TI-Basic Programming Guide for the TI-82 Advanced Edition Python

Learn more about TI Technology through the online help at education.ti.com/eguide.

Important Information

Except as otherwise expressly stated in the License that accompanies a program, 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 amount set forth in the license for the program. Moreover, Texas Instruments shall not be liable for any claim of any kind whatsoever against the use of these materials by any other party.

.

Contents

Introduction to CE TI-Basic Programming on your TI-82 Advanced Edition Python

You can use TI-Basic to create a program on your graphing calculator. You can create a program that will calculate a desired output or control an experience, such as a game.

What Is a Program?

A program is a set of one or more command lines, each containing one or more instructions. When you execute a program, the TI-82 Advanced Edition Python performs each instruction on each command line in the same order in which you entered them. The number and size of programs that the calculator can store is limited only by available memory.

To create a program, simply enter command lines using the Program Editor. The program will run from the Home Screen. Use this guide to learn how to create, edit, and delete programs.

Tip: Use Catalog Help by pressing [+] on most commands to help you fill in the correct arguments for the commands before you paste them into the Program Editor.

As you progress in programming, a TI-Basic Program Editor is also available in TI Connect™ CE software. You can use the Program Editor workspace in TI Connect™ CE to create programs, to send programs to a connected calculator via USB, to test your programs, and to save programs to your computer. The Program Editor workspace in TI Connect™ CE allows copy, cut, paste, and undo commands. Please use the TI-83 Premium CE catalog in the Program Editor in TI Connect CE v5.6.3 or higher.

Please select only the TI-82 Advanced Python Edition commands and functions as described in the TI-82 Advanced Edition Python calculator catalog or Reference Guide. Running unsupported commands and functions will result in an Invalid error on the TI-82 Advanced Edition Python.

Note: The TI-Basic program editor on the calculator contains editing features in the [alpha] [f5] menu such as undo, copy, cut and paste.

Getting Started Activity:

Programming the Formula to find the Volume of a Cylinder given Radius and Height

Given the Radius and Height of a cylinder, you can compute the Volume using this formula. This activity allows you to write a program to prompt for the values of the Radius and Height of a cylinder so that you can then compute the Volume.

The formula for the volume of a cylinder is

$V = \pi R^2 H$ cubic units

Where

V = Volume

R = Radius of the base

H = Height of the cylinder

This program could be useful for a variety of activities such as:

- Providing a table with many values of Radius and Height and having students fill out the Volume column
- Running a program to fill in the values for Volume in the table

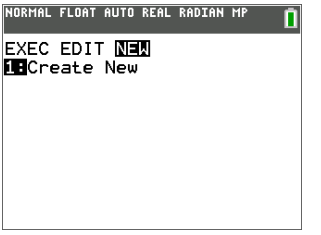
Some questions to investigate:

- (If formula is unknown to the student), what pattern do you see in the Volume numbers to make a good guess at the formula?
- What is the largest Volume found?
- How much does the Volume increase if the Height increases by one unit?
- How much does the Volume increase if the Radius increases by one unit?

Running a program repeatedly as a tool allows quick analysis for higher-level thinking problems.

Creating a NEW Program

1. Press prgm ▶ ▶ to display the **PRGM NEW** menu.

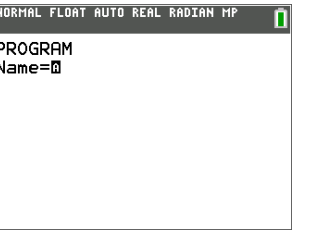


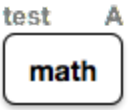
Naming the Program

1. Press entrer to select **1:Create New**.

   The **Name=** prompt is displayed, and [2nde] [verr A] (alpha-lock) is on.

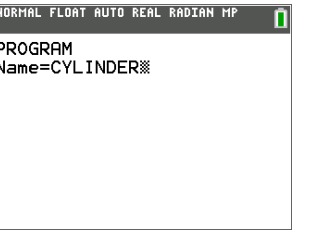
   Tip: The alpha characters are upper right above keys on the keypad and are pasted when [alpha] or [2nde] [verr A] is pressed before pressing the primary key.





2. Press **C Y L I N D E R**, and then press entrer to name the program **CYLINDER**.

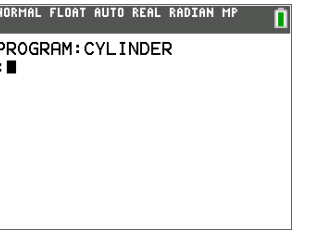
   Tip: Program names can have a maximum of eight characters. First character must be a letter. Notice the checkerboard cursor on the screen when the maximum is reached.



3. Press entrer and you are now in the program editor.

   The colon (:) in the first column of the second line indicates the beginning of a command line.

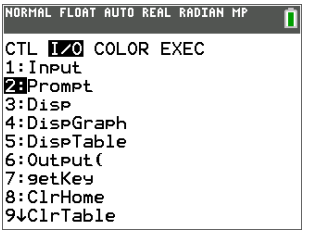
   Note: On the calculator, the command lines are not numbered as when using the TI Connect™ CE Program Editor.



Entering Commands

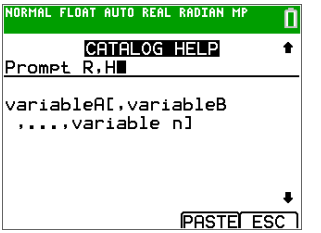
Whoever uses your program will have to input the Radius and Height values. You will use the **Prompt** command.

1. Press prgm ▶ to access the I/O (Input/Output) command menu.
2. Press ▼ to highlight the **Prompt** command.

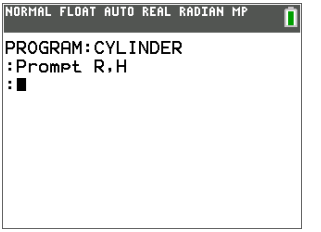


   Note: For this example, you will use the Catalog Help feature to illustrate this built-in argument syntax help in the calculator. If you already know the arguments for a command, you can select a menu item and paste them to the Program Editor without using Catalog Help.

3. The **Prompt** menu item number is highlighted so press +. Use the Catalog Help syntax editor (if needed). The syntax for the arguments of Prompt is shown below the editing line as variables separated by commas. Anything within a square bracket [ ] is an optional argument, so Prompt needs at least one variable name.

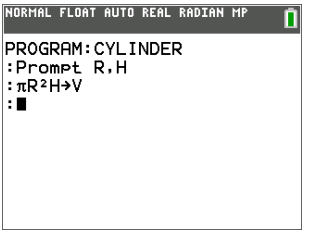


4. Press alpha **R** , alpha **H** to enter the variable names for Radius and Height.
5. Press [PASTE] (trace) to paste the command with the arguments back to the Program Editor. Press [ESC] (graphe) to return to the last cursor location without pasting.
6. Back on the Program Editor, press entrer to move the cursor to the next command line.



Store the formula for the volume of a cylinder:

7. To enter the expression [π] **R2 H** and store value to the variable V, press [2nde] [π] [alpha] **R** [x²] [alpha] **H** [sto→] [alpha] **V** [entrer].



Displaying the Calculated Volume.

Create a command line to display the calculated volume:

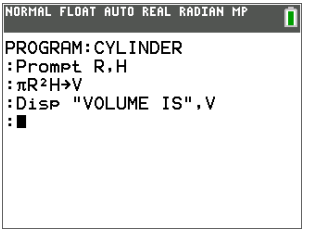
1. Press [prgm] [▸] **3** to select 3:Disp from the **PRGM I/O** menu.

   Disp is pasted to the command line.

   Tip: Remember you can press [+] on most commands to use the Catalog Help syntax editor to see the correct arguments for commands.

2. Press [2nde] [verr A] ["] **VOLUME IS** ["] [alpha] [,] [alpha] **V** [entrer]

   This will display the text **VOLUME IS** on one line and the calculated value of **V** on the next line of the Home Screen when you run the program.

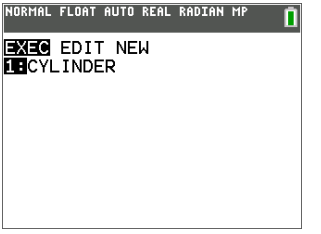


Running a Program

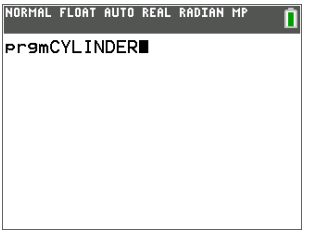
Your program is complete! Now run the program from the Home Screen.

1. Press [2nde] [quitter] to display the **Home Screen**.
2. Press [prgm] to display the **PRGM EXEC** menu.

   The items on this menu are the names of stored programs.



3. Press [entrer] to paste **prgm CYLINDER** to the current cursor location. (If **CYLINDER** is not item 1 on your **PRGM EXEC** menu, move the cursor to **CYLINDER** before you press [entrer].)



Finding the Volume

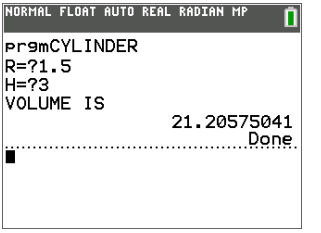
To find the volume of the cylinder with Radius 1.5 cm and Height 3 cm, complete the following steps.

1. Press [entrer] to execute (run) the program.
2. When prompted for **R**, enter **1.5** and press [entrer]
3. When prompted for **H**, enter **3** and press [entrer].

   The text **VOLUME IS**, the value of **V**, and **Done** are displayed.

   The volume of the cylinder is displayed to 8 decimal places as 21. 20575041 cubic cm.

4. At this point, to rerun the program, press [entrer] and repeat for different values of **R** and **H**.



Creating and Deleting Programs

This section describes how to create programs, and how to delete programs.

Operating Systems Versions and Programming

- Programs can run in Classic or MathPrint™ mode.
- Shortcut menus are available wherever the MATH menu can be accessed.
- MathPrint™ templates are not available for programs. All input and output is in Classic format.
- You can use fractions in programs, but you should test the program to make sure that you get the desired results.
- The spacing of the display may be slightly different in MathPrint™ mode than in Classic mode. If you prefer the spacing in Classic mode, set the mode using a command in your program. Screen shots for the examples in this chapter were taken in MathPrint™ mode.

Note: Press [+] when a command is highlighted in a menu to use the syntax help for your programming.

Creating a New Program

To create a new program, follow these steps.

1. Press [prgm] [◀] to display the **PRGM NEW** menu.

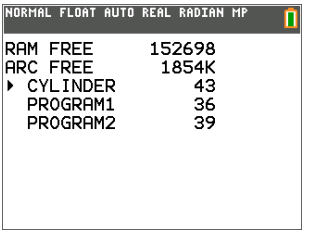
2. Press [entrer] to select **1:Create New**. The **Name=** prompt is displayed, and alpha-lock is on.
3. Press a letter from A to Z or θ to enter the first character of the new program name.

   Note: A program name can be one to eight characters long. The first character must be a letter from A to Z or θ. The second through eighth characters can be letters, numbers, or θ.
4. Enter zero to seven letters, numbers, or θ to complete the new program name.
5. Press [entrer]. The program editor is displayed.
6. Enter one or more program commands.
7. Press [2nde] [quitter] to leave the program editor and return to the home screen.

Managing Memory and Deleting a Program

To check whether adequate memory is available for a program you want to enter:

1. Press [2nde] [mém] to display the **MEMORY** menu.
2. Select **2:Mem Management/Delete** to display the **MEMORY MANAGEMENT/DELETE** menu.
3. Select **7:Prgm** to display the **PRGM** editor.



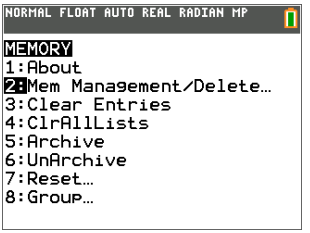
The TI-82 Advanced Edition Python calculator expresses memory quantities in bytes.

Increase Available Memory

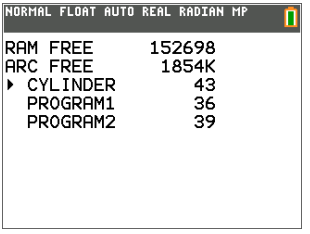
You can increase available memory in one of two ways. You can delete one or more programs or you can archive some programs.

To increase available memory by deleting a specific program:

1. Press [2nde] [mém] and then select **2:Mem Management/Delete** from the **MEMORY** menu.



2. Select **7:Prgm** to display the program files.



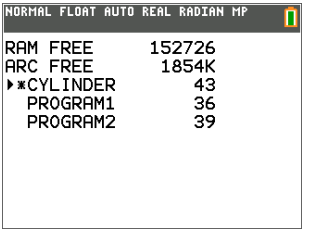
3. Press ▲ and ▼ alpha o move the selection cursor (▶) next to the program you want to delete, and then press suppr. The program is deleted from memory.

   Note: You will receive a message asking you to confirm this delete action. Select **2:yes** to continue.

   To leave the **PRGM** editor screen without deleting anything, press 2nde [quitter], which displays the home screen.

To increase available memory by archiving a program:

1. Press 2nde [mém] and then select **2:Mem Management/Delete** from the **MEMORY** menu.
2. Select **2:Mem Management/Delete** to display the **MEMORY MANAGEMENT/DELETE** menu.
3. Select **7:Prgm...** to display the program files.



4. Press entrer to archive the program. An asterisk will appear to the left of the program to indicate it is an archived program.

   To unarchive a program in this screen, put the cursor next to the archived program and press entrer. The asterisk will disappear.

   Note: Archive programs cannot be edited or executed. In order to edit or execute an archived program, you must first unarchive it.

Entering Command Lines and Executing Programs

This section describes how to enter a command line and how to execute programs.

Entering a Program Command Line

You can enter on a command line any command, instruction, or expression that you could execute from the home screen. In the program editor, each new command line begins with a colon. To enter more than one instruction or expression on a single command line, separate each with a colon.

Note: A command line can be longer than the screen is wide.

While in the program editor, you can display and select from menus. You can return to the program editor from a menu in either of two ways.

- Select a menu item, which pastes the item to the current command line.

  — or —

- Press annul.

When you complete a command line, press entrer. The cursor moves to the next command line.

Programs can access variables, lists, matrices, and strings saved in memory. If a program stores a new value to a variable, list, matrix, or string, the program changes the value in memory during execution.

You can call another program as a subroutine.

Executing a Program

To execute a program, begin on a blank line on the home screen and follow these steps.

1. Press [prgm] to display the **PRGM EXEC** menu.
2. Select a program name from the **PRGM EXEC** menu. **prgm***name* is pasted to the home screen (for example, **prgmCYLINDER**).
3. Press [entrer] to execute the program. While the program is executing, the busy indicator is on.
   - Last Answer (Ans) is updated during program execution.
   - Last Entry is not updated as each command is executed.
   - The TI-82 Advanced Edition Python calculator checks for errors during program execution.
   - There is no syntax checking as you enter a program in the program editor.

Note: CE OS 5.3 and later

- Programs can be executed from RAM or Archive. Programs cannot be edited if in Archive.
- Editing MENU ([alpha] [f5]) in the Program Editor
  - **1:Execute Program**
    - The program being edited will execute directly from the program editor.
  - **7: Insert Comment Above ([alpha] [f5] 7)**
    - **Insert Comment Above** pastes the quote " token at the start of a new command line above the current cursor location. When this command line is executed, a String variable of the comment text is created and is stored in the Ans variable. When planning out your program and use of variables, if comments are included using the quote " token, plan that the comment string will be stored to Ans upon execution.

Breaking a Program

To stop program execution, press [on]. The **ERR:BREAK** menu is displayed.

- To return to the home screen, select **1:Quit**.
- To go where the interruption occurred, select **2:Goto**.

Editing Programs

In this section you will follow steps to edit a program. This section describes how to insert and delete command line.

Editing a Program

To edit a stored program, follow these steps.

1. Press prgm ▶ to display the **PRGM EDIT** menu.
2. Select a program name from the **PRGM EDIT** menu. Up to the first nine lines of the program are displayed.

   Note: The program editor does not display a ↓ to indicate that a program continues beyond the screen.
3. Edit the program command lines.
   - Move the cursor to the appropriate location, and then delete, overwrite, or insert.
   - Press annul to clear all program commands on the command line (the leading colon remains), and then enter a new program command.

Note: To move the cursor to the beginning of a command line, press 2nde ◀; to move to the end, press 2nde ▶. To scroll the cursor down seven command lines, press alpha ▼. To scroll the cursor up seven command lines, press alpha ▲.

Inserting and Deleting Command Lines

To insert a new command line anywhere in the program, place the cursor where you want the new line, press 2nde [insérer], and then press entrer. A colon indicates a new line.

To delete a command line, place the cursor on the line, press annul to clear all instructions and expressions on the line, and then press suppr to delete the command line, including the colon.

Editing Feature

Editing features have been added to the Program Editor. These include undo, insert, and cut/copy/paste. Press prgm to EDIT your program. Pressing alpha [f5] opens the new editing MENU.

Tip: MENU Item 7: Insert Comment Above (alpha [f5] 7)

Insert Comment Above pastes the quote " token at the start of a new command line above the current cursor location. When this command line is executed, a String variable of the comment text is created and is stored in the Ans variable. When planning out your program and use of variables, if comments are included using the quote " token, plan that the comment string will be stored to Ans upon execution.

Copying and Renaming Programs

This section describes how to copy and rename a program, and how to scroll the menus.

Copying and Renaming a Program

To copy all command lines from one program into a new program, follow steps 1 through 4 for Creating a New Program, and then follow these steps.

1. Press [2nde] [rappel]. **rappel** is displayed on the bottom line of the program editor in the new program.
2. Press [prgm] to display the **PRGM EXEC** menu.
3. Select a name from the menu. **prgm***name* is pasted to the bottom line of the program editor.
4. Press [entrer]. All command lines from the selected program are copied into the new program.

Copying programs has at least two convenient applications.

- You can create a template for groups of instructions that you use frequently.
- You can rename a program by copying its contents into a new program.

Note: You also can copy all the command lines from one existing program to another existing program using **rappel** (**RCL**).

Scrolling the PRGM EXEC and PRGM EDIT Menus

The TI-82 Advanced Edition Python calculator sort **PRGM EXEC** and **PRGM EDIT** menu items automatically into alphanumerical order. Each menu only labels the first 10 items using 1 through 9, then 0.

To jump to the first program name that begins with a particular alpha character or θ, press [alpha] [letter from A to Z or θ].

Note: From the top of either the **PRGM EXEC** or **PRGM EDIT** menu, press [▲] to move to the bottom. From the bottom, press [▼] to move to the top. To scroll the cursor down the menu seven items, press [alpha] [▼]. To scroll the cursor up the menu seven items, press [alpha] [▲].

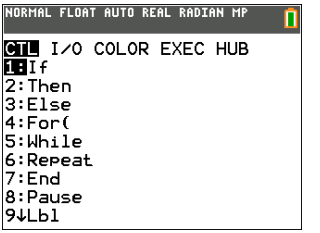
PRGM CTL (Control) Instructions

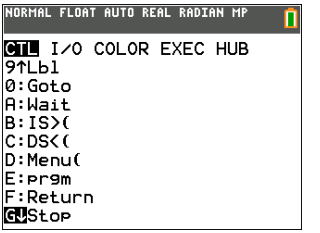
This section describes the **PRGM CTL** (Control) Instructions.

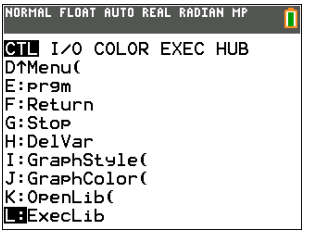
PRGM CTL Menu

To display the **PRGM CTL** (program control) menu, press prgm from the program editor only.

Important Tip: To quickly find a command, use alpha ▲ or alpha ▼ to page through screens.







CTRL **I/O** **COLOR** **EXEC** **HUB**

		Description
1:	If	Creates a conditional test.
2:	Then	Executes commands when **If** is true.
3:	Else	Executes commands when **If** is false.
4:	For(	Creates an incrementing loop.
5:	While	Creates a conditional loop.
6:	Repeat	Creates a conditional loop.

7:	End	Signifies the end of a block.
8:	Pause	Pauses program execution.
9:	Lbl	Defines a label.
0:	Goto	Goes to a label.
A:	Wait	Suspends execution of a program for a given time.
B:	IS>(	Increments and skips if greater than.
C:	DS<(	Decrements and skips if less than.
D:	Menu(	Defines menu items and branches.
E:	prgm	Executes a program as a subroutine.
F:	Return	Returns from a subroutine.
G:	Stop	Stops execution.
H:	DelVar	Deletes a variable from within program.
I:	GraphStyle(	Designates the graph style to be drawn.
J:	GraphColor(	Designates the color of the graph to be drawn
K:	OpenLib(	Extends TI-Basic (not available)
L:	ExecLib(	Extends TI-Basic (not available)

Note: Press [+] when a command is highlighted in a menu to use the syntax help for your programming.

These menu items direct the flow of an executing program. They make it easy to repeat or skip a group of commands during program execution. When you select an item from the menu, the name is pasted to the cursor location on a command line in the program.

To return to the program editor without selecting an item, press [annul].

Controlling Program Flow

Program control instructions tell the TI CE graphing calculator which command to execute next in a program. **If**, **While**, and **Repeat** check a defined condition to determine which command to execute next. Conditions frequently use relational or Boolean tests, as in:

If A<7:A+1→A
or
If N=1 and M=1:Goto Z

If

Use **If** for testing and branching. If *condition* is false (zero), then the *command* immediately following **If** is skipped. If *condition* is true (nonzero), then the next *command* is executed. **If** instructions can be nested.

:If *condition*
:*command* (if true)
:*command*

Program

```
PROGRAM:COUNT
:0→A
:Lbl Z
:A+1→A
:Disp "A IS ",A
:If A≥2
:Stop
:Goto Z
:
```

Output

```
prgmCOUNT
A IS
                1
A IS
                2
Done
```

If-Then

Then following an **If** executes a group of *commands* if *condition* is true (nonzero). **End** identifies the end of the group of *commands*.

:If *condition*
:Then
:*command* (if true)
:*command* (if true)
:End
:*command*

Program

```
PROGRAM:TEST
:1→X:10→Y
:If X<10
:Then
:2X+3→X
:2Y-3→Y
:End
:Disp X,Y
:
```

Output

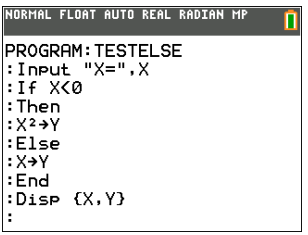
```
prgmTEST
                5
               17
Done
```

If-Then-Else

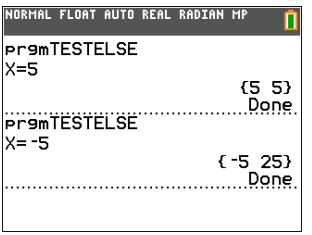
Else following **If-Then** executes a group of *commands* if *condition* is false (zero). **End** identifies the end of the group of *commands*.

:If *condition*
:Then
:*command* (if true)
:*command* (if true)
:Else
:*command* (if false)
:*command* (if false)
:End
:*command*

Program



Output



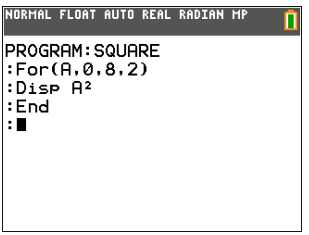
Note: Press entrer to repeat the program.

For(

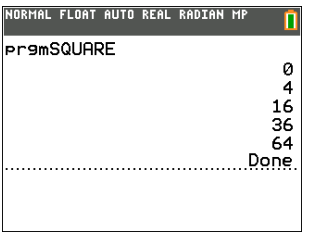
For(loops and increments. It increments *variable* from *begin* to *end* by *increment*. *increment* is optional (default is 1) and can be negative (*end*<*begin*). *end* is a maximum or minimum value not to be exceeded. **End** identifies the end of the loop. **For(** loops can be nested.

:For(*variable*,*begin*,*end*[,*increment*]**)**
:*command* (while *end* not exceeded)
:*command* (while *end* not exceeded)
:End
:*command*

Program



Output

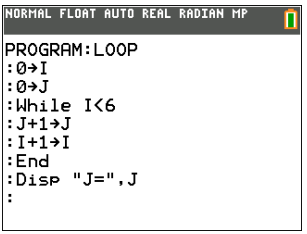


While

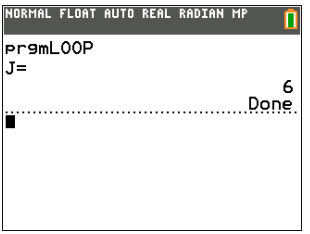
While performs a group of *commands* while *condition* is true. *condition* is frequently a relational test. *condition* is tested when **While** is encountered. If *condition* is true (nonzero), the program executes a group of *commands*. **End** signifies the end of the group. When *condition* is false (zero), the program executes each *command* following **End**. **While** instructions can be nested.

:While *condition*
:*command* (while *condition* is true)
:*command* (while *condition* is true)
:End
:*command*

Program



Output

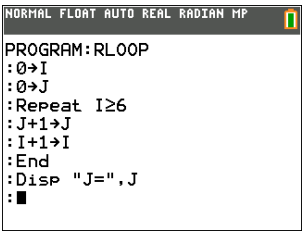


Repeat

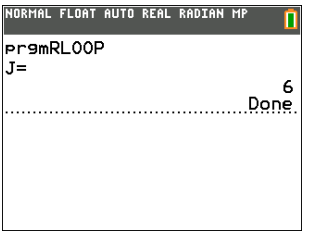
Repeat repeats a group of *commands* until *condition* is true (nonzero). It is similar to **While**, but *condition* is tested when **End** is encountered; therefore, the group of *commands* is always executed at least once. **Repeat** instructions can be nested.

:Repeat *condition*
:*command* (until *condition* is true)
:*command* (until *condition* is true)
:End
:*command*

Program



Output



End

End identifies the end of a group of *commands*. You must include an **End** instruction at the end of each **For(**, **While**, or **Repeat** loop. Also, you must paste an **End** instruction at the end of each **If-Then** group and each **If-Then-Else** group.

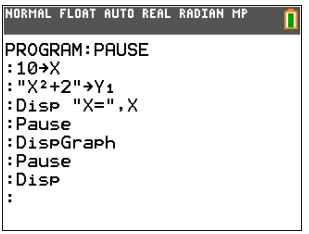
Pause

Pause suspends execution of the program so that you can see answers or graphs. During the pause, the pause indicator is on in the top-right corner.

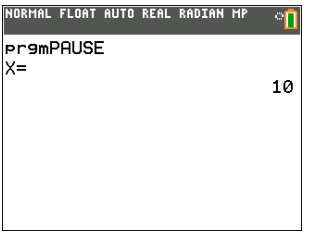
- **Pause** without an argument temporarily pauses the program. If the **DispGraph** or **Disp** instruction has been executed, the appropriate screen is displayed. Press [entrer] to resume execution.
- **Pause** with *value* displays *value* on the current home screen. *value* can be scrolled. **Pause** *value*. Press [entrer] to resume execution.
- **Pause** with *value* and *time* displays value on the current home screen and execution of the program continues for the time period specified. For time only, use Pause *"",time* where the value is a blank string. Time is in seconds. **Pause** *value,time*.

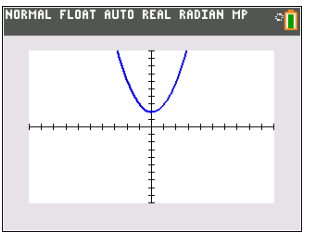
Note: When using TI Connect™ CE Program Editor, Pause must have a space after the command even if no argument is entered.

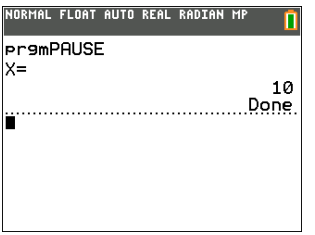
Program



Output







Lbl, Goto

Lbl

Lbl (label) and **Goto** (go to) are used together for branching.

Lbl specifies the *label* for a command. *label* can be one or two characters (A through Z, 0 through 99, or θ).

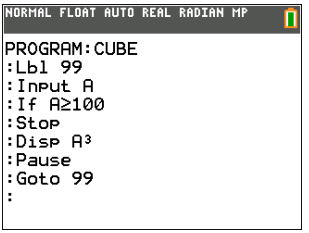
Lbl *label*

Goto

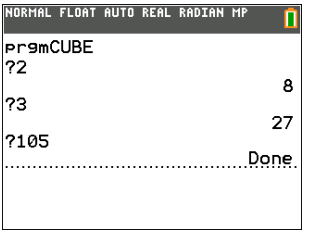
Goto causes the program to branch to *label* when **Goto** is encountered.

Goto *label*

Program



Output

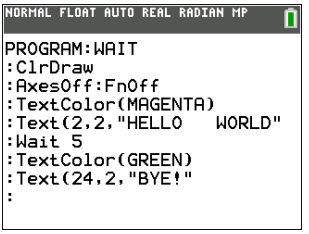


Wait

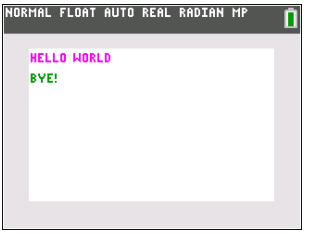
Wait suspends execution of a program for a given time. Maximum time is 100 seconds. During the wait time, the busy indicator is on in the top-right corner of the screen.

Wait *time*

Program



Output: “Bye!” displays after 5 seconds.

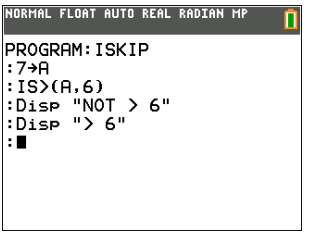


IS>(

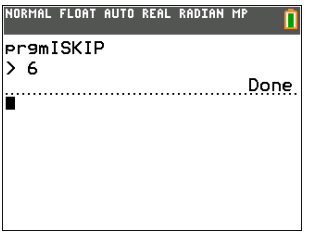
IS>((increment and skip) adds 1 to *variable*. If the answer is > *value* (which can be an expression), the next *command* is skipped; if the answer is ≤ *value*, the next *command* is executed. *variable* cannot be a system variable.

:IS>(*variable*,*value***)**
:*command* (if answer *value*)
:*command* (if answer > *value*)

Program



Output



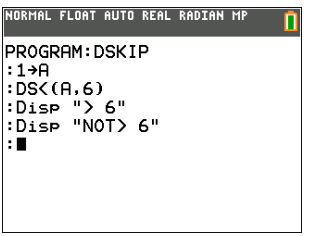
Note: IS>(is not a looping instruction.

DS<(

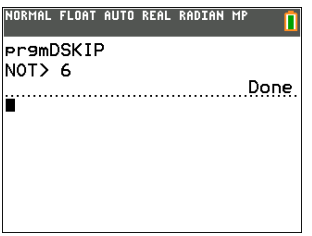
DS<((decrement and skip) subtracts 1 from *variable*. If the answer is < *value* (which can be an expression), the next *command* is skipped; if the answer is ≥ *value*, the next *command* is executed. *variable* cannot be a system variable.

:DS<(*variable*,*value***)**
:*command* (if answer *value*)
:*command* (if answer < *value*)

Program



Output



Note: DS<(is not a looping instruction.

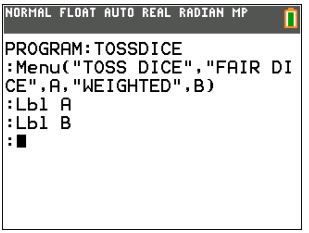
Menu(

Menu(sets up branching within a program. If **Menu(** is encountered during program execution, the menu screen is displayed with the specified menu items, the pause indicator is on, and execution pauses until you select a menu item.

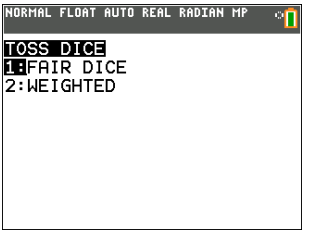
The menu *title* is enclosed in quotation marks ("). Up to nine pairs of menu items are allowed. Each pair comprises a *text* item (also enclosed in quotation marks) to be displayed as a menu selection, and a *label* item to which to branch if you select the corresponding menu selection.

Menu("*title***","***text1***",***label1***,"***text2***",***label2***, . . .)**

Program



Output



The program above pauses until you select **1** or **2**. If you select **2**, for example, the menu disappears and the program continues execution at **Lbl B**.

---

prgm

Use **prgm** to execute other programs as subroutines. When you select **prgm**, it is pasted to the cursor location. Enter characters to spell a program *name*. Using **prgm** is equivalent to selecting existing programs from the **PRGM EXEC** menu; however, it allows you to enter the name of a program that you have not yet created.

prgm*name*

Note: You cannot directly enter the subroutine name when using **RCL**. You must paste the name from the **PRGM EXEC** menu.

Return

Return quits the subroutine and returns execution to the calling program, even if encountered within nested loops. Any loops are ended. An implied **Return** exists at the end of any program that is called as a subroutine. Within the main program, **Return** stops execution and returns to the home screen.

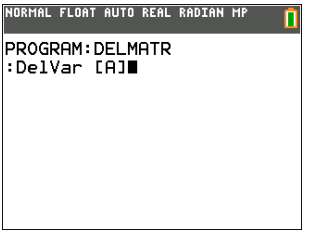
Stop

Stop stops execution of a program and returns to the home screen. **Stop** is optional at the end of a program.

DelVar

DelVar deletes from memory the contents of *variable*.

DelVar *variable*



GraphStyle(

GraphStyle(designates the style of the graph to be drawn. *function#* is the number of the Y= function name in the current graphing mode. *graphstyle* is a number from 1 to 7 that corresponds to the graph style, as shown below.

1 = (Thin)
2 = (Thick)
3 = (Shade above)
4 = (Shade below)
5 = (Path)
6 = (Animate)
7 = (Dot-Thick)
8 = (Dot-Thin)

GraphStyle(*function#*,*graphstyle*)

For example, **GraphStyle(1,5)** in **Func** mode sets the graph style for Y1 to (path; 5).

Not all graph styles are available in all graphing modes.

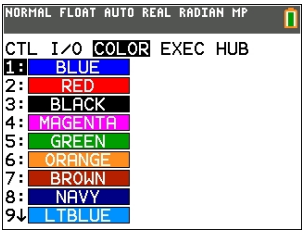
GraphColor

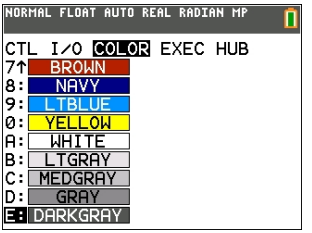
GraphColor(designates the color of the graph to be drawn. *function#* is the number of the Y= function name in the current graphing mode. *color#* is a number from 10 to 24 that corresponds to the graph color, as shown in the table below:

Color Number	Color Name
10	BLUE

11	RED
12	BLACK
13	MAGENTA
14	GREEN
15	ORANGE
16	BROWN
17	NAVY
18	LTBLUE
19	YELLOW
20	WHITE
21	LTGRAY
22	MEDGRAY
23	GRAY
24	DARKGRAY

You can also choose a color name in the [var] menu (**color** sub-menu).





GraphColor(*function#***,***color#***)**

For example,**GraphColor(2, 4) or GraphColor(2, MAGENTA).**

OpenLib(

Extends TI-Basic (not available)

ExecLib(

Extends TI-Basic (not available)

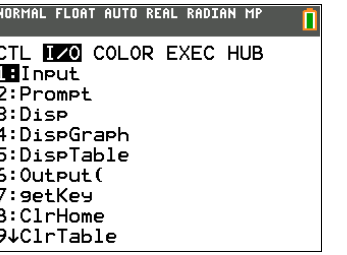
PRGM I/O (Input/Output) Instructions

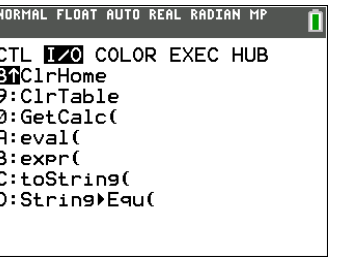
This section describes the **PRGM I/O** (Input/Output) Instructions.

PRGM I/O Menu

To display the **PRGM I/O** (program input/output) menu, press prgm ▶ from within the program editor only.

Important Tip: To quickly find a command, use alpha ▲ or alpha ▼ to page through screens.





CTRL **I/O** **COLOR** **EXEC** **HUB**

		Description
1:	Input	Enters a value or uses the cursor.
2:	Prompt	Prompts for entry of variable values.
3:	Disp	Displays text, value, or the home screen.
4:	DispGraph	Displays the current graph.
5:	DispTable	Displays the current table.
6:	Output(	Displays text at a specified position.
7:	getKey	Checks the keyboard for a keystroke.
8:	ClrHome	Clears the display.
9:	ClrTable	Clears the current table.
0:	GetCalc(	Gets a variable from another TI-82 Advanced Edition Python.
A:	eval(	Returns an evaluated expression as a string with 8 significant digits.

B:	expr(	Converts the character string contained in *string* to an expression and executes it.
C:	toString(	Converts value to a string where *value* can be real, complex, an evaluated expression, list, or matrix.
D:	String▶Equ(	Converts a string into an equation and stores it in Y= var. The string can be a string or string variable.

Note: Press + when a command is highlighted in a menu to use the syntax help for your programming.

These instructions control input to and output from a program during execution. They allow you to enter values and display answers during program execution.

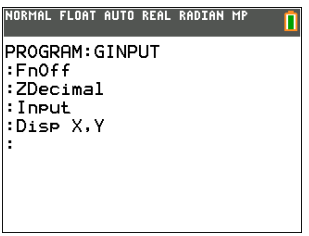
To return to the program editor without selecting an item, press annul.

Displaying a Graph with Input

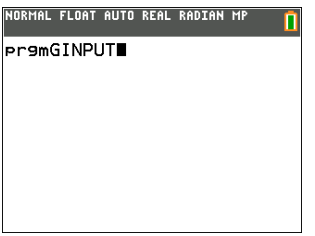
Input without a variable displays the current graph. You can move the free-moving cursor, which updates X and Y (and R and θ for **PolarGC** format). The pause indicator is on. Press entrer to resume program execution.

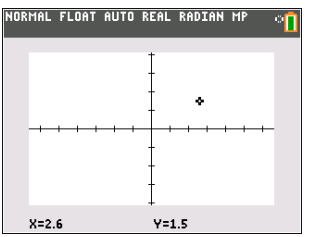
Input

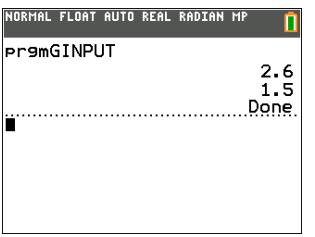
Program



Output







Storing a Variable Value with Input

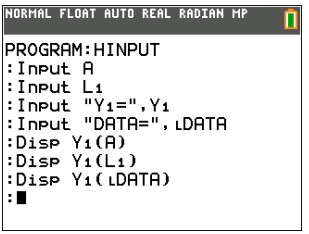
Input with *variable* displays a ? (question mark) prompt during execution. *variable* may be a real number, complex number, list, matrix, string, or Y= function. During program execution, enter a value, which can be an expression, and then press entrer. The value is evaluated and stored to *variable*, and the program resumes execution.

Input [*variable*]

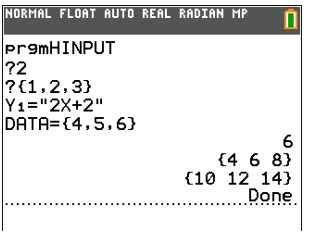
You can display *text* or the contents of **Str***n* (a string variable) of up to 26 characters as a prompt. During program execution, enter a value after the prompt and then press entrer. The value is stored to *variable*, and the program resumes execution.

Input ["*text*",*variable*]
Input [**Str***n*,*variable*]

Program



Output



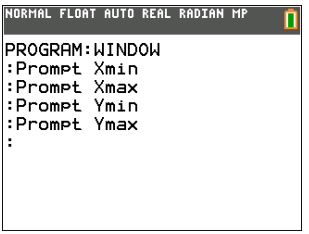
Note: When a program prompts for input of lists and **Y***n* functions during execution, you must include the braces (**{ }**) around the list elements and quotation marks (") around the expressions.

Prompt

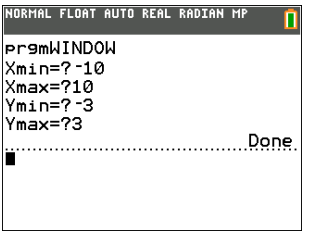
During program execution, **Prompt** displays each *variable*, one at a time, followed by =?. At each prompt, enter a value or expression for each *variable*, and then press entrer. The values are stored, and the program resumes execution.

Prompt *variableA*[,*variableB*,...,*variable n*]

Program



Output



Note: Y= functions are not valid with **Prompt**.

Disp

Displaying the Home Screen

Disp (display) without a value displays the home screen. To view the home screen during program execution, follow the **Disp** instruction with a **Pause** instruction.

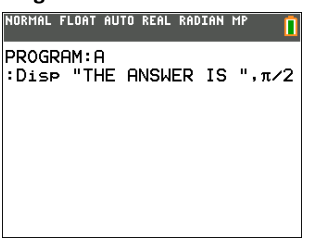
Displaying Values and Messages

Disp with one or more *values* displays the value of each.

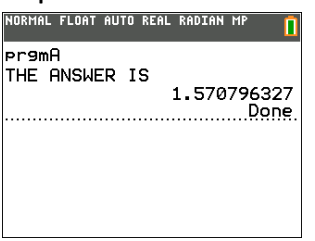
Disp [*valueA*,*valueB*,*valueC*,...,*value n*]

- If *value* is a variable, the current value is displayed.
- If *value* is an expression, it is evaluated and the result is displayed on the right side of the next line.
- If *value* is text within quotation marks, it is displayed on the left side of the current display line. → is not valid as text.

Program



Output



If **Pause** is encountered after **Disp**, the program halts temporarily so you can examine the screen. To resume execution, press enter.

Note: If a matrix or list is too large to display in its entirety, ellipses (**...**) are displayed in the last column, but the matrix or list cannot be scrolled. To scroll, use **Pause** *value*.

DispGraph

DispGraph (display graph) displays the current graph. If **Pause** is encountered after **DispGraph**, the program halts temporarily so you can examine the screen. Press entrer to resume execution.

DispTable

DispTable (display table) displays the current table. The program halts temporarily so you can examine the screen. Press entrer to resume execution.

Output(

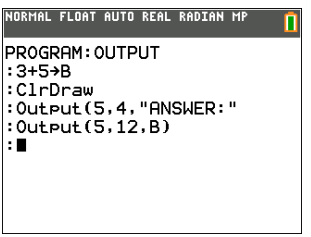
Output(displays *text* or *value* on the current home screen beginning at *row* (1 through 10) and *column* (1 through 26), overwriting any existing characters.

Note: You may want to precede **Output(** with **ClrHome**.

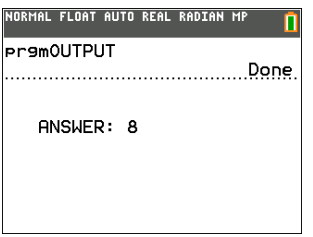
Expressions are evaluated and values are displayed according to the current mode settings. Matrices are displayed in entry format and wrap to the next line. → is not valid as text.

Output(*row*,*column*,"*text*"**)**
Output(*row*,*column*,*value***)**

Program



Output



For **Output(** on a **Horiz** split screen, the maximum value for *row* is 4.

getKey

getKey returns a number corresponding to the last key pressed, according to the key code diagram below. If no key has been pressed, **getKey** returns 0. Use **getKey** inside loops to transfer control, for example, when creating video games.

Program

```
NORMAL FLOAT AUTO REAL RADIAN MP
PROGRAM:GETKEY
:While 1
:getKey→K
:While K=0
:getKey→K
:End
:Disp K
:If K=105
:Stop
:End
```

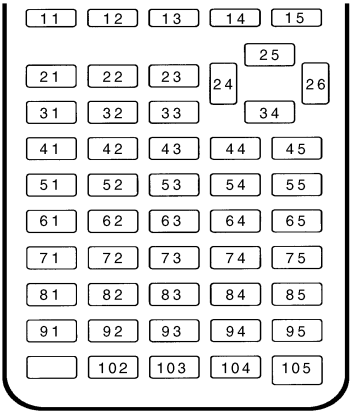
Output

```
NORMAL FLOAT AUTO REAL RADIAN MP
prgmGETKEY
                              41
                              42
                              43
                             105
                            Done
```

Note: [math], [apps], [prgm], and [entrer] were pressed during program execution.

Note: You can press [on] at any time during execution to break the program.

Key Code Diagram



ClrHome, ClrTable

ClrHome (clear home screen) clears the home screen during program execution.

ClrTable (clear table) clears the values in the table during program execution.

GetCalc(

GetCalc(gets the contents of *variable* on another TI-82 Advanced Edition Python and stores it to *variable* on the receiving TI-82 Advanced Edition Python. *variable* can be a real or complex number, list element, list name, matrix element, matrix name, string, Y= variable, graph database, or picture.

GetCalc(*variable*[,*portflag*]**)**

By default, the TI-82 Advanced Edition Python uses the USB port if it is connected. If the USB cable is not connected, it uses the I/O port. If you want to specify either the USB or I/O port, use the following portflag numbers:

portflag=0 use USB port if connected;
portflag=1 use USB port;
portflag=2 use I/O port

Note: TI-82 Advanced Edition Python only links to another TI-82 Advanced Edition Python.

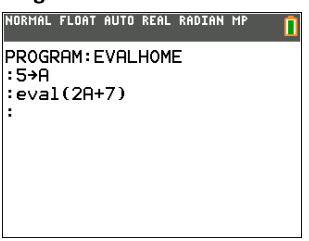
---

eval(

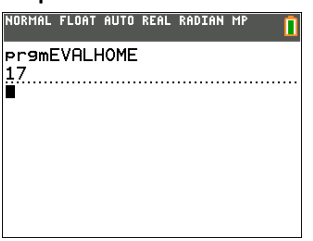
eval(returns an evaluated expression as a string with 8 significant digits. The expression must simplify to a real expression.

eval(*expression*)

Program



Output



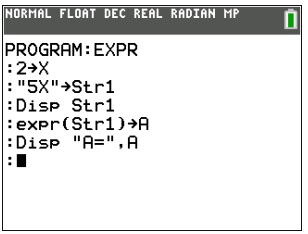
---

expr(

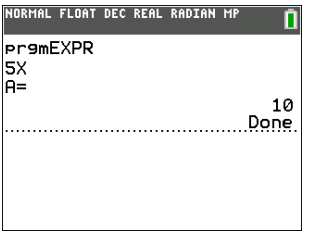
Converts the character string contained in *string* to an expression and executes the expression. *string* can be a string or a string variable.

expr(*string***)**

Program



Output



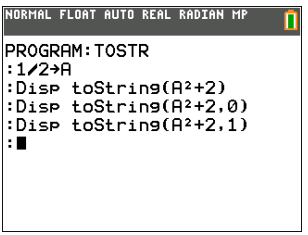
---

toString(

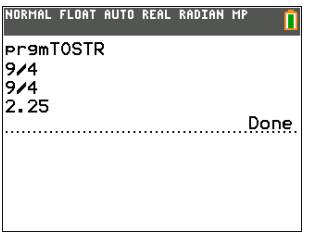
Converts value to a string where *value* can be real, complex, an evaluated expression, list, or matrix. String *value* displays in classic *format (0)* following the mode setting AUTO/DEC or in decimal *format (1)*.

toString(*value*[,*format*]**)**

Program



Output

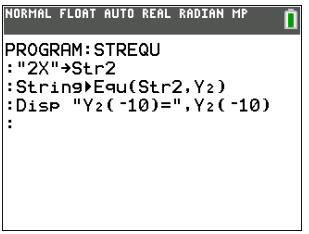


String▶Equ(

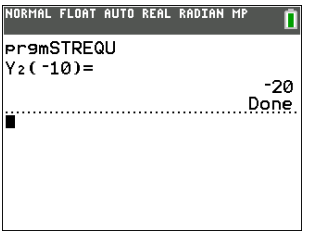
String▶Equ(converts *string* into an equation and stores the equation to **Y***n*. string can be a string or string variable. **String▶Equ(** is the inverse of **Equ▶String(**.

String▶Equ(*string*,**Y***n***)**

Program



Output



PRGM COLOR Instructions

This section describes the **COLOR** menu and the color numbers to use as arguments where setting color is an option such as **GraphColor(**.

You can paste the color token, such as **BLUE**, or use the color number, such as **10**, shown in the table below.

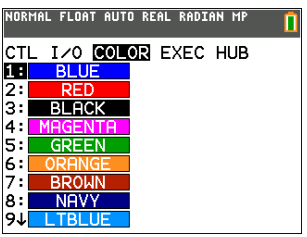
PRGM COLOR Menu

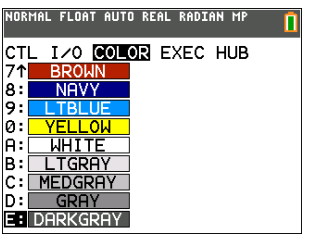
To display the **PRGM COLOR** menu, press prgm ▸ from within the program editor only.

CTRL **I/O** **COLOR** **EXEC** **HUB**

		Description
1:	BLUE	#color = 10
2:	RED	#color = 11
3:	BLACK	#color = 12
4:	MAGENTA	#color = 13
5:	GREEN	#color = 14
6:	ORANGE	#color = 15
7:	BROWN	#color = 16
8:	NAVY	#color = 17
9:	LTBLUE	#color = 18
0:	YELLOW	#color = 19
A:	WHITE	#color = 20
B:	LTGRAY	#color = 21
C:	MEDGRAY	#color = 22
D:	GRAY	#color = 23
E:	DARKGRAY	#color = 24

Note: You can also choose a color name in the var menu (COLOR sub-menu).





PRGM EXEC Instructions

Calling Other Programs as Subroutines

On the TI-82 Advanced Edition Python graphing calculator, any stored program can be called from another program as a subroutine. Enter the name of the program to use as a subroutine on a line by itself.

Calling a Program from Another Program

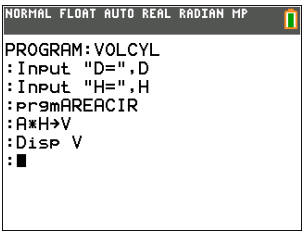
You can enter a program name on a command line in either of two ways.

- Press prgm ◂ to display the **PRGM EXEC** menu and select the name of the program **prgm***name* is pasted to the current cursor location on a command line.
- Select **prgm** from the **PRGM CTL** menu, and then enter the program name.

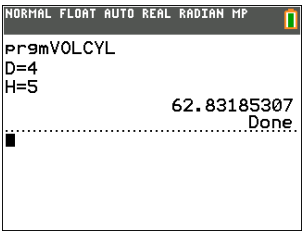
  prgm*name*

When **prgm***name* is encountered during execution, the next command that the program executes is the first command in the second program. It returns to the subsequent command in the first program when it encounters either **Return** or the implied **Return** at the end of the second program.

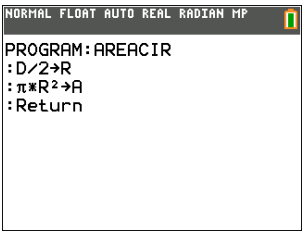
Program



Output



Subroutine ↓↑



Notes about Calling Programs

Variables are global.

label used with **Goto** and **Lbl** is local to the program where it is located. *label* in one program is not recognized by another program. You cannot use **Goto** to branch to a *label* in another program.

Return exits a subroutine and returns to the calling program, even if it is encountered within nested loops.

General Information

Online Help

education.ti.com/eguide

Select your country for more product information.

Contact TI Support

education.ti.com/ti-cares

Select your country for technical and other support resources.

Service and Warranty Information

education.ti.com/warranty

Select your country for information about the length and terms of the warranty or about product service.

Limited Warranty. This warranty does not affect your statutory rights.