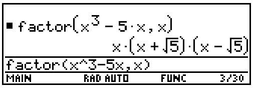
2. **factor(x^3+5x)** [ENTER] displays a rational result.



▪ $\text{factor}(x^3 + 5\cdot x)$ $\quad x\cdot(x^2 + 5)$

```
factor(x^3+5x)
```

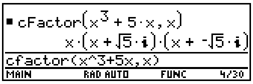
3. **factor(x^3−5x,x)** [ENTER] displays a real result.



▪ $\text{factor}(x^3 - 5\cdot x, x)$ $\quad x\cdot(x + \sqrt{5})\cdot(x - \sqrt{5})$

```
factor(x^3-5x,x)
```

4. **cfactor(x^3+5x,x)** [ENTER] displays a complex result.



▪ $\text{cFactor}(x^3 + 5\cdot x, x)$ $\quad x\cdot(x + \sqrt{5}\cdot i)\cdot(x + -\sqrt{5}\cdot i)$

```
cfactor(x^3+5x,x)
```

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 *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
```

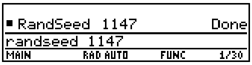
(*continued in next column*)

```
: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
```

Sampling without Replacement

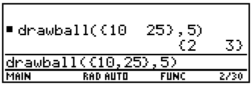
Suppose an urn contains **n1** balls of a color, **n2** balls of a second color, **n3** balls of a third color, etc. Simulate drawing balls without replacing them.

1. Enter a random seed using the **RandSeed** command.



```
■ RandSeed 1147                Done
randseed 1147
MAIN        RAD AUTO    FUNC    1/30
```

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)**.



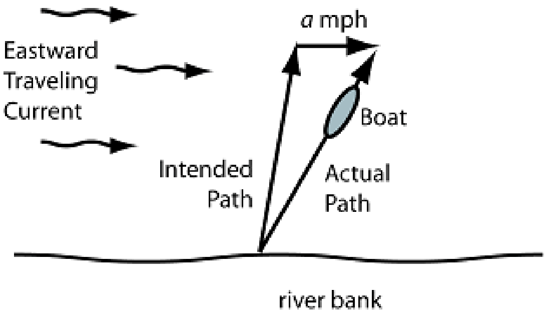
```
■ drawball({10  25},5)
                         {2    3}
drawball({10,25},5)
MAIN        RAD AUTO    FUNC    2/30
```

Result: 2 red balls and 3 white balls.

Using Vectors to Determine Velocity

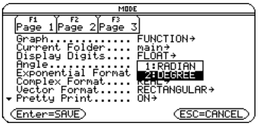
A small fishing boat leaves from the south bank of the Allegheny River and heads at a 80° angle with an engine speed of 20 knots. However, the eastward force of the current carries the boat along so it actually travels at a 60° angle with the shore.

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 instead of radians 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**.



```
Graph.............. FUNCTION→
Current Folder.... main→
Display Digits.... FLOAT→
Angle.............. 1:RADIAN
Exponential Format  2:DEGREE
Complex Format.... REAL→
Vector Format..... RECTANGULAR→
▾ Pretty Print...... ON→
```

2. Enter vectors describing the initial path of the boat, water current, and resultant path of the boat.

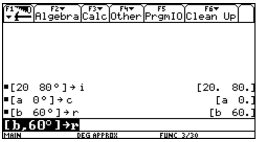
   Store these vectors as **i**, **c**, and **r**. Use the value **a** for the unknown speed of the current. Use the value **b** for the speed of the boat.

   Enter:

   [20,80°]→i

   [a,0°]→c

   [b,60°]→r

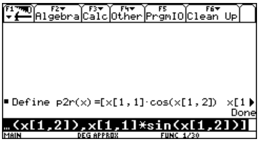


```
■[20  80°]→i                [20.  80.]
■[a  0°]→c                      [a  0.]
■[b  60°]→r                    [b  60.]
[b,60°]→r
MAIN        DEG APPROX        FUNC 3/30
```

Vectors are commonly written in either polar or rectangular form, so it is useful to convert polar vectors into rectangular form.

3. Define function **p2r**.

   Enter: **Define p2r(x)=[x[1,1]*cos(x[1,2]), x[1,1]*sin(x[1,2])]**



```
■ Define p2r(x) =[x[1,1]·cos(x[1,2])  x[1 ▸
                                     Done
…(x[1,2]),x[1,1]*sin(x[1,2])]
MAIN        DEG APPROX        FUNC 1/30
```

When converted to rectangular form, the sum of vectors **i** and **c** equals the resultant vector **r**.

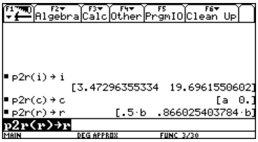
4. Using function **p2r**, convert vectors **i**, **c**, and **r** to rectangular form.

   Enter:

   p2r(i)→i

   p2r(c)→c

   p2r(r)→r



```
■ p2r(i) → i
              [3.47296355334  19.6961550602]
■ p2r(c) → c                          [a  0.]
■ p2r(r) → r        [.5·b  .866025403784·b]
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**.

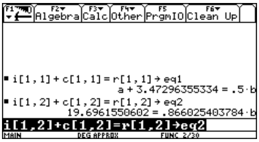
5. Set up two equations involving vectors **i**+**c** and **r**.

   - 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



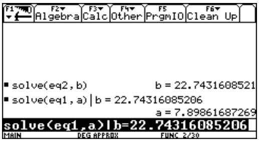
```
■ i[1, 1] + c[1, 1] = r[1, 1] → eq1
                    a + 3.47296355334 = .5·b
■ i[1, 2] + c[1, 2] = r[1, 2] → eq2
          19.6961550602 = .866025403784·b
i[1,2]+c[1,2]=r[1,2]→eq2
```

6. Solve **eq2** for **b** to calculate the actual speed of the boat.

   solve(eq2,b)

7. 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



```
■ solve(eq2, b)            b = 22.7431608521
■ solve(eq1, a)|b = 22.74316085206
                           a = 7.89861687269
solve(eq1,a)|b=22.74316085206
```

The boat travels at a speed of 22.7 knots, and the water current is approximately 7.9 knots.

Connectivity

Connecting Two Units

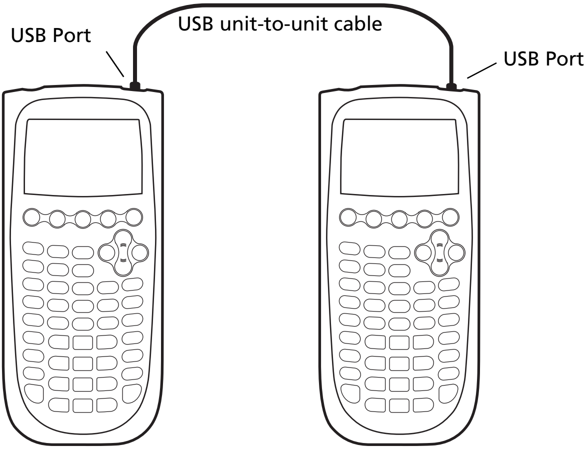
The TI-89 Titanium comes with a cable that lets you connect two units. Once connected, you can transmit information between two units. A USB unit-to-unit cable is included with the TI-89 Titanium; use the calculator's USB port with this cable.

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.

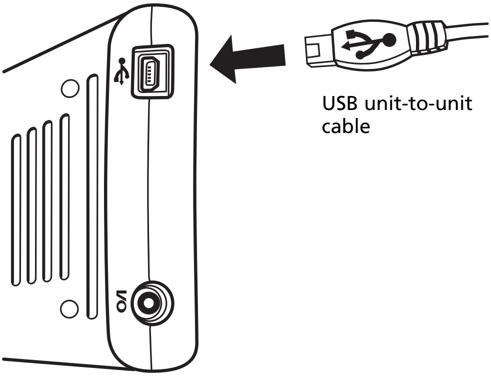
Connecting before Sending or Receiving

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.

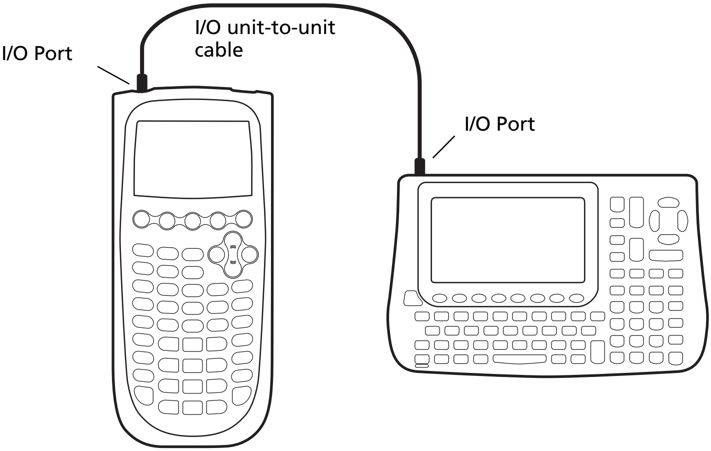
You can link a TI-89 Titanium or Voyage™ 200 to another TI-89 Titanium, Voyage™ 200, TI-89, or TI-92 Plus.



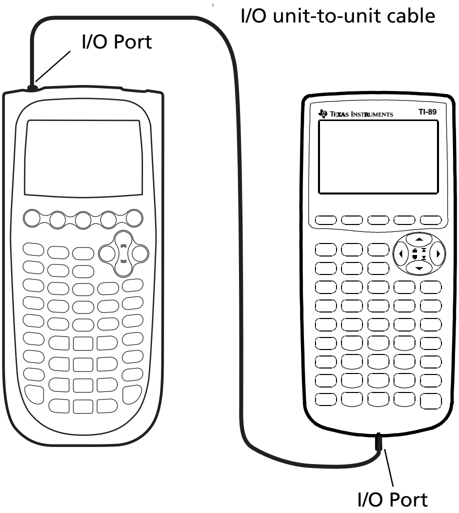
Two TI-89 Titanium calculators linked together



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.

Setting Up the Units

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.

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.

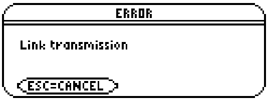
Canceling a Transmission

From either the sending or receiving unit:

1. Press ON.

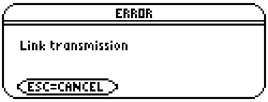
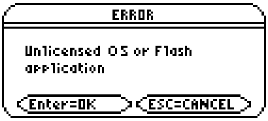
   An error message is displayed.

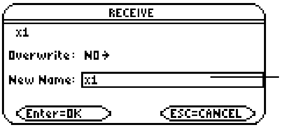
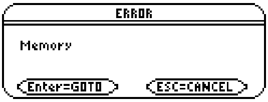
2. Press ESC or ENTER.


ERROR

Link transmission

Common Error and Notification Messages

Shown on:	Message and Description:
Sending unit	 ERROR Link transmission 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.
Sending unit	 ERROR Unlicensed OS or Flash application The receiving unit does not have the correct certification for the operating system (OS) or Flash application being sent.

Shown on:	Message and Description:
Receiving unit	 x1 Overwrite: NO → New Name: x1 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.
Receiving unit	 Memory 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 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
❶  :      GetCalc msg
   :      Disp msg               ❸
❷  :      InputStr msg
   :      SendCalc msg
   :   EndWhile
   :Else
   :   While true
❹  :     InputStr msg
   :     SendCalc msg            ❻
❺  :     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.

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 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

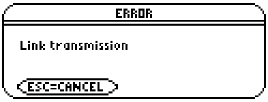
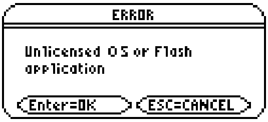
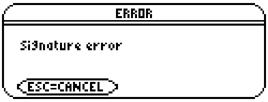
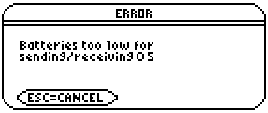
After the transfer starts, the receiving unit’s existing OS is effectively deleted. If you interrupt the transfer before it is complete, the receiving unit will not operate properly. You will then need to reinstall the OS upgrade.

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
 ERROR Link transmission ESC=CANCEL	The sending and receiving units are not connected properly, or the receiving unit is not set up to receive.
 ERROR Unlicensed OS or Flash application Enter=OK ESC=CANCEL	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.
 ERROR Signature error ESC=CANCEL	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 Batteries too low for sending/receiving OS ESC=CANCEL	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

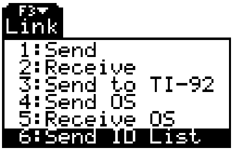
The ID list feature provides a convenient way to collect device IDs for group purchase of commercial applications. After the IDs are collected, transmit them to Texas Instruments so a group certificate can be issued.

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] [CALC 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**.  F3▾ Link 1:Send 2:Receive 3:Send to TI-92 4:Send OS 5:Receive OS 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.

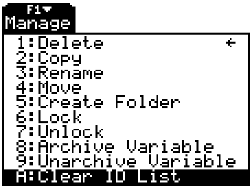
- If an ID is collected from a device twice, the duplicate ID is automatically deleted from the list.

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**.



F1▾
Manage
1:Delete ←
2:Copy
3:Rename
4:Move
5:Create Folder
6:Lock
7:Unlock
8:Archive Variable
9:Unarchive Variable
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.

Link Transmission Table

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

5

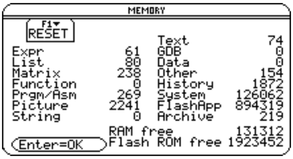
Memory and Variable Management

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.)



```
                        Text          74
Expr          61        GDB            0
List          80        Data           0
Matrix       238        Other        154
Function       0        History     1872
Prgm/Asm     269        System    126062
Picture     2241        FlashApp  894319
String         0        Archive      219
            RAM free              131312
            Flash ROM free       1923452
```

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.

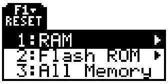
Note: To display the size of individual variables and determine if they are in the user data archive, use the **VAR-LINK** screen.

To close the screen, press ENTER. To reset the memory, use the following procedure.

Resetting the Memory

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.

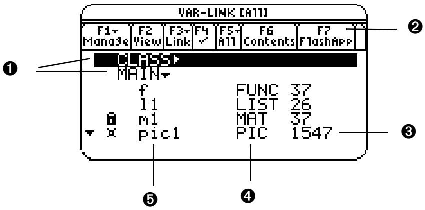
4. Press ENTER to acknowledge the message.

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.



```
 CLASS▸
 MAIN▾
     f       FUNC 37
     l1      LIST 26
  🔒 m1      MAT  37
▾ ¤  pic1    PIC  1547
```

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).
▾	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.
[ESC]	Return to the current application without pasting the highlighted name.

Manipulating Variables and Folders with VAR-LINK

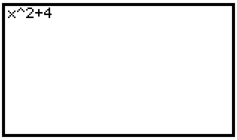
On the **VAR-LINK** screen, you can show the contents of a variable. You can also select one or more listed items and manipulate them by using the operations in this section.

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]

   ```
   x^2+4
   ```

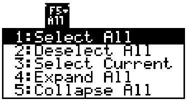
   If you highlight a folder, the screen shows the number of variables in that folder.

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.
All folders and all variables  F5▾ All 1:Select All 2:Deselect All 3:Select Current 4:Expand All 5:Collapse All	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

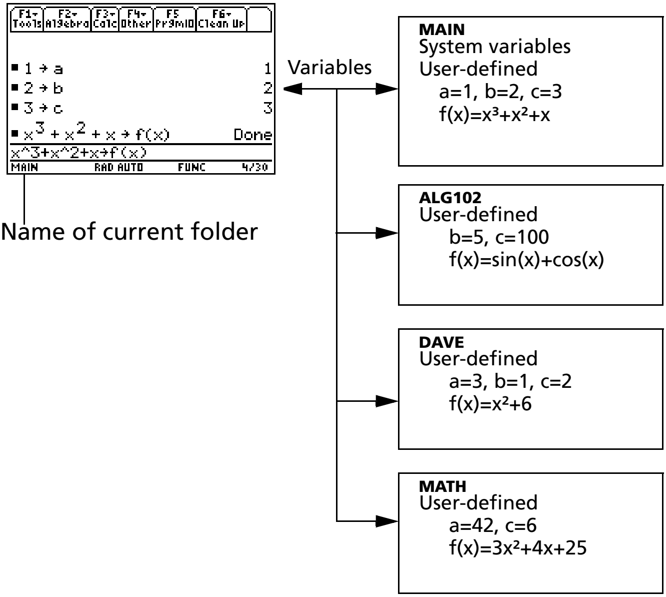
Folders give you a convenient way to manage variables by organizing them into related groups.

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.



```
■ 1 → a                    1
■ 2 → b                    2
■ 3 → c                    3
■ x^3 + x^2 + x → f(x)  Done
x^3+x^2+x→f(x)
MAIN     RAD AUTO    FUNC   4/30
```

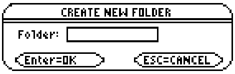
You cannot create a folder within another folder.

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.

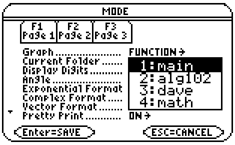
Setting the Current Folder from the MODE Dialog Box

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].


MODE

Graph .... FUNCTION →
Current Folder ....
Display Digits ....
Angle ....
Exponential Format
Complex Format ....
Vector Format ....
Pretty Print .... ON →

1:main
2:alg102
3:dave
4:math

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

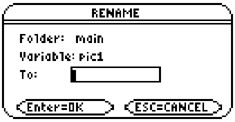
Remember, if you use [F4] to select a folder, the variables in that folder are selected automatically. As necessary, use [F4] to deselect individual variables.

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.


RENAME

Folder: main
Variable: pic1
To:

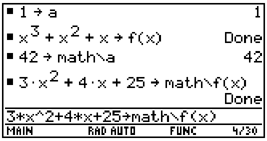
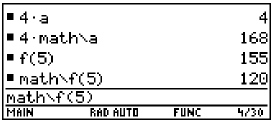
Using Variables in Different Folders

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:

If Current Folder = MAIN	Folders and Variables
 ■ 1 → a 1 ■ $x^3 + x^2 + x$ → f(x) Done ■ 42 → math\a 42 ■ $3 \cdot x^2 + 4 \cdot x + 25$ → math\f(x) Done 3*x^2+4*x+25→math\f(x) MAIN RAD AUTO FUNC 4/30	**MAIN** a=1 f(x)=x^3+x^2+x
 ■ 4·a 4 ■ 4·math\a 168 ■ f(5) 155 ■ math\f(5) 120 math\f(5) MAIN RAD AUTO FUNC 4/30	**MATH** a=42 f(x)=$3x^2+4x+25$

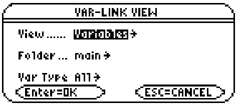
To see a list of existing folders and variables, press [2nd] [VAR-LINK]. On the **VAR-LINK** screen, you can highlight a variable and press [ENTER] to paste that variable name to the open application's entry line. If you paste a variable name that is not in the current folder, the pathname (*folderName\variableName*) 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**.

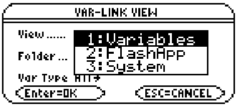


VAR-LINK VIEW
View ...... Variables→
Folder ... main→
Var Type All→
Enter=OK ESC=CANCEL

2. Highlight the setting you want to change, and press ⊳. This displays a menu of valid choices. (To cancel a menu, press [ESC].)

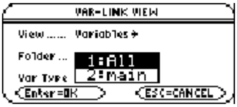
   View — Allows you to choose variables, Flash applications, or system variables to view.

   Note: To list system variables (window variables, etc.), select **3:System**



VAR-LINK VIEW
View 1:Variables
Folder ... 2:FlashApp
Var Type 3:System
Enter=OK ESC=CANCEL

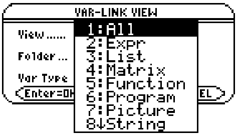
   Folder — Always lists **1:All** and **2:main**, but lists other folders only if you have created them.



VAR-LINK VIEW
View ...... Variables→
Folder ... 1:All
Var Type 2:main
Enter=OK ESC=CANCEL

Var Type — Lists the valid variable types.

↓ — indicates that you can scroll for additional variable types.



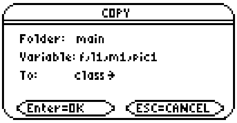
3. Select the new setting.
4. When you are back on the **VAR-LINK VIEW** screen, press [ENTER].

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.



Folder: main
Variable: f,l1,m1,pic1
To: class →

4. Press [ENTER]. The copied or moved variables retain their original names.

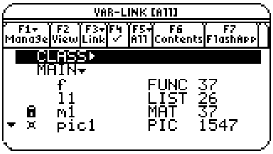
   Note: To copy a variable to a different name in the same folder, use [STO▸] (such as a1→a2) or the CopyVar command from the Home screen.

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**.



```
  CLASS▸
  MAIN▾
     f          FUNC 37
     l1         LIST 26
  🔒 m1         MAT  37
▾ ¤ pic1        PIC  1547
```

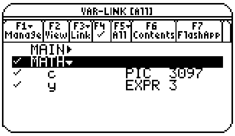
🔒 indicates a locked variable or folder in RAM.

¤ indicates an archived variable, which is locked automatically.

Deleting a Folder from the VAR-LINK Screen

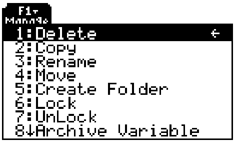
When you delete a folder from the **VAR-LINK** screen, all of the variables in that folder are also deleted. You cannot delete the **MAIN** folder.

1. Press [2nd] [VAR-LINK].
2. Press [F4] to select the folder(s) to delete. (The folder's variables become selected automatically.)

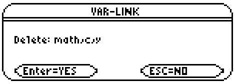


```
   MAIN▸
✓  MATH▾
✓    c          PIC  3097
✓    y          EXPR 3
```

3. Press [F1] **1:Delete** or [←].



4. Press [ENTER] to confirm the deletion of the folder and all its variables.



Delete: math,c,y

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 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

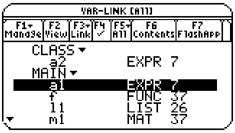
Starting from an application listed above:

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.



```
CLASS ▾
  a2        EXPR 7
MAIN ▾
  a1        EXPR 7
  f         FUNC 37
  l1        LIST 26
▾ m1        MAT  37
```

4. Press [ENTER] to paste the variable name.

   Note: This pastes the variable's name, not its contents. Use [2nd] [RCL], instead of [2nd] [VAR-LINK], to recall a variable's contents.

```
sin(a1|
```

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.

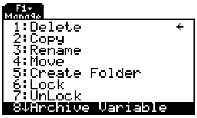
From the VAR-LINK Screen

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.)

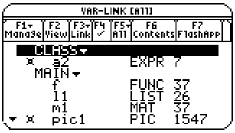
   Note: To select a single variable, highlight it. To select multiple variables, highlight each variable and press [F4] ✓.
3. Press [F1] and select either:

   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*, …

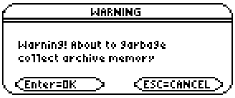
Unarchiv *variable1*, *variable2*, …

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:



WARNING

Warnin9! About to 9arba9e collect archive memory

- 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:

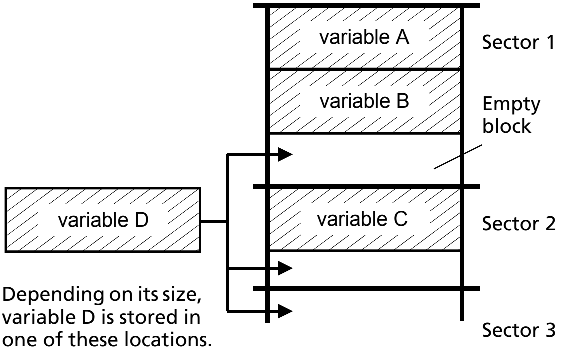
- Lets you know why an archive will take longer than usual. It also alerts you that the archive may fail if there is not enough memory.
- Can alert you when a program is caught in a loop that repetitively fills the user data archive. Cancel the archive and investigate the reason.

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.

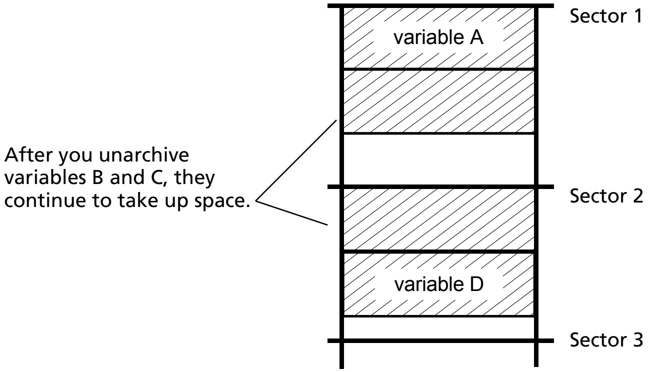


This process continues to the end of the last sector. Depending on the size of individual variables, the empty blocks may account for a significant amount of space.

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.


Sector 1
variable A
After you unarchive variables B and C, they continue to take up space.
Sector 2
variable D
Sector 3

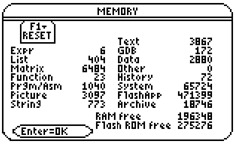
Unarchived variables are “marked for deletion,” meaning they will be deleted during the next garbage collection.

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.

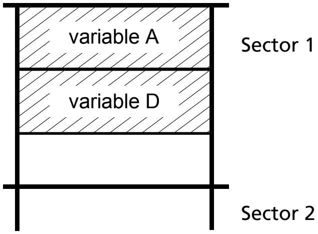


```
                            Text          3867
Expr            6           GDB            172
List          404           Data          2880
Matrix       6484           Other            0
Function       23           History         72
Pr9m/Asm     1040           System       65724
Picture      3097           FlashApp    471399
Strin9        773           Archive      18746
                      RAM free          196348
                      Flash ROM free    275276
```

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:

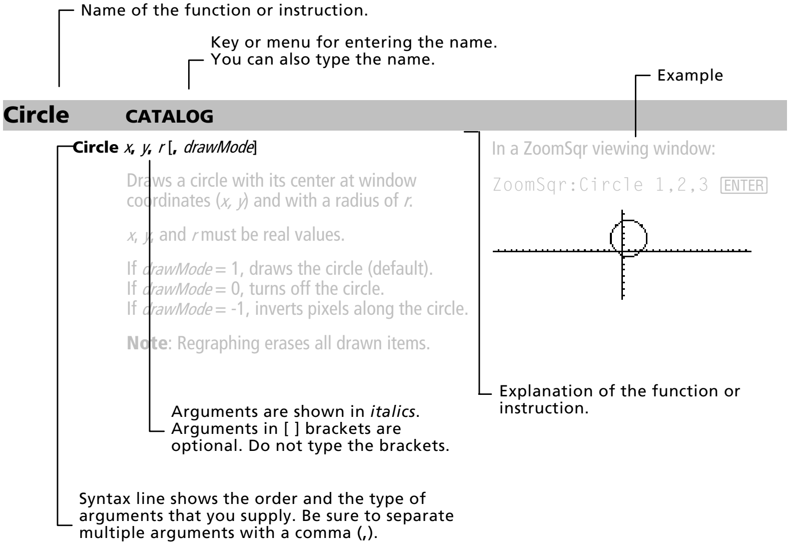
1. Use the **VAR-LINK** screen (2nd [VAR-LINK]) to determine the size of the archived variable that you want to access.
2. Use the **MEMORY** screen (2nd [MEM]) to check the RAM free size.
3. Free up the needed amount of memory by:

- 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.

Appendix A: Functions and Instructions

This section describes the syntax and action of each TI-89 Titanium/ Voyage™ 200 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.



Name of the function or instruction.

Key or menu for entering the name. You can also type the name.

Example

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.

In a ZoomSqr viewing window:

```
ZoomSqr:Circle 1,2,3 [ENTER]
```

Explanation of the function or instruction.

Arguments are shown in *italics*. Arguments in [ ] brackets are optional. Do not type the brackets.

Syntax line shows the order and the type of arguments that you supply. Be sure to separate multiple arguments with a comma (,).

Quick-Find Locator

This section lists the TI-89 Titanium / Voyage™ 200 functions and instructions in functional groups along with the page numbers where they are described.

Algebra

| ("**with**")	277	**cFactor()**	158	**comDenom()**	161
cSolve()	166	**cZeros**	170	**expand()**	184
factor()	186	**getDenom()**	192	**getNum()**	193
nSolve()	214	**propFrac()**	221	**randPoly()**	227
solve()	243	**tCollect()**	253	**tExpand()**	253
zeros()	260				

Calculus

∫**()** (integrate)	272	Π**()** (product)	273	Σ**()** (sum)	273
arcLen()	156	**avgRC()**	157	***d()***	172
deSolve()	174	**fMax()**	188	**fMin()**	188
limit()	200	**nDeriv()**	210	**nInt()**	212
' (prime)	275	**seq()**	234	**taylor()**	252

Graphics

AndPic	155	**BldData**	158	**Circle**	159
ClrDraw	160	**ClrGraph**	160	**CyclePic**	170
DrawFunc	179	**DrawInv**	179	**DrawParm**	179
DrawPol	179	**DrawSlp**	180	**DrwCtour**	180
FnOff	188	**FnOn**	188	**Graph**	195
Line	201	**LineHorz**	201	**LineTan**	201
LineVert	202	**NewPic**	211	**PtChg**	221
PtOff	221	**PtOn**	222	**ptTest()**	222
PtText	222	**PxlChg**	222	**PxlCrcl**	222
PxlHorz	222	**PxlLine**	223	**PxlOff**	223
PxlOn	223	**pxlTest()**	223	**PxlText**	223
PxlVert	224	**RclGDB**	227	**RclPic**	227
RplcPic	231	**Shade**	238	**StoGDB**	247
StoPic	248	**Style**	248	**Trace**	256
XorPic	259	**ZoomBox**	261	**ZoomData**	262
ZoomDec	262	**ZoomFit**	263	**ZoomIn**	263
ZoomInt	263	**ZoomOut**	264	**ZoomPrev**	264
ZoomRcl	264	**ZoomSqr**	264	**ZoomStd**	265
ZoomSto	265	**ZoomTrig**	265		

Lists

+ (add)	265	− (subtract)	266	∗ (multiply)	266
⁄ (divide)	267	⁻ (negate)	269	**^** (power)	268
augment()	156	**crossP()**	165	**cumSum()**	168
dim()	177	**dotP()**	178	**exp▸list()**	184
left()	200	**∆list()**	202	**list▸mat()**	202
mat▸list()	207	**max()**	207	**mid()**	208
min()	209	**newList()**	211	**polyEval()**	219
product()	220	**right()**	229	**rotate()**	230
shift()	239	**SortA**	246	**SortD**	246
sum()	249				

Math

+ (add)	265	− (subtract)	266	∗ (multiply)	266
⁄ (divide)	267	$^{-}$ (negate)	269	**%** (percent)	269
! (factorial)	271	√**()** (sqr. root)	273	**^** (power)	268
° (degree)	274	∠ (angle)	274	**°, ', "**	275
_ (underscore)	275	▶ (convert)	276	**10^()**	276
0b, 0h	278	**▶Bin**	157	**▶Cylind**	170
▶DD	172	**▶Dec**	173	**▶DMS**	178
▶Hex	196	**▶Polar**	219	**▶Rect**	228
▶Sphere	246	**abs()**	154	**and**	154
angle()	155	**approx()**	156	**ceiling()**	158
conj()	162	**cos**	163	**cos**$^{-1}$**()**	163
cosh()	164	**cosh**$^{-1}$**()**	164	**cot()**	164
cot^{-1}**()**	165	**coth()**	165	**coth**$^{-1}$**()**	165
csc()	165	**csc**$^{-1}$**()**	166	**csch()**	166
cosh^{-1}**()**	166	**E**	181	**e^*()***	181
exact()	183	**floor()**	187	**fPart()**	190
gcd()	190	**imag()**	197	**int()**	198
intDiv()	198	**iPart()**	199	**isPrime()**	199
lcm()	200	**ln()**	203	**log()**	204
max()	207	**min()**	209	**mod()**	209
nCr()	210	**nPr()**	213	**P▶Rx()**	216
P▶Ry()	216	**r** (radian)	274	**R▶Pθ()**	226
R▶Pr()	226	**real()**	227	**remain()**	228
rotate()	230	**round()**	230	**sec()**	232
sec^{-1}**()**	232	**sech()**	232	**sech**$^{-1}$**()**	233
shift()	239	**sign()**	240	**sin()**	241
sin^{-1}**()**	241	**sinh()**	242	**sinh**$^{-1}$**()**	242
tan()	251	**tan**$^{-1}$**()**	251	**tanh()**	251
tanh^{-1}**()**	252	**tmpCnv()**	254	**∆tmpCnv()**	255
x^{-1}	276				

Matrices

+ (add)	265	− (subtract)	266	∗ (multiply)	266
⁄ (divide)	267	$^{-}$ (negate)	269	**.+** (dot add)	268
.- (dot subt.)	268	**.** (dot mult.)	269	**. /** (dot divide)	269
.^ (dot power)	269	**^** (power)	268	**augment()**	156
colDim()	161	**colNorm()**	161	**crossP()**	165
cumSum()	168	**det()**	176	**diag()**	176
dim()	177	**dotP()**	178	**eigVc()**	181
eigVl()	182	**Fill**	187	**identity()**	196
list▶mat()	202	**LU**	206	**mat▶list()**	207
max()	207	**mean()**	207	**median()**	207
min()	209	**mRow()**	209	**mRowAdd()**	209
newMat()	211	**norm()**	213	**product()**	220
QR	224	**randMat()**	226	**ref()**	228
rowAdd()	231	**rowDim()**	231	**rowNorm()**	231
rowSwap()	231	**rref()**	232	**simult()**	240
stdDev()	247	**subMat()**	249	**sum()**	249
T	250	**unitV()**	257	**variance()**	257
x^{-1}	276				

Programming

=	270	≠	270	**<**	270
≤	271	**>**	271	≥	271
# (indirection)	273	→ (store)	277	© (comment)	278
and	154	**ans()**	156	**Archive**	156
checkTmr()	159	**ClockOff**	160	**ClockOn**	160
ClrErr	160	**ClrGraph**	160	**ClrHome**	161
ClrIO	161	**ClrTable**	161	**CopyVar**	162
CustmOff	169	**CustmOn**	169	**Custom**	169
Cycle	169	**dayOfWk()**	172	**Define**	173
DelFold	174	**DelVar**	174	**Dialog**	177
Disp	177	**DispG**	178	**DispHome**	178
DispTbl	178	**DropDown**	180	**Else**	182
ElseIf	182	**EndCustm**	182	**EndDlog**	182
EndFor	182	**EndFunc**	182	**EndIf**	182
EndLoop	182	**EndPrgm**	182	**EndTBar**	182
EndTry	182	**EndWhile**	182	**entry()**	183
Exec	183	**Exit**	184	**For**	189
format()	189	**Func**	190	**Get**	190
GetCalc	191	**getConfg()**	191	**getDate()**	192
getDtFmt()	192	**getDtStr()**	192	**getFold()**	192
getKey()	193	**getMode()**	193	**getTime()**	193
getTmFmt()	193	**getTmStr()**	194	**getTmZn()**	194
getType()	194	**getUnits()**	195	**Goto**	195
If	196	**Input**	197	**InputStr**	198
isClkOn()	199	**Item**	199	**Lbl**	199
left()	200	**Local**	204	**Lock**	204
Loop	205	**MoveVar**	209	**NewFold**	211
NewProb	212	**not**	213	**or**	215
Output	215	**part()**	216	**PassErr**	218
Pause	218	**PopUp**	220	**Prgm**	220
Prompt	221	**Rename**	229	**Request**	229
Return	229	**right()**	229	**Send**	233
SendCalc	233	**SendChat**	233	**setDate()**	234
setDtFmt()	234	**setFold()**	234	**setGraph()**	234
setMode()	235	**setTable()**	236	**setTime()**	236
setTmFmt()	237	**setTmZn()**	237	**startTmr()**	246
setUnits()	237	**Stop**	247	**Style**	248
switch()	249	**Table**	250	**Text**	253
Then	253	**timeCnv()**	253	**Title**	254
Toolbar	255	**Try**	256	**Unarchiv**	257
Unlock	257	**when()**	258	**While**	258
xor	259				

Statistics

! (factorial)	271	**BldData**	158	**CubicReg**	168
cumSum()	168	**ExpReg**	186	**LinReg**	202
LnReg	203	**Logistic**	205	**mean()**	207
median()	207	**MedMed**	208	**nCr()**	210
NewData	210	**NewPlot**	212	**nPr()**	213
OneVar	214	**PlotsOff**	219	**PlotsOn**	219
PowerReg	220	**QuadReg**	225	**QuartReg**	225
rand()	226	**randNorm()**	226	**RandSeed**	227
ShowStat	240	**SinReg**	243	**SortA**	246
SortD	246	**stdDev()**	247	**TwoVar**	256
variance()	257				

Strings

& (append)	272	# (indirection)	273	**char()**	159
dim()	177	**expr()**	185	**format()**	189
inString()	198	**left()**	200	**mid()**	208
ord()	215	**right()**	229	**rotate()**	230
shift()	239	**string()**	248		

Alphabetical Listing of Operations

Operations whose names are not alphabetic (such as +, !, and >) are listed at the end of this appendix, starting on page 265. 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] $\{\frac{\pi}{2} \;\; \frac{\pi}{3}\}$

abs(2−3*i*) [ENTER] $\sqrt{13}$

abs(z) [ENTER] |z|

abs(x+y*i*) [ENTER] $\sqrt{x^2+y^2}$

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*]

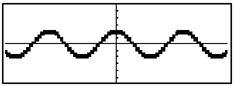
Displays the Graph screen and logically “ANDS” the picture stored in *picVar* and the current graph screen at pixel coordinates *(row, column)*.

picVar must be a picture type.

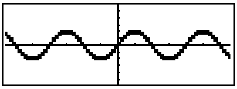
Default coordinates are (0,0), which is the upper left corner of the screen.

In function graphing mode and Y= Editor:

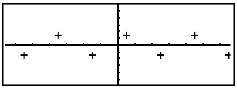
```
y1(x) = cos(x) ⊙
🖩   [2nd][F6] Style = 3:Square
⌨   [F6] Style = 3:Square
[F2] Zoom = 7:ZoomTrig
[F1] = 2:Save Copy As...
Type = Picture, Variable =
PIC1
```



```
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.

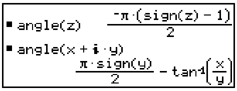
In Degree angle mode:

```
angle(0+2i) [ENTER]            90
```

In Radian angle mode:

angle(1+*i*) [ENTER] $\frac{\pi}{4}$

```
angle(z) [ENTER]
angle(x+ iy) [ENTER]
```



■ angle(z) $\frac{-\pi\cdot(\mathrm{sign}(z)-1)}{2}$

■ angle(x + i·y) $\frac{\pi\cdot\mathrm{sign}(y)}{2}-\tan^{-1}\left(\frac{x}{y}\right)$

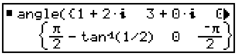
---

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+2i,3+0i,0−4i}) [ENTER]
```



■ angle({1 + 2·i 3 + 0·i 0▸
$\left\{\frac{\pi}{2}-\tan^{-1}(1/2)\quad 0\quad \frac{-\pi}{2}\right\}$

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

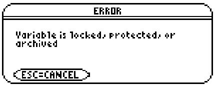
Archive *var1* [, *var2*] [, *var3*] ...

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]
```


ERROR

Variable is locked, protected, or archived

```
[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
$\dfrac{\texttt{f(x+h) - f(x)}}{\texttt{h}}$

`avgRC(sin(x),x,h)|x=2` ENTER
$\dfrac{\texttt{sin(h+2) - sin(2)}}{\texttt{h}}$

`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.

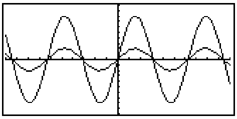
Note: The first time you start the Data/Matrix Editor after using **BldData**, *dataVar* or sysData (depending on the argument you used with **BldData**) is set as the current data variable.

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.

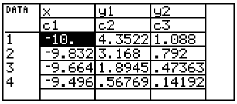
In function graphing mode and Radian angle mode:

```
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]
```

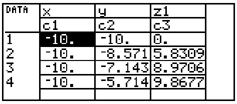


DATA	x	y1	y2
	c1	c2	c3
1	⁻10.	4.3522	1.088
2	⁻9.832	3.168	.792
3	⁻9.664	1.8945	.47363
4	⁻9.496	.56769	.14192

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]
$\begin{bmatrix} 0 & ^-3.\cdot i \\ 2. & 4 \end{bmatrix}$

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]
$a\cdot(a + ^-i)\cdot(a + i)\cdot(x + ^-i)\cdot(x + i)$

`cFactor(x^2+4/9)` [ENTER]
$\frac{(3\cdot x + ^-2\cdot i)\cdot(3\cdot x + 2\cdot i)}{9}$

`cFactor(x^2+3)` [ENTER] $x^2 + 3$

`cFactor(x^2+a)` [ENTER] $x^2 + a$

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
```

$$a\cdot(a^2+1)\cdot(x+{}^-i)\cdot(x+i)$$

```
cFactor(x^2+3,x) ENTER
```

$$(x+\sqrt{3}\cdot i)\cdot(x+{}^-\sqrt{3}\cdot i)$$

```
cFactor(x^2+a,x) ENTER
```

$$(x+\sqrt{a}\cdot{}^-i)\cdot(x+\sqrt{a}\cdot i)$$

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
```

$$x^5+4\cdot x^4+5\cdot x^3-6\cdot x-3$$

```
cFactor(ans(1),x) ENTER
```

$$(x-.965)\cdot(x+.612)\cdot(x+2.13)\cdot(x+1.11-1.07\cdot i)\cdot(x+1.11+1.07\cdot i)$$

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*.

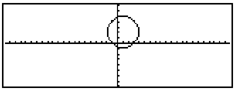
x, *y*, and *r* must be real values.

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()**.

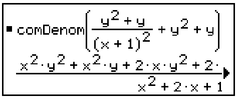
```
[1,⁻2,3;4,5,⁻6]→mat ENTER
                          [1  ⁻2   3]
                          [4   5  ⁻6]
colNorm(mat) ENTER                 9
```

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
```

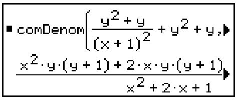


■ comDenom$\left(\frac{y^2+y}{(x+1)^2}+y^2+y\right)$

$\frac{x^2\cdot y^2+x^2\cdot y+2\cdot x\cdot y^2+2\cdot ▸}{x^2+2\cdot x+1}$

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.

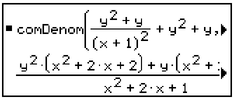
```
comDenom((y^2+y)/(x+1)
^2+y^2+y,x) ENTER
```



■ comDenom$\left(\frac{y^2+y}{(x+1)^2}+y^2+y,\right.$▸

$\frac{x^2\cdot y\cdot(y+1)+2\cdot x\cdot y\cdot(y+1)}{x^2+2\cdot x+1}$▸

```
comDenom((y^2+y)/(x+1)
^2+y^2+y,y) ENTER
```

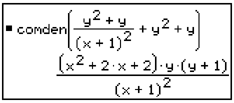


■ comDenom$\left(\frac{y^2+y}{(x+1)^2}+y^2+y,\right.$▸

$\frac{y^2\cdot(x^2+2\cdot x+2)+y\cdot(x^2+}{x^2+2\cdot x+1}$▸

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
```



■ 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:

$$\cos(B) = \begin{bmatrix} \cos(\lambda_1) & 0 & \dots & 0 \\ 0 & \cos(\lambda_2) & \dots & 0 \\ 0 & 0 & \dots & 0 \\ 0 & 0 & \dots & \cos(\lambda_n) \end{bmatrix}$$

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]
$\{\frac{\pi}{2}$ `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]

$$\begin{bmatrix} 1.734...+.064...\cdot \boldsymbol{i} & {}^{-}1.490...+2.105...\cdot \boldsymbol{i} & ... \\ {}^{-}.725...+1.515...\cdot \boldsymbol{i} & .623...+.778...\cdot \boldsymbol{i} & ... \\ {}^{-}2.083...+2.632...\cdot \boldsymbol{i} & 1.790...-1.271...\cdot \boldsymbol{i} & ... \end{bmatrix}$$

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]

$$\begin{bmatrix} 421.255 & 253.909 & 216.905 \\ 327.635 & 255.301 & 202.958 \\ 226.297 & 216.623 & 167.628 \end{bmatrix}$$

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]

$$\begin{bmatrix} 2.525...+1.734...\cdot \boldsymbol{i} & {}^{-}.009...-1.490...\cdot \boldsymbol{i} & ... \\ .486...-.725...\cdot \boldsymbol{i} & 1.662...+.623...\cdot \boldsymbol{i} & ... \\ {}^{-}.322...-2.083...\cdot \boldsymbol{i} & 1.267...+1.790...\cdot \boldsymbol{i} & ... \end{bmatrix}$$

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]

$$\frac{1}{\tan(1)} \quad {}^{-}.584... \quad \frac{1}{\tan(3)}$$

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 $\frac{\pi}{4}$

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

$\frac{1}{\text{tanh(1)}}$ 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

$\frac{-\ln(3)}{2}$ .518… $\frac{\ln(7/5)}{2}$

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 $\frac{2\cdot\sqrt{3}}{3}$

$\mathbf{csc^{-1}()}$ MATH/Trig menu

$\mathbf{csc^{-1}(}$*expression1***)** ⇒ *expression*
$\mathbf{csc^{-1}(}$*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:

$\mathtt{csc^{-1}(1)}$ [ENTER] $\quad 90$

In Radian angle mode:

$\mathtt{csc^{-1}(\{1,4,6\})}$ [ENTER]

$$\left\{\frac{\pi}{2}\ \mathtt{sin^{-1}(1/4)}\ \mathtt{sin^{-1}(1/6)}\right\}$$

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] $\quad \frac{1}{\mathtt{sinh(3)}}$

`csch({1,2.1,4})` [ENTER]

$$\left\{\frac{1}{\mathtt{sinh(1)}}\ .248\ldots\ \frac{1}{\mathtt{sinh(4)}}\right\}$$

$\mathbf{csch^{-1}()}$ MATH/Hyperbolic menu

$\mathbf{csch^{-1}(}$*expression1***)** ⇒ *expression*
$\mathbf{csch^{-1}(}$*list1***)** ⇒ *list*

Returns the inverse hyperbolic cosecant of *expression1* or returns a list containing the inverse hyperbolic cosecants of each element of *list1*.

$\mathtt{csch^{-1}(1)}$ [ENTER] $\quad \mathtt{sinh^{-1}(1)}$

$\mathtt{csch^{-1}(\{1,2.1,3\})}$ [ENTER]

$$\{\mathtt{sinh^{-1}(1)}\ .459\ldots\ \mathtt{sinh^{-1}(1/3)}\}$$

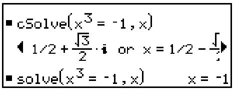
cSolve() MATH/Algebra/Complex menu

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]
```



■ $\mathtt{cSolve}(x^3 = {}^-1, x)$
◀ $1/2 + \frac{\sqrt{3}}{2}\cdot i$ or $x = 1/2 - \frac{\sqrt{\ }}{}$▶
■ $\mathtt{solve}(x^3 = {}^-1, x)$ $\quad x = {}^-1$

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
```

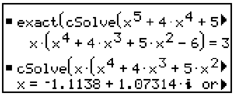
cSolve() starts with exact symbolic methods. Except in EXACT mode, **cSolve()** also uses iterative approximate complex polynomial factoring, if necessary.

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 _ (TI-89: ◆ [_]; Voyage 200: 2nd [_]) at the end of *var*. By default, a variable is treated as a real value.

Display Digits mode in Fix 2:

```
exact(cSolve(x^5+4x^4+5x
  ^3-6x-3=0,x)) [ENTER]
cSolve(ans(1),x) [ENTER]
```



■ $\mathtt{exact}(\mathtt{cSolve}(x^5 + 4\cdot x^4 + 5$▶
$\quad x\cdot(x^4 + 4\cdot x^3 + 5\cdot x^2 - 6) = 3$
■ $\mathtt{cSolve}(x\cdot(x^4 + 4\cdot x^3 + 5\cdot x^2$▶
$x = {}^-1.1138 + 1.07314\cdot i$ or▶

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.

z is treated as real:

`cSolve(conj(z)=1+`***i***`,z)` [ENTER]

$z=1+\boldsymbol{i}$

z_ is treated as complex:

`cSolve(conj(z_)=1+`***i***`,z_)` [ENTER]

$z\_=1-\boldsymbol{i}$

---

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.

Note: The following examples use an underscore _
🖩 [◆] [_]
⌨ [2nd] [_] so that the variables will be treated as complex.

Complex solutions can include both real and non-real solutions, as in the example to the right.

```
cSolve(u_*v_-u_=v_ and
v_^2=-u_,{u_,v_}) [ENTER]
```

$$u\_=1/2+\frac{\sqrt{3}}{2}\cdot\boldsymbol{i} \text{ and } v\_=1/2-\frac{\sqrt{3}}{2}\cdot\boldsymbol{i}$$
$$\text{or } u\_=1/2-\frac{\sqrt{3}}{2}\cdot\boldsymbol{i} \text{ and } v\_=1/2+\frac{\sqrt{3}}{2}\cdot\boldsymbol{i}$$
$$\text{or } u\_=0 \text{ and } v\_=0$$

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
$$u\_=\frac{^-(\sqrt{1-4\cdot c\_}-1)^2}{4} \text{ and } v\_=\frac{^-(\sqrt{1-4\cdot c\_}-1)}{2}$$
$$\text{or } u\_=0 \text{ and } v\_=0$$

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.

```
cSolve(u_*v_-u_=v_ and
v_^2=-u_,{u_,v_,w_}) [ENTER]
```

$$u\_=1/2+\frac{\sqrt{3}}{2}\cdot\boldsymbol{i} \text{ and } v\_=1/2-\frac{\sqrt{3}}{2}\cdot\boldsymbol{i} \text{ and } w\_=@1$$
or
$$u\_=1/2-\frac{\sqrt{3}}{2}\cdot\boldsymbol{i} \text{ and } v\_=1/2+\frac{\sqrt{3}}{2}\cdot\boldsymbol{i} \text{ and } w\_=@1$$
$$\text{or } u\_=0 \text{ and } v\_=0 \text{ and } w\_=@1$$

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 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.

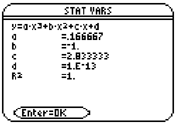
All the lists must have equal dimensions except for *list5*.

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.

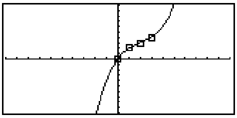
```
{0,1,2,3}→L1 [ENTER]        {0 1 2 3}
{0,2,3,4}→L2 [ENTER]        {0 2 3 4}
CubicReg L1,L2 [ENTER]           Done
ShowStat [ENTER]
```



```
STAT VARS
y=a·x³+b·x²+c·x+d
a          =.166667
b          =⁻1.
c          =2.833333
d          =1.E⁻13
R²         =1.
```

```
[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. The three pictures (3) will be displayed automatically—one-half second (.5) between pictures, for four cycles (4), and forward and backwards (⁻1).

▸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·√2 ∠π/4  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
```

$$\begin{bmatrix} 1/2-\frac{\sqrt{3}}{2}\cdot \boldsymbol{i} & 1/2+\frac{\sqrt{3}}{2}\cdot \boldsymbol{i} \\ 1/2+\frac{\sqrt{3}}{2}\cdot \boldsymbol{i} & 1/2-\frac{\sqrt{3}}{2}\cdot \boldsymbol{i} \\ 0 & 0 \end{bmatrix}$$

Extract row 2:

```
ans(1)[2] ENTER
```

$$\begin{bmatrix} 1/2+\frac{\sqrt{3}}{2}\cdot \boldsymbol{i} & 1/2-\frac{\sqrt{3}}{2}\cdot \boldsymbol{i} \end{bmatrix}$$

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
```

$$\begin{bmatrix} 1/2-\frac{\sqrt{3}}{2}\cdot \boldsymbol{i} & 1/2+\frac{\sqrt{3}}{2}\cdot \boldsymbol{i} & @1 \\ 1/2+\frac{\sqrt{3}}{2}\cdot \boldsymbol{i} & 1/2-\frac{\sqrt{3}}{2}\cdot \boldsymbol{i} & @1 \\ 0 & 0 & @1 \end{bmatrix}$$

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
```

$$\begin{bmatrix} \frac{e^{w\_}}{2}+1/2\cdot \boldsymbol{i} & \frac{e^{w\_}-\boldsymbol{i}}{2} \end{bmatrix}$$

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
```

$$\begin{bmatrix} .494\ldots & ^-.703\ldots \end{bmatrix}$$

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
```

$$\begin{bmatrix} .149\ldots+4.89\ldots\cdot \boldsymbol{i} & 1.588\ldots+1.540\ldots\cdot \boldsymbol{i} \end{bmatrix}$$

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+x^2 - 4·x+6

right(ans(1))→temp ENTER
           (@1·x+@2)·e^-x+x^2 - 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.

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.

```
deSolve(y'=(cos(y))^2*x,x,y) ENTER
```

$$\tan(y)=\frac{x^{2}}{2}+@3$$

```
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:

🖩 ◆ STO▸

🖥 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
```

$$y=\frac{2^{2/3}\cdot(3\cdot t)^{4/3}}{4}\text{ and }t\geq 0$$

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·x³ - 55·x² + 12·x - 1)
```

```
[1E20,1;0,1]→mat1
```

$$\begin{bmatrix}1.\text{E}20 & 1\\ 0 & 1\end{bmatrix}$$

```
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]
```

$$\begin{bmatrix}2 & 0 & 0\\ 0 & 4 & 0\\ 0 & 0 & 6\end{bmatrix}$$

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]
```

$$\begin{bmatrix}4 & 6 & 8\\ 1 & 2 & 3\\ 5 & 7 & 9\end{bmatrix}$$

```
diag(ans(1)) [ENTER]          [4 2 9]
```

Dialog CATALOG

Dialog
  block
EndDlog

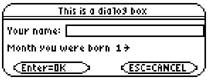
Generates a dialog box when the program is executed.

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
```



This is a dialog box
Your name:
Month you were born 1→

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

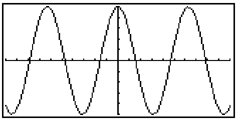
DispG

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
```

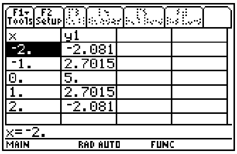
DispTbl CATALOG

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]
```



x	y1		
⁻2.	⁻2.081		
⁻1.	2.7015		
0.	5.		
1.	2.7015		
2.	⁻2.081		

x=⁻2.

▸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 275 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

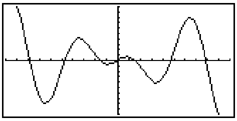
DrawFunc *expression*

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*

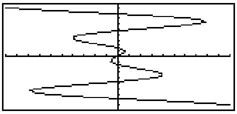
Draws the inverse of *expression* by plotting x values on the y axis and y values on the x axis.

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*]

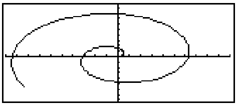
Draws the parametric equations *expression1* and *expression2*, using t as the independent variable.

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*]

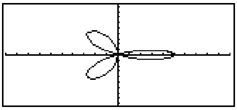
Draws the polar graph of *expression*, using θ as the independent variable.

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

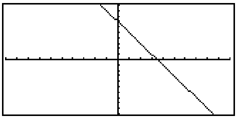
DrawSlp *x1*, *y1*, *slope*

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*

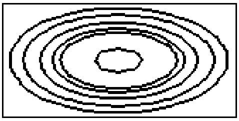
Draws contours on the current 3D graph at the z values specified by *expression* or *list*. The 3D graphing mode must already be set. **DrwCtour** automatically sets the graph format style to CONTOUR LEVELS.

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:
  🖩 I
  🖩 F

- Press X, Y, or Z to look down the corresponding axis.

E   [EE] **key**   [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^()   [◆] [eˣ] **key**   [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]                       e⁹
```

---

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, … , 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 196.

ElseIf **CATALOG** See also **If**, page 196.

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 169.

EndDlog See **Dialog**, page 177.

EndFor See **For**, page 189.

EndFunc See **Func**, page 190.

EndIf See **If**, page 196.

EndLoop See **Loop**, page 205.

EndPrgm See **Prgm**, page 220.

EndTBar See **ToolBar**, page 255.

EndTry See **Try**, page 256.

EndWhile See **While**, page 258.

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 $\boxed{\text{ENTER}}$ is equivalent to executing **entry(1)**.

On the Home screen:

`1+1/x` $\boxed{\text{ENTER}}$	$\frac{1}{x}+1$
`1+1/entry(1)` $\boxed{\text{ENTER}}$	$2-\frac{1}{x+1}$
$\boxed{\text{ENTER}}$	$\frac{1}{2\cdot(2\cdot x+1)}+3/2$
$\boxed{\text{ENTER}}$	$5/3-\frac{1}{3\cdot(3\cdot x+2)}$
`entry(4)` $\boxed{\text{ENTER}}$	$\frac{1}{x}+1$

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)` $\boxed{\text{ENTER}}$	$1/4$
`exact(.333333)` $\boxed{\text{ENTER}}$	$\frac{333333}{1000000}$
`exact(.33333,.001)`	$1/3$
`exact(3.5x+y)` $\boxed{\text{ENTER}}$	$\frac{7\cdot x}{2}+y$
`exact({.2,.33,4.125})` $\boxed{\text{ENTER}}$	$\{1/5 \;\; \frac{33}{100} \;\; 33/8\}$

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*

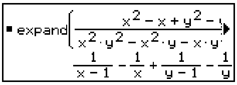
expand(*expression1***)** returns *expression1* expanded with respect to all its variables. The expansion is polynomial expansion for polynomials and partial fraction expansion for rational expressions.

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
```

$x^2 + 2\cdot x\cdot y + 2\cdot x + y^2 + 2\cdot y + 1$

```
expand((x^2-x+y^2-y)/(x^2*y^2
 -x^2*y-x*y^2+x*y)) ENTER
```



■ expand $\left(\frac{x^2 - x + y^2 - y}{x^2\cdot y^2 - x^2\cdot y - x\cdot y}\right.$ ▸

$\frac{1}{x-1} - \frac{1}{x} + \frac{1}{y-1} - \frac{1}{y}$

---

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.

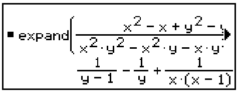
```
expand((x+y+1)^2,y) ENTER
```

$y^2 + 2\cdot y\cdot (x+1) + (x+1)^2$

```
expand((x+y+1)^2,x) ENTER
```

$x^2 + 2\cdot x\cdot (y+1) + (y+1)^2$

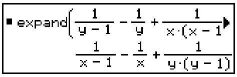
```
expand((x^2-x+y^2-y)/(x^2*y^2
 -x^2*y-x*y^2+x*y),y) ENTER
```



■ expand $\left(\frac{x^2 - x + y^2 - y}{x^2\cdot y^2 - x^2\cdot y - x\cdot y}\right.$ ▸

$\frac{1}{y-1} - \frac{1}{y} + \frac{1}{x\cdot(x-1)}$

```
expand(ans(1),x) ENTER
```



■ expand $\left(\frac{1}{y-1} - \frac{1}{y} + \frac{1}{x\cdot(x-1}\right.$ ▸

$\frac{1}{x-1} - \frac{1}{x} + \frac{1}{y\cdot(y-1)}$

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.

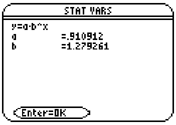
All the lists must have equal dimensions except for *list5*.

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:

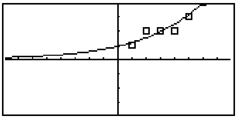
```
{1,2,3,4,5,6,7,8}→L1 ENTER
                                    {1 2 …}
{1,2,2,2,3,4,5,7}→L2 ENTER
                                    {1 2 …}
ExpReg L1,L2 ENTER                     Done
ShowStat ENTER
```



```
STAT VARS
y=a·b^x
a         =.910912
b         =1.279261
Enter=OK
```

```
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

$a\cdot(a-1)\cdot(a+1)\cdot(x-1)\cdot(x+1)$

factor(x^2+1) ENTER x^2+1

factor(x^2−4) ENTER $(x-2)\cdot(x+2)$

factor(x^2−3) ENTER x^2-3

factor(x^2−a) ENTER x^2-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

$a\cdot(a^2-1)\cdot(x-1)\cdot(x+1)$

factor(x^2−3,x) ENTER

$(x+\sqrt{3})\cdot(x-\sqrt{3})$

factor(x^2−a,x) ENTER

$(x+\sqrt{a})\cdot(x-\sqrt{a})$

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
[1,2;3,4]→amatrx [ENTER]       [3   4]
Fill 1.01,amatrx [ENTER]          Done
                             1.01 1.01
amatrx [ENTER]              [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.

```
fMax(1-(x-a)^2-(x-b)^2,x)
ENTER
                                      a+b
                                  x = ───
                                       2

fMax(.5x^3-x-2,x) ENTER       x = ∞
```

Use the "|" operator to restrict the solution interval and/or specify the sign of other undefined variables.

```
fMax(.5x^3-x-2,x)|x≤1 ENTER
                                x = ⁻.816...
```

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.

```
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
```

Note: See also **fMin()** and **max()**.

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.

```
fMin(1-(x-a)^2-(x-b)^2,x)
ENTER
                       x = ∞ or x = ⁻∞

fMin(.5x^3-x-2,x)|x≥1 ENTER  x = 1
```

Use the "|" operator to restrict the solution interval and/or specify the sign of other undefined variables.

```
fMin(a*x^2,x) ENTER
 x = ∞ or x = ⁻∞ or x = 0 or a = 0
```

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.

```
fMin(a*x^2,x)|a>0 and x>1 ENTER
                                x = 1.

fMin(a*x^2,x)|a>0 ENTER        x = 0
```

Note: See also **fMax()** and **min()**.

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.

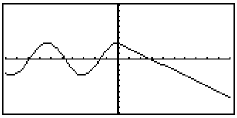
block can be either a single statement or a series of statements separated with the ":" character.

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
```

$$\begin{bmatrix}2 & 4\\6 & 8\end{bmatrix}$$

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 ✋) 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.

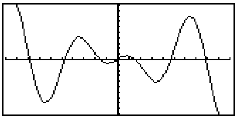
Expressions entered using the **Graph** or **Table** commands are assigned increasing function numbers starting with 1. They can be modified or individually deleted using the edit functions available when the table is displayed by pressing F4 Header. The currently selected Y= functions are ignored.

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.

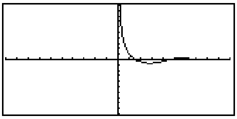
In function graphing mode and ZoomStd window:

```
Graph 1.25a∗cos(a),a ENTER
```



In parametric graphing mode and ZoomStd window:

```
Graph
time,2cos(time)/time,time ENTER
```



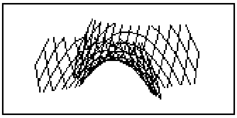
Some valid variations of this instruction are:

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.

```
                                [0 0]
imag([a,b;ic,id]) ENTER         [c d]
```

Indirection See #(), page 273.

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 272.

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*]

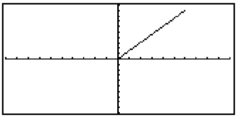
Displays the Graph screen and draws, erases, or inverts a line segment between the window coordinates (*xStart*, *yStart*) and (*xEnd*, *yEnd*), including both endpoints.

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**.

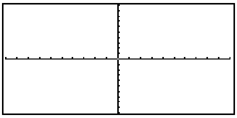
In the ZoomStd window, draw a line and then erase it.

```
Line 0,0,6,9 ENTER
```



HOME
◆ [CALC HOME]

```
Line 0,0,6,9,0 ENTER
```



LineHorz CATALOG

LineHorz *y* [, *drawMode*]

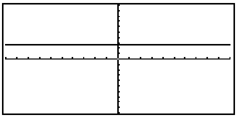
Displays the Graph screen and draws, erases, or inverts a horizontal line at window position *y*.

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.

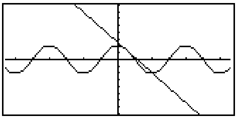
Note: In the example shown, *expression1* is graphed separately. **LineTan** does not graph *expression1*.

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*]

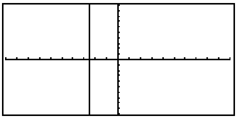
Displays the Graph screen and draws, erases, or inverts a vertical line at window position *x*.

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.

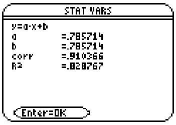
All the lists must have equal dimensions except for *list5*.

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:

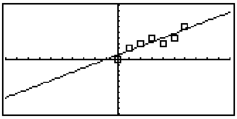
```
{0,1,2,3,4,5,6}→L1 [ENTER]
                              {0 1 2 ...}
{0,2,3,4,3,4,6}→L2 [ENTER]
                              {0 2 3 ...}
LinReg L1,L2 [ENTER]                 Done
ShowStat [ENTER]
```



```
STAT VARS
y=a·x+b
a        =.785714
b        =.785714
corr     =.910366
R²       =.828767
Enter=OK
```

```
[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]
```

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 0 \end{bmatrix}$$

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
```

$$\begin{bmatrix} 1.831...+1.734...\cdot i & .009...-1.490...\cdot i & ... \\ .448...-.725...\cdot i & 1.064...+.623\cdot i & ... \\ ^-.266...-2.083...\cdot i & 1.124...+1.790...\cdot i & ... \end{bmatrix}$$

LnReg MATH/Statistics/Regressions menu

LnReg *list1***,** *list2*[**,** [*list3*] [**,** *list4***,** *list5*]]

Calculates the logarithmic regression and updates all the system statistics variables.

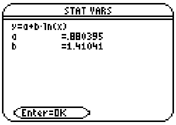
All the lists must have equal dimensions except for *list5*.

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:

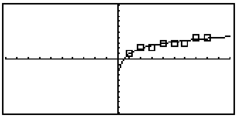
```
{1,2,3,4,5,6,7,8}→L1 ENTER
                          {1 2 3 …}
{1,2,2,3,3,3,4,4}→L2 ENTER
                          {1 2 2 …}
LnReg L1,L2 ENTER                Done
ShowStat ENTER
```



```
STAT VARS
y=a+b·ln(x)
a          =.880395
b          =1.41041
Enter=OK
```

```
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*.

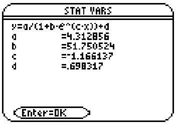
list1 represents xlist.
list2 represents ylist.
list3 represents frequency.
list4 represents category codes.
list5 represents category include list.

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 .

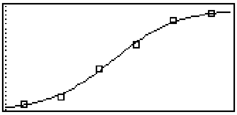
In function graphing mode:

```
{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]
```



```
STAT VARS
y=a/(1+b·e^(c·x))+d
a          =4.312856
b          =51.750524
c          =⁻1.166137
d          =.698317
```

```
[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*.

```
max([1,⁻3,7;⁻4,0,.3]) ENTER
                            [1 0 7]
```

Note: See also **fMax()** and **min()**.

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*.

```
median([.2,0;1,⁻.3;.4,⁻.5])
ENTER
                         [.4 ⁻.3]
```

Note: All entries in the list or matrix must simplify to numbers.

MedMed MATH/Statistics/Regressions menu

MedMed *list1*, *list2*[, [*list3*] [, *list4*, *list5*]]

Calculates the median-median line and updates all the system statistics variables.

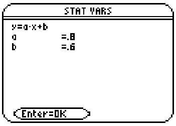
All the lists must have equal dimensions except for *list5*.

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:

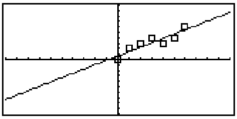
```
{0,1,2,3,4,5,6}→L1 [ENTER] {0 1 2 …}
{0,2,3,4,3,4,6}→L2 [ENTER] {0 2 3 …}
MedMed L1,L2 [ENTER]             Done
ShowStat [ENTER]
```



```
STAT VARS
y=a·x+b
a          =.8
b          =.6
Enter=OK
```

```
[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

$$\frac{^-(\cos(x-h)-\cos(x+h))}{2\cdot h}$$

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.

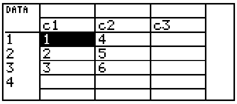
Must have at least one list.

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] $\begin{bmatrix}0 & 0 & 0\\0 & 0 & 0\end{bmatrix}$

NewPic CATALOG

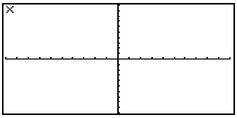
NewPic *matrix*, *picVar* [, *maxRow*][, *maxCol*]

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

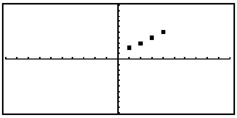
mark specifies the display type of the mark.
1 = ▫ (box)
2 = × (cross)
3 = + (plus)
4 = ▪ (square)
5 = • (dot)

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

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**.

```
NewProb ENTER                           Done
```

nInt() MATH/Calculus menu

nInt(*expression1, var, lower, upper***)** ⇒ *expression*

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*.

```
nInt(e^(⁻x^2),x,⁻1,1) ENTER
                                    1.493...
```

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.

A warning is displayed ("Questionable accuracy") when it seems that the goal has not been achieved.

```
nInt(cos(x),x,⁻π,π+1E⁻12) ENTER
                               ⁻1.041...E⁻12

∫(cos(x),x,⁻π,π+10^(⁻12)) ENTER
                             1
                  ⁻sin(─────────────)
                       1000000000000

ans(1)◆ ENTER                       ⁻1.E⁻12
```

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*]]

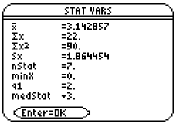
Calculates 1-variable statistics and updates all the system statistics variables.

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]
```



```
STAT VARS
x̄          =3.142857
Σx         =22.
Σx²        =90.
Sx         =1.864454
nStat      =7.
minX       =0.
q1         =2.
medStat    =3.
Enter=OK
```

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.

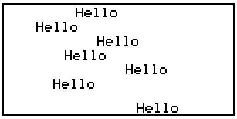
If Pretty Print = ON, *exprOrString* is “pretty printed.”

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:



```
        Hello
   Hello
            Hello
       Hello
                Hello
     Hello
                  Hello
```

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

See **ClrErr** program listing example.

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**.)

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 $\boxed{\text{ENTER}}$ Done

PlotsOff $\boxed{\text{ENTER}}$ 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 $\boxed{\text{ENTER}}$ Done

PlotsOn $\boxed{\text{ENTER}}$ Done

▸Polar MATH/Matrix/Vector ops menu

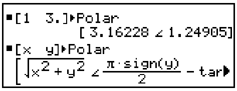
vector ▸**Polar**

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 $\boxed{\text{ENTER}}$
[x,y]▸Polar $\boxed{\text{ENTER}}$



■[1 3.]▸Polar
[3.16228 ∠ 1.24905]
■[x y]▸Polar
$\left[\sqrt{x^2+y^2}\ \angle\ \frac{\pi\cdot\text{sign}(y)}{2} - \text{tar}\blacktriangleright\right.$

---

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 $\boxed{\text{ENTER}}$ $e^{i\cdot(\frac{\pi}{2}-\tan^{-1}(3/4))}\cdot 5$

(4∠π/3)▸Polar $\boxed{\text{ENTER}}$ $e^{\frac{i\cdot\pi}{3}}\cdot 4$

In Degree angle mode:

3+4*i*▸Polar $\boxed{\text{ENTER}}$(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) $\boxed{\text{ENTER}}$
$a\cdot x^2+b\cdot x+c$

polyEval({1,2,3,4},2) $\boxed{\text{ENTER}}$ 26

polyEval({1,2,3,4},{2,⁻7})
$\boxed{\text{ENTER}}$ {26 ⁻262}

PopUp CATALOG

PopUp *itemList*, *var*

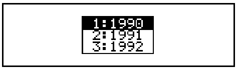
Displays a pop-up menu containing the character strings from *itemList*, waits for you to select an item, and stores the number of your selection in *var*.

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
```



```
1:1990
2:1991
3:1992
```

PowerReg MATH/Statistics/Regressions menu

PowerReg *list1*, *list2*[, [*list3*] [, *list4*, *list5*]]

Calculates the power regression and updates all the system statistics variables.

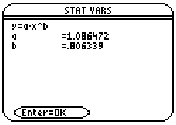
All the lists must have equal dimensions except for *list5*.

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:

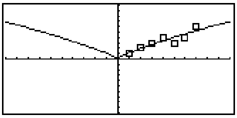
```
{1,2,3,4,5,6,7}→L1 ENTER
                        {1 2 3 ...}
{1,2,3,4,3,4,6}→L2 ENTER
                        {1 2 3 ...}
PowerReg L1,L2 ENTER          Done
ShowStat ENTER
```



```
STAT VARS
y=a·x^b
a        =1.086472
b        =.806339
Enter=OK
```

```
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 273.

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

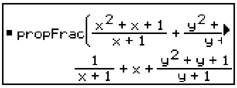
propFrac(*expression1*[**,** *var*]**)** ⇒ *expression*

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
```



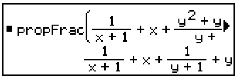
■ propFrac$\left(\frac{x^2+x+1}{x+1}+\frac{y^2+}{y+}\right.$▸

$\frac{1}{x+1}+x+\frac{y^2+y+1}{y+1}$

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))
```



■ propFrac$\left(\frac{1}{x+1}+x+\frac{y^2+y}{y+}\right.$▸

$\frac{1}{x+1}+x+\frac{1}{y+1}+y$

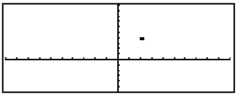
PtChg   CATALOG

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
```

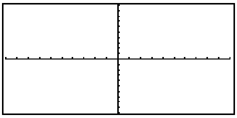


PtOff   CATALOG

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
```

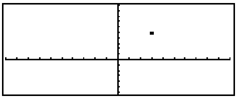


PtOn CATALOG

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
```

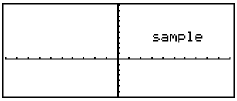
PtText CATALOG

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]
```



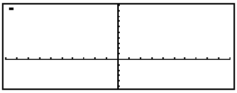
PxlChg CATALOG

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

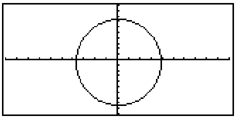
PxlCrcl *row*, *col*, *r* [, *drawMode*]

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

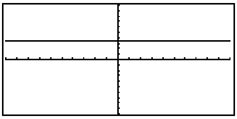
PxlHorz *row* [, *drawMode*]

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

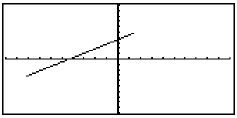
PxlLine *rowStart*, *colStart*, *rowEnd*, *colEnd* [, *drawMode*]

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]
```



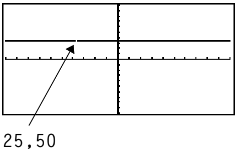
PxlOff CATALOG

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]
```



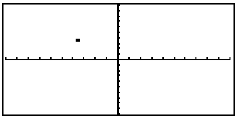
PxlOn CATALOG

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

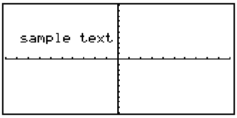
PxlText *string*, *row*, *col*

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

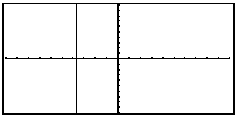
PxlVert *col* [**,** *drawMode*]

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.

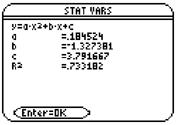
All the lists must have equal dimensions except for *list5*.

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:

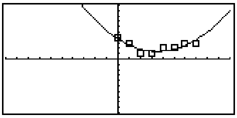
```
{0,1,2,3,4,5,6,7}→L1 [ENTER]
                            {1 2 3 ...}
{4,3,1,1,2,2,3,3}→L2 [ENTER]
                            {4 3 1 ...}
QuadReg L1,L2 [ENTER]              Done
ShowStat [ENTER]
```



```
STAT VARS
y=a·x²+b·x+c
a         =.184524
b         =⁻1.327381
c         =3.791667
R²        =.733182
Enter=OK
```

```
[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.

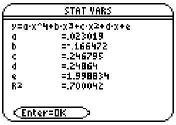
All the lists must have equal dimensions except for *list5*.

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:

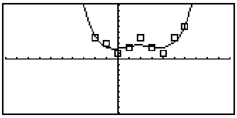
```
{⁻2,⁻1,0,1,2,3,4,5,6}→L1 [ENTER]
                         {⁻2 ⁻1 0 ...}
{4,3,1,2,4,2,1,4,6}→L2 [ENTER]
                            {4 3 1 ...}
QuartReg L1,L2 [ENTER]             Done
ShowStat [ENTER]
```



```
STAT VARS
y=a·x^4+b·x³+c·x²+d·x+e
a         =.023019
b         =⁻.166472
c         =.246795
d         =.24864
e         =1.998834
R²        =.700042
Enter=OK
```

```
[ENTER]
Regeq(x)→y1(x) [ENTER]             Done
NewPlot 1,1,L1,L2 [ENTER]          Done
```

[◆] [GRAPH]



R▸Pθ() MATH/Angle menu

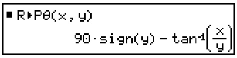
R▸Pθ (*xExpression*, *yExpression***)** ⇒ *expression*
R▸Pθ (*xList*, *yList***)** ⇒ *list*
R▸Pθ (*xMatrix*, *yMatrix***)** ⇒ *matrix*

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:

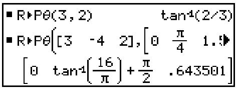
```
R▶Pθ(x,y) ENTER
```



■ R▸Pθ(x,y)
$90\cdot\text{sign}(y)-\tan^{-1}\left(\frac{x}{y}\right)$

In Radian angle mode:

```
R▶Pθ(3,2) ENTER
R▶Pθ([3,-4,2],[0,π/4,1.5]) ENTER
```



■ R▸Pθ(3,2) $\tan^{-1}(2/3)$
■ R▸Pθ$\left([3\ \ {}^-4\ \ 2],\left[0\ \ \frac{\pi}{4}\ \ 1.5\right.\right.$▸
$\left[0\ \ \tan^{-1}\left(\frac{16}{\pi}\right)+\frac{\pi}{2}\ \ .643501\right]$

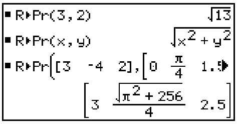
R▸Pr() MATH/Angle menu

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
```



■ R▸Pr(3,2) $\sqrt{13}$
■ R▸Pr(x,y) $\sqrt{x^2+y^2}$
■ R▸Pr$\left([3\ \ {}^-4\ \ 2],\left[0\ \ \frac{\pi}{4}\ \ 1.5\right.\right.$▸
$\left[3\ \ \frac{\sqrt{\pi^2+256}}{4}\ \ 2.5\right]$

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 $\begin{bmatrix} 8 & {}^-3 & 6 \\ {}^-2 & 3 & {}^-6 \\ 0 & 4 & {}^-6 \end{bmatrix}$

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
```

$^{-}2\cdot x^5+3\cdot x^4-6\cdot x^3+4\cdot x-6$

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 r$e^{i\theta}$ entry causes an error in Degree angle mode.

Note: You must use parentheses for an (r∠θ) polar entry.

In Radian angle mode:

```
4e^(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 $\boxed{\text{2nd}}$ [▸] for the ▸ operator. To type ∠, press $\boxed{\text{2nd}}$ [∠].

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 $\boxed{\blacklozenge}$ $\boxed{\text{ENTER}}$ 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}
```

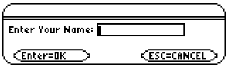
Request CATALOG

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
```

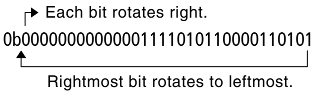
rotate() MATH/Base menu

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.
0b00000000000001111010110000110101
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) ENTER0h800001E3

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
```

$$\begin{bmatrix} 1.6 & 1.1 \\ 3.1 & 2.7 \end{bmatrix}$$

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
```

$$\begin{bmatrix} 3 & 4 \\ 0 & 2 \end{bmatrix}$$

```
rowAdd([a,b;c,d],1,2) ENTER
```

$$\begin{bmatrix} a & b \\ a+c & b+d \end{bmatrix}$$

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
```

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}$$

```
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
```

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}$$

```
rowSwap(Mat,1,3) ENTER
```

$$\begin{bmatrix} 5 & 6 \\ 3 & 4 \\ 1 & 2 \end{bmatrix}$$

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]                  √(2)

sec({1,2.3,4}) [ENTER]
```

$$\frac{1}{\cos(1)} \quad 1.000\ldots \quad \frac{1}{\cos(4)}$$

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]
```

$$0 \quad \frac{\pi}{3} \quad \cos^{-1}(1/5)$$

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]
```

$$\frac{1}{\cosh(1)} \quad .198\ldots \quad \frac{1}{\cosh(4)}$$

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]

$0\ (\frac{2\cdot\pi}{3})\cdot i\ 1.074\ldots\cdot i$

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]

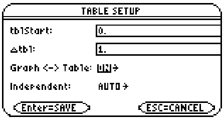


TABLE SETUP

tblStart: 0.

Δtbl: 1.

Graph <-> Table: ON →

Independent: AUTO →

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.

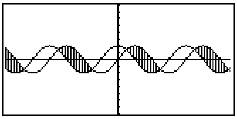
pattern specifies one of four shading patterns:
1 = vertical (default)
2 = horizontal
3 = negative-slope 45°
4 = positive-slope 45°

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.

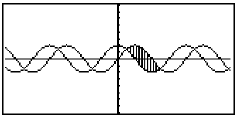
In the ZoomTrig viewing window:

```
Shade cos(x),sin(x) [ENTER]
```



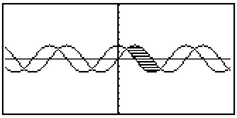
🖩 [HOME]
🖩 [◆] [CALC HOME]

```
ClrDraw [ENTER]                 Done
Shade cos(x),sin(x),0,5 [ENTER]
```



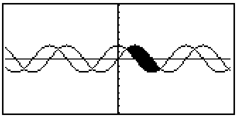
🖩 [HOME]
🖩 [◆] [CALC HOME]

```
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

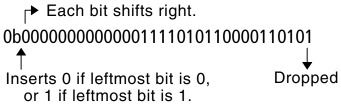
shift(*integer1*[**,***#ofShifts*]**)** ⇒ *integer*

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:


Each bit shifts right.
0b00000000000001111010110000110101
Inserts 0 if leftmost bit is 0, or 1 if leftmost bit is 1.
Dropped

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 "
```

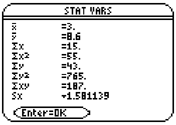
ShowStat CATALOG

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]
```



```
STAT VARS
x̄      =3.
ȳ      =8.6
Σx     =15.
Σx²    =55.
Σy     =43.
Σy²    =765.
Σxy    =187.
Sx     ≈1.581139
```

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]
```

$$\begin{bmatrix} ^-3 \\ 2 \end{bmatrix}$$

The solution is x=⁻3 and y=2.

Solve: ax + by = 1
cx + dy = 2

```
[a,b;c,d]→matx1 [ENTER]
```

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

```
simult(matx1,[1;2]) [ENTER]
```

$$\begin{bmatrix} \frac{^-(2\cdot b-d)}{a\cdot d-b\cdot c} \\ \frac{2\cdot a-c}{a\cdot d-b\cdot c} \end{bmatrix}$$

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: $x + 2y = 1$ $\quad$ $x + 2y = 2$
$3x + 4y = ^{-}1$ $\quad$ $3x + 4y = ^{-}3$

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 $\quad \frac{\sqrt{2}}{2}$

sin(45) ENTER $\quad \frac{\sqrt{2}}{2}$

sin({0,60,90}) ENTER $\quad \{0 \quad \frac{\sqrt{3}}{2} \quad 1\}$

In Radian angle mode:

sin(π/4) ENTER $\quad \frac{\sqrt{2}}{2}$

sin(45°) ENTER $\quad \frac{\sqrt{2}}{2}$

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 $\quad 90$

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) $\boxed{\text{ENTER}}$ 1.509...

sinh({0,1.2,3.}) $\boxed{\text{ENTER}}$
{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])
$\boxed{\text{ENTER}}$

$$\begin{bmatrix} 360.954 & 305.708 & 239.604 \\ 352.912 & 233.495 & 193.564 \\ 298.632 & 154.599 & 140.251 \end{bmatrix}$$

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) $\boxed{\text{ENTER}}$ 0

sinh^{-1}({0,2.1,3}) $\boxed{\text{ENTER}}$
{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])
$\boxed{\text{ENTER}}$

$$\begin{bmatrix} .041... & 2.155... & 1.158... \\ 1.463... & .926... & .112... \\ 2.750... & ^{-}1.528... & .572... \end{bmatrix}$$

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.

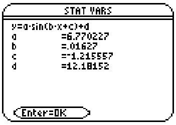
period specifies an estimated period. If omitted, the difference between values in *list1* should be equal and in sequential order. If you specify *period*, the differences between x values can be unequal.

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:

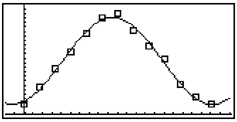
```
seq(x,x,1,361,30)→L1 [ENTER]
                                {1 31 61 …}
{5.5,8,11,13.5,16.5,19,19.5,17
,
14.5,12.5,8.5,6.5,5.5}→L2 [ENTER]
                               {5.5 8 11 …}
SinReg L1,L2 [ENTER]                   Done
ShowStat [ENTER]
```



STAT VARS

```
y=a·sin(b·x+c)+d
a        =6.770227
b        =.01627
c        =-1.215557
d        =12.18152
```

```
[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
```

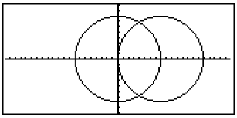
Optionally, you can specify an initial guess for a variable. Each *varOrGuess* must have the form:

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
```

$x=\frac{r}{2}$ and $y=\frac{\sqrt{3}\cdot r}{2}$

or $x=\frac{r}{2}$ 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
```

$x=\frac{r}{2}$ and $y=\frac{\sqrt{3}\cdot r}{2}$ and z=@1

or $x=\frac{r}{2}$ and $y=\frac{^-\sqrt{3}\cdot r}{2}$ 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
```

$x=\frac{e^z\cdot \sin(z)+1}{e^z+1}$ 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].

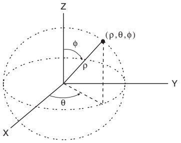
vector must be of dimension 3 and can be either a row or a column vector.

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 $[\sqrt{13}\ \ \angle\frac{\pi}{4}\ \ \angle\cos^{-1}(\frac{3\cdot\sqrt{13}}{13})]$



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

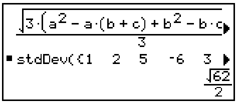
stdDev(*list*[, *freqlist*]**)** ⇒ *expression*

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
```



$\frac{\sqrt{3\cdot(a^2-a\cdot(b+c)+b^2-b\cdot c}}{3}$ ▸

■ stdDev({1 2 5 ⁻6 3 ▸ $\frac{\sqrt{62}}{2}$

```
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, ∆tbl, 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 277.

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 273.

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.

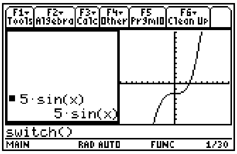
If *integer1* = 0, returns the active window number.

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.

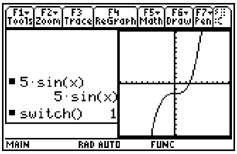
If *integer1* is omitted, switches windows and returns the previously active window number.

integer1 is ignored if the TI-89 Titanium/Voyage™ 200 is not displaying a split screen.



```
■ 5·sin(x)
      5·sin(x)
switch()
```

```
switch() [ENTER]
```



```
■ 5·sin(x)
      5·sin(x)
■ switch()    1
```

^{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

$$\begin{bmatrix}1 & 2 & 3\\4 & 5 & 6\\7 & 8 & 9\end{bmatrix}$$

$\texttt{mat1}^{T}$ ENTER

$$\begin{bmatrix}1 & 4 & 7\\2 & 5 & 8\\3 & 6 & 9\end{bmatrix}$$

`[a,b;c,d]→mat2` ENTER $\begin{bmatrix}a & b\\c & d\end{bmatrix}$

$\texttt{mat2}^{T}$ ENTER $\begin{bmatrix}a & c\\b & d\end{bmatrix}$

`[1+`*i*`,2+`*i*`;3+`*i*`,4+`*i*`]→mat3` ENTER

$$\begin{bmatrix}1+i & 2+i\\3+i & 4+i\end{bmatrix}$$

$\texttt{mat3}^{T}$ 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.

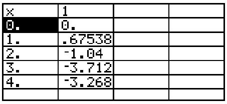
If the *var* parameter is omitted, the current graph-mode independent variable is assumed. Some valid variations of this instruction are:

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**.

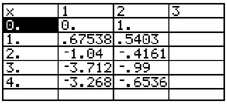
In function graphing mode.

`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]

$\{\infty \quad .518... - 1.570...\cdot \boldsymbol{i} \quad \frac{\ln(2)}{2} - \frac{\pi}{2}\cdot \boldsymbol{i}\}$

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]

$$\begin{bmatrix} ^{-}.099...+.164...\cdot \boldsymbol{i} & .267...-1.490...\cdot \boldsymbol{i} & ... \\ ^{-}.087...-.725...\cdot \boldsymbol{i} & .479...-.947...\cdot \boldsymbol{i} & ... \\ .511...-2.083...\cdot \boldsymbol{i} & ^{-}.878...+1.790...\cdot \boldsymbol{i} & ... \end{bmatrix}$$

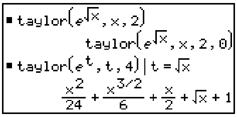
taylor() MATH/Calculus menu

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.

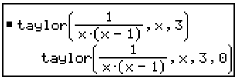
`taylor(e^(√(x)),x,2)` [ENTER]
`taylor(e^(t),t,4)|t=√(x)` [ENTER]



■ $\text{taylor}\left(e^{\sqrt{x}}, x, 2\right)$ $\quad \text{taylor}\left(e^{\sqrt{x}}, x, 2, 0\right)$

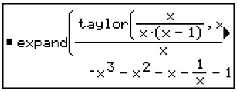
■ $\text{taylor}\left(e^{t}, t, 4\right) | t = \sqrt{x}$ $\quad \frac{x^2}{24} + \frac{x^{3/2}}{6} + \frac{x}{2} + \sqrt{x} + 1$

`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]
```

$$\cos(\alpha)\cdot\cos(\beta)+\sin(\alpha)\cdot\sin(\beta)$$

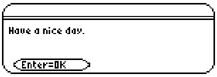
Text **CATALOG**

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
```



Have a nice day.

Then See **If,** page 196.

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

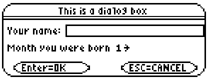
Title *titleString*, [*Lbl*]

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

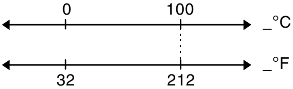
tmpCnv(*expression1*_°*tempUnit1*, _°*tempUnit2***)**
⇒ *expression* _°*tempUnit2*

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 **∆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*

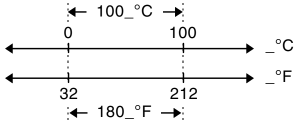
Converts a temperature range (the difference between two temperature values) specified by *expression1* from one unit to another. Valid temperature units are:

_°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*]]

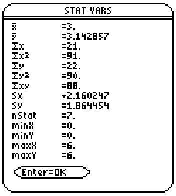
Calculates the **TwoVar** statistics and updates all the system statistics variables.

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]
```



```
        STAT VARS
x̄        =3.
ȳ        =3.142857
Σx       =21.
Σx²      =91.
Σy       =22.
Σy²      =90.
Σxy      =88.
Sx       =2.160247
Sy       =1.864454
nStat    =7.
minX     =0.
minY     =0.
maxX     =6.
maxY     =6.
<Enter=OK>
```

Unarchiv CATALOG

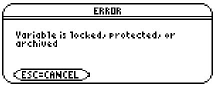
Unarchiv *var1* [, *var2*] [, *var3*] …

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
```



ERROR

Variable is locked, protected, or archived

ESC=CANCEL

```
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
```

$$\left[\frac{a}{\sqrt{a^2+b^2+c^2}}\ \frac{b}{\sqrt{a^2+b^2+c^2}}\ \frac{c}{\sqrt{a^2+b^2+c^2}}\right]$$

```
unitV([1,2,1]) ENTER
```

$$\left[\frac{\sqrt{6}}{6}\ \frac{\sqrt{6}}{3}\ \frac{\sqrt{6}}{6}\right]$$

```
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*

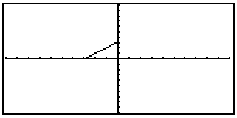
Returns *trueResult*, *falseResult*, or *unknownResult*, depending on whether *condition* is true, false, or unknown. Returns the input if there are too few arguments to specify the appropriate result.

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)
```

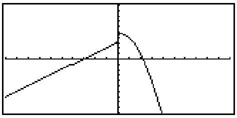
Use an undef *falseResult* to define an expression that graphs only on an interval.

```
ClrGraph [ENTER]
Graph when(x≥⁻π and
x<0,x+3,undef) [ENTER]
```



Omit only the *unknownResult* to define a two-piece expression.

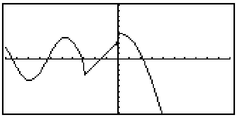
```
Graph when(x<0,x+3,5-x^2) [ENTER]
```



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 277.

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*

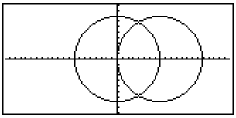
For example, x is valid and so is x=3.

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.

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*].



```
zeros({x^2+y^2-r^2,
(x-r)^2+y^2-r^2},{x,y}) [ENTER]
```

$$\begin{bmatrix}\frac{r}{2} & \frac{\sqrt{3}\cdot r}{2}\\ \frac{r}{2} & \frac{^-\sqrt{3}\cdot r}{2}\end{bmatrix}$$

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
```

$$[.041\ldots \quad 3.183\ldots]$$

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
```

$$[.001\ldots \quad 6.281\ldots]$$

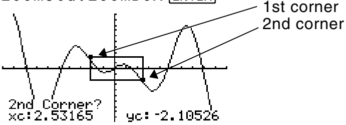
ZoomBox CATALOG

ZoomBox

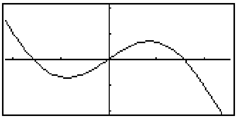
Displays the Graph screen, lets you draw a box that defines a new viewing window, and updates the window.

In function graphing mode:

```
1.25x*cos(x)→y1(x) ENTER      Done
ZoomStd:ZoomBox ENTER
```



2nd Corner?
xc:2.53165 yc:⁻2.10526



The display after defining ZoomBox by pressing ENTER the second time.

ZoomData CATALOG

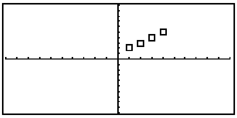
ZoomData

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.

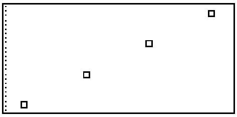
Note: Does not adjust ymin and ymax for histograms.

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]
```



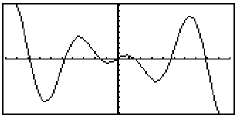
ZoomDec CATALOG

ZoomDec

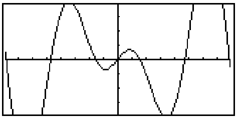
Adjusts the viewing window so that ∆x and ∆y = 0.1 and displays the Graph screen with the origin centered on the screen.

In function graphing mode:

```
1.25x∗cos(x)→y1(x) [ENTER]     Done
ZoomStd [ENTER]
```



```
🖩    [HOME]
⌨    [◆] [CALC HOME]
ZoomDec [ENTER]
```



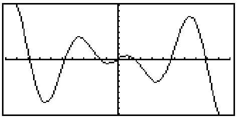
ZoomFit CATALOG

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:

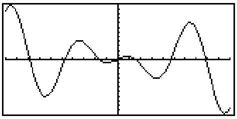
```
1.25x∗cos(x)→y1(x) [ENTER]      Done
ZoomStd [ENTER]
```



[HOME]

[◆] [CALC HOME]

```
ZoomFit [ENTER]
```



ZoomIn CATALOG

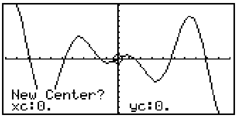
ZoomIn

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.

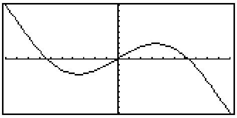
In function graphing mode:

```
1.25x∗cos(x)→y1(x) [ENTER]      Done
ZoomStd:ZoomIn [ENTER]
```



```
New Center?
xc:0.           yc:0.
```

[ENTER]



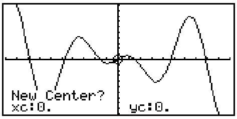
ZoomInt CATALOG

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.

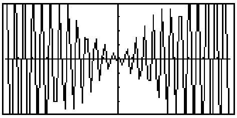
In function graphing mode:

```
1.25x∗cos(x)→y1(x) [ENTER]      Done
ZoomStd:ZoomInt [ENTER]
```



```
New Center?
xc:0.           yc:0.
```

[ENTER]



ZoomOut CATALOG

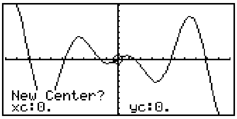
ZoomOut

Displays the Graph screen, lets you set a center point for a zoom out, and updates the viewing window.

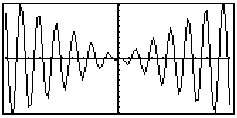
The magnitude of the zoom is dependent on the Zoom factors xFact and yFact. In 3D Graph mode, the magnitude is dependent on xFact, yFact, and zFact.

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

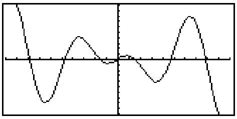
ZoomSqr

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.

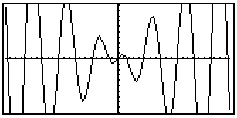
In 3D Graph mode, **ZoomSqr** lengthens the shortest two axes to be the same as the longest axis.

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]

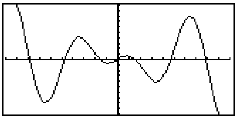
Sequence graphing:
nmin=1, nmax=10, plotStrt=1, plotStep=1,
x: [⁻10, 10, 1], y: [⁻10, 10, 1]

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.

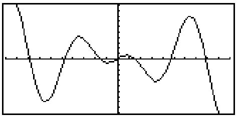
ZoomTrig CATALOG

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.

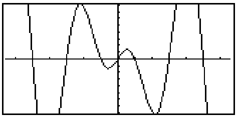
In function graphing mode:

```
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]
                                a+1   b
                             [   c   d+1 ]
```

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]

                                  21   2
                                [ 3   24 ]
```

− (subtract) [−] key

expression1 − *expression2* ⇒ *expression*

Returns *expression1* minus *expression2*.

```
6−2 [ENTER]                              4

                                       5•π
π−π/6 [ENTER]                          ───
                                        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]
                                 19   ⁻2
                               [ ⁻3   16 ]
```

∗ (multiply) [×] key

expression1 ∗ *expression2* ⇒ *expression*

Returns the product of *expression1* and *expression2*.

```
2∗3.45 [ENTER]                         6.9

x∗y∗x [ENTER]                          x²•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 $\{2 \cdot a \quad \frac{b}{2}\}$

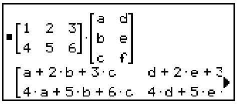
---

*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

$\{\frac{1}{b\cdot c} \quad \frac{1}{a\cdot c} \quad \frac{1}{a\cdot b}\}$

---

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

$[\frac{1}{b\cdot c} \quad \frac{1}{a\cdot c} \quad \frac{1}{a\cdot b}]$

^ (power) $\boxed{\wedge}$ **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 $\boxed{\text{ENTER}}$ 16

{a,2,c}^{1,b,3} $\boxed{\text{ENTER}}$ $\{a\ \ 2^{b}\ \ c^{3}\}$

expression **^** *list1* ⇒ *list*

Returns *expression* raised to the power of the elements in *list1*.

p^{a,2,$^{-}$3} $\boxed{\text{ENTER}}$ $\{p^{a}\ \ p^{2}\ \ \frac{1}{p^{3}}\}$

list1 **^** *expression* ⇒ *list*

Returns the elements in *list1* raised to the power of *expression*.

{1,2,3,4}^$^{-}$2 $\boxed{\text{ENTER}}$
{1 1/4 1/9 1/16}

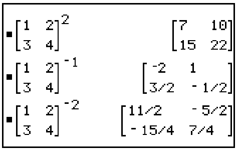
squareMatrix1 **^** *integer* ⇒ *matrix*

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 $\boxed{\text{ENTER}}$
[1,2;3,4]^$^{-}$1 $\boxed{\text{ENTER}}$
[1,2;3,4]^$^{-}$2 $\boxed{\text{ENTER}}$



$$\begin{bmatrix}1&2\\3&4\end{bmatrix}^{2} \quad \begin{bmatrix}7&10\\15&22\end{bmatrix}$$
$$\begin{bmatrix}1&2\\3&4\end{bmatrix}^{-1} \quad \begin{bmatrix}-2&1\\3/2&-1/2\end{bmatrix}$$
$$\begin{bmatrix}1&2\\3&4\end{bmatrix}^{-2} \quad \begin{bmatrix}11/2&-5/2\\-15/4&7/4\end{bmatrix}$$

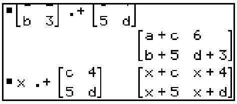
.+ (dot add) $\boxed{.}\ \boxed{+}$ **keys**

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] $\boxed{\text{ENTER}}$
x.+[c,4;5,d] $\boxed{\text{ENTER}}$



$$\begin{bmatrix}a+c&6\\b+5&d+3\end{bmatrix}$$
$$x\ .+\begin{bmatrix}c&4\\5&d\end{bmatrix} \quad \begin{bmatrix}x+c&x+4\\x+5&x+d\end{bmatrix}$$

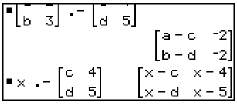
.− (dot subt.) $\boxed{.}\ \boxed{-}$ **keys**

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] $\boxed{\text{ENTER}}$
x.−[c,4;d,5] $\boxed{\text{ENTER}}$



$$\begin{bmatrix}a-c&-2\\b-d&-2\end{bmatrix}$$
$$x\ .-\begin{bmatrix}c&4\\d&5\end{bmatrix} \quad \begin{bmatrix}x-c&x-4\\x-d&x-5\end{bmatrix}$$

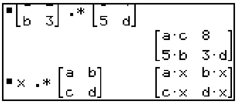
.∗ (dot mult.) [.] [×] **keys**

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]
```



$\blacksquare\begin{bmatrix} & \\ b & 3\end{bmatrix} .* \begin{bmatrix} & \\ 5 & d\end{bmatrix} \qquad \begin{bmatrix} a\cdot c & 8 \\ 5\cdot b & 3\cdot d\end{bmatrix}$

$\blacksquare x .* \begin{bmatrix} a & b \\ c & d\end{bmatrix} \qquad \begin{bmatrix} a\cdot x & b\cdot x \\ c\cdot x & d\cdot x\end{bmatrix}$

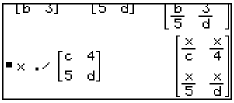
./ (dot divide) [.] [÷] **keys**

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]
```



$\begin{bmatrix} b & 3\end{bmatrix} \quad \begin{bmatrix} 5 & d\end{bmatrix} \qquad \begin{bmatrix} \frac{b}{5} & \frac{3}{d}\end{bmatrix}$

$\blacksquare x ./ \begin{bmatrix} c & 4 \\ 5 & d\end{bmatrix} \qquad \begin{bmatrix} \frac{x}{c} & \frac{x}{4} \\ \frac{x}{5} & \frac{x}{d}\end{bmatrix}$

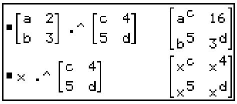
.^ (dot power) [.] [^] **keys**

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]
```



$\blacksquare\begin{bmatrix} a & 2 \\ b & 3\end{bmatrix} .\hat{} \begin{bmatrix} c & 4 \\ 5 & d\end{bmatrix} \qquad \begin{bmatrix} a^c & 16 \\ b^5 & 3^d\end{bmatrix}$

$\blacksquare x .\hat{} \begin{bmatrix} c & 4 \\ 5 & d\end{bmatrix} \qquad \begin{bmatrix} x^c & x^4 \\ x^5 & x^d\end{bmatrix}$

⁻ (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

⁻{⁻1,0.4,1.2ᴇ19} [ENTER]
                          {1  ⁻.4  ⁻1.2ᴇ19}

⁻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) $\boxed{=}$ **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*.

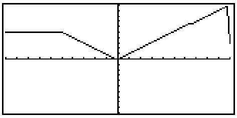
Anything else returns a simplified form of the equation.

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)` $\boxed{\text{ENTER}}$



≠ $\boxed{◆}$ $\boxed{=}$ **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.

< $\boxed{\text{2nd}}$ [<] **key**

expression1 < *expression2* ⇒ *Boolean expression*
list1 < *list2* ⇒ *Boolean list*
matrix1 < *matrix2* ⇒ *Boolean matrix*

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.

See "=" (equal) example.

≤ [◆] [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) 🖩 [◆] [÷] **key** ⌨ [2nd] W **key**

*expression1***!** ⇒ *expression*
*list1***!** ⇒ *list*
*matrix1***!** ⇒ *matrix*

```
5! [ENTER]                              120
{5,4,3}! [ENTER]              {120   24   6}
[1,2;3,4]! [ENTER]
```

$$\begin{bmatrix} 1 & 2 \\ 6 & 24 \end{bmatrix}$$

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.

& (append) ◆ × key 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.

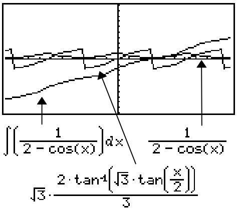
`∫(x^2,x) [ENTER]` $\frac{x^3}{3}$

However, *lower* is added as a constant of integration if only *upper* is omitted.

`∫(a*x^2,x,c) [ENTER]` $\frac{a \cdot x^3}{3} + c$

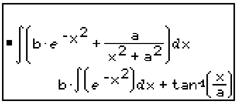
Equally valid anti-derivatives might differ by a numeric constant. Such a constant might be disguised—particularly when an anti-derivative contains logarithms or inverse trigonometric functions. Moreover, piecewise constant expressions are sometimes added to make an anti-derivative valid over a larger 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]
```



■ $\int\left(b\cdot e^{-x^2} + \frac{a}{x^2+a^2}\right)dx$
$b\cdot\int\left(e^{-x^2}\right)dx + \tan^{-1}\left(\frac{x}{a}\right)$

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.

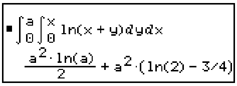
For the AUTO setting of the Exact/Approx mode, numerical integration is used where applicable when an anti-derivative or a limit cannot be determined.

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.

`∫(∫(ln(x+y),y,0,x),x,0,a) [ENTER]`



■ $\int_0^a\int_0^x \ln(x+y)\,dy\,dx$
$\frac{a^2\cdot\ln(a)}{2} + a^2\cdot(\ln(2) - 3/4)$

Note: See also **nInt()**.

√() (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 $\frac{(\sqrt{3}+1)\cdot\sqrt{2}}{4}$ $^-$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…}

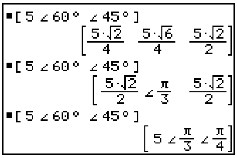
∠ (angle) 2nd [∠] key

[*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:



■[5 ∠60° ∠45°]
$\left[\frac{5\cdot\sqrt{2}}{4} \quad \frac{5\cdot\sqrt{6}}{4} \quad \frac{5\cdot\sqrt{2}}{2}\right]$
■[5 ∠60° ∠45°]
$\left[\frac{5\cdot\sqrt{2}}{2} \quad \angle\frac{\pi}{3} \quad \frac{5\cdot\sqrt{2}}{2}\right]$
■[5 ∠60° ∠45°]
$\left[5 \quad \angle\frac{\pi}{3} \quad \angle\frac{\pi}{4}\right]$

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+3***i***–(10∠π/4) ENTER
5–5·$\sqrt{2}$+(3–5·$\sqrt{2}$)·***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*

```
3_m▸_ft [ENTER]            9.842…•_ft
```

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.

10^() **CATALOG**

10^ (*expression1***)** ⇒ *expression*
10^ (*list1***)** ⇒ *list*

```
10^(1.5) [ENTER]           31.622…

10^{0,⁻2,2,a} [ENTER]
```

$$\{1 \;\; \frac{1}{100} \;\; 100 \;\; 10^{a}\}$$

Returns 10 raised to the power of the argument.

For a list, returns 10 raised to the power of the elements in *list1*.

10^(*squareMatrix1***)** ⇒ *squareMatrix*

```
10^([1,5,3;4,2,1;6,-2,1]) [ENTER]
```

$$\begin{bmatrix} 1.143\ldots E7 & 8.171\ldots E6 & 6.675\ldots E6 \\ 9.956\ldots E6 & 7.115\ldots E6 & 5.813\ldots E6 \\ 7.652\ldots E6 & 5.469\ldots E6 & 4.468\ldots E6 \end{bmatrix}$$

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.

$\mathbf{x}^{-1}$ **CATALOG** (^-1)

expression1 $\mathbf{x}^{-1}$ ⇒ *expression*
list1 $\mathbf{x}^{-1}$ ⇒ *list*

```
3.1^⁻1 [ENTER]             .322581

{a,4,⁻.1,x-2}^⁻1 [ENTER]
```

$$\{\frac{1}{a} \;\; \frac{1}{4} \;\; ⁻10. \;\; \frac{1}{x-2}\}$$

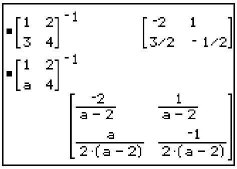
Returns the reciprocal of the argument.

For a list, returns the reciprocals of the elements in *list1*.

---

squareMatrix1 $\mathbf{x}^{-1}$ ⇒ *squareMatrix*

```
[1,2;3,4]^⁻1 [ENTER]
[1,2;a,4]^⁻1 [ENTER]
```



$$\blacksquare \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}^{-1} \quad \begin{bmatrix} -2 & 1 \\ 3/2 & -1/2 \end{bmatrix}$$

$$\blacksquare \begin{bmatrix} 1 & 2 \\ a & 4 \end{bmatrix}^{-1} \quad \begin{bmatrix} \frac{-2}{a-2} & \frac{1}{a-2} \\ \frac{a}{2\cdot(a-2)} & \frac{-1}{2\cdot(a-2)} \end{bmatrix}$$

Returns the inverse of *squareMatrix1*.

squareMatrix1 must be a non-singular square matrix.

| ("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]` $\sqrt{\frac{1}{x}}\cdot\sqrt{x}$

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]` $\frac{\pi}{4}$

```
2cos(x)→Y1(x) [ENTER]               Done
{1,2,3,4}→Lst5 [ENTER]         {1 2 3 4}
```

`[1,2,3;4,5,6]→MatG [ENTER]` $\begin{bmatrix}1 & 2 & 3\\4 & 5 & 6\end{bmatrix}$

```
"Hello"→str1 [ENTER]             "Hello"
```

© (comment) **Program Editor/Control menu** or

🖩 [◆] [)] **key**

⌨ [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

🖩 [0] [alpha] [B] **keys** ⌨ [0] B **keys**

🖩 [0] [alpha] [H] **keys** ⌨ [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: General Information

Texas Instruments Support and Service

For general information

Home Page:	education.ti.com
KnowledgeBase and e-mail inquires:	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.

Texas Instruments (TI) Warranty Information

Customers in the U.S. and Canada Only

One-Year Limited Warranty for Commercial Electronic Product

This Texas Instruments (“TI”) electronic product warranty extends only to the original purchaser and user of the product.

Warranty Duration. This TI electronic product is warranted to the original purchaser for a period of one (1) year from the original purchase date.

Warranty Coverage. This TI electronic product is warranted against defective materials and construction. **THIS WARRANTY IS VOID IF THE PRODUCT HAS BEEN DAMAGED BY ACCIDENT OR UNREASONABLE USE, NEGLECT, IMPROPER SERVICE, OR OTHER CAUSES NOT ARISING OUT OF DEFECTS IN MATERIALS OR CONSTRUCTION.**

Warranty Disclaimers. ANY IMPLIED WARRANTIES ARISING OUT OF THIS SALE, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, ARE LIMITED IN DURATION TO THE ABOVE ONE-YEAR PERIOD. TEXAS INSTRUMENTS SHALL NOT BE LIABLE FOR LOSS OF USE OF THE PRODUCT OR OTHER INCIDENTAL OR CONSEQUENTIAL COSTS, EXPENSES, OR DAMAGES INCURRED BY THE CONSUMER OR ANY OTHER USER.

Some states/provinces do not allow the exclusion or limitation of implied warranties or consequential damages, so the above limitations or exclusions may not apply to you.

Legal Remedies. This warranty gives you specific legal rights, and you may also have other rights that vary from state to state or province to province.

Warranty Performance. During the above one (1) year warranty period, your defective product will be either repaired or replaced with a reconditioned model of an equivalent quality (at TI’s option) when the product is returned, postage prepaid, to Texas Instruments Service Facility. The warranty of the repaired or replacement unit will continue for the warranty of the original unit or six (6) months, whichever is longer. Other than the postage requirement, no charge will be made for such repair and/or replacement. TI strongly recommends that you insure the product for value prior to mailing.

Software. Software is licensed, not sold. TI and its licensors do not warrant that the software will be free from errors or meet your specific requirements. **All software is provided “AS IS.”**

Copyright. The software and any documentation supplied with this product are protected by copyright.

Australia & New Zealand Customers only

One-Year Limited Warranty for Commercial Electronic Product

This Texas Instruments electronic product warranty extends only to the original purchaser and user of the product.

Warranty Duration. This Texas Instruments electronic product is warranted to the original purchaser for a period of one (1) year from the original purchase date.

Warranty Coverage. This Texas Instruments electronic product is warranted against defective materials and construction. This warranty is void if the product has been damaged by accident or unreasonable use, neglect, improper service, or other causes not arising out of defects in materials or construction.

Warranty Disclaimers. Any implied warranties arising out of this sale, including but not limited to the implied warranties of merchantability and fitness for a particular purpose, are limited in duration to the above one-year period. Texas Instruments shall not be liable for loss of use of the product or other incidental or consequential costs, expenses, or damages incurred by the consumer or any other user.

Except as expressly provided in the One-Year Limited Warranty for this product, Texas Instruments does not promise that facilities for the repair of this product or parts for the repair of this product will be available.

Some jurisdictions do not allow the exclusion or limitation of implied warranties or consequential damages, so the above limitations or exclusions may not apply to you.

Legal Remedies. This warranty gives you specific legal rights, and you may also have other rights that vary from jurisdiction to jurisdiction.

Warranty Performance. During the above one (1) year warranty period, your defective product will be either repaired or replaced with a new or reconditioned model of an equivalent quality (at TI's option) when the product is returned to the original point of purchase. The repaired or replacement unit will continue for the warranty of the original unit or six (6) months, whichever is longer. Other than your cost to return the product, no charge will be made for such repair and/or replacement. TI strongly recommends that you insure the product for value if you mail it.

Software. Software is licensed, not sold. TI and its licensors do not warrant that the software will be free from errors or meet your specific requirements. **All software is provided "AS IS."**

Copyright. The software and any documentation supplied with this product are protected by copyright.

All Other Customers

For information about the length and terms of the warranty, refer to your package and/or to the warranty statement enclosed with this product, or contact your local Texas Instruments retailer/distributor.

Index