Python Programming for the TI-83 Premium CE Graphing Calculator

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.

"Python" and the Python logos are trademarks or registered trademarks of the Python Software Foundation, used by Texas Instruments Incorporated with permission from the Foundation.

© 2019 Texas Instruments Incorporated

Contents

Python Adapter App

See the following for using, navigating, and running the Python Adapter App.

- Using Python Adapter App
- Python Adapter App Navigation
- Example Activity
- Running the Python Adapter App with the TI-Python Adapter

Using Python Adapter App

The Python Adapter App v5.3.5 is available for the TI-83 Premium CE. You will also need the TI-Python Adapter which will be available to educators for classroom use. The information in this eGuide is for use after you have updated your TI-83 Premium CE with CE Bundle v5.3.5. Please see at education.ti.com/83ceupdate to update your TI-83 Premium CE.

The Python Adapter App offers a File Manager, an Editor to create programs, and a Shell to run programs and interact with the Python interpreter. Python programs stored or created as Python AppVars will execute from RAM. You may store Python AppVars in Archive for memory management 2nde [mém] 2:.

.

Python Adapter App Navigation

Use the shortcut keys on the screen in the App to navigate between workspaces in the Python Adapter App. In the image, the shortcut tab labels indicate:

* Navigation to the File Manager [Script]

** Navigation the Editor [Édit] or [Éditer]

*** Navigation to the Shell [Shell]

Access shortcut tabs on the screen using the graphing key row immediately under the screen. Also, see Keypad . The Editor>Tools menu and Shell>Tools menu also contain navigation actions.

Example Activity

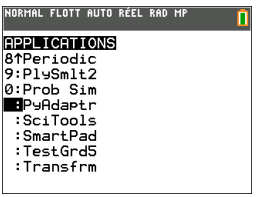
Use the example activity provided as an experience to become familiar with the workspaces in the Python Adapter App.

- Create a new program from the File Manager
- Write the program in the Editor
- Execute the program in the Shell in the Python Adapter App.

For more about Python programming on your CE, please see resources for Python for the TI-83 Premium CE including TI-Codes in Python.

Getting Started:

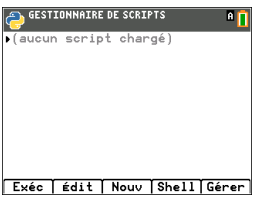
- Connect the TI-Python Adapter using the USB cable.
  - B-side of the cable connects to the Adapter.
- Run the Python Adapter App named PyAdaptr in the 2nde [apps] menu.
  - 2nde [apps] alpha [P] (above [8]) displays the apps starting with P. Then select PyAdaptr.



Note: Actual screens may vary slightly from provided images.

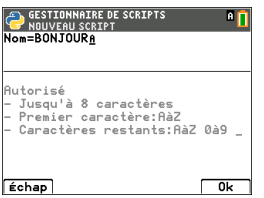
Enter new program name from File Manager.

- Press zoom ([Nouv]) to create a new program.



New File Name Entry

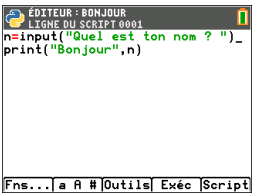
- The example program will be BONJOUR. Enter the program name and press graphe ([Ok]).
- Notice the cursor is in ALPHA lock. Always enter a program name following the given rules on the screen.



TIp: If the cursor is not in ALPHA lock, press 2nde alpha alpha for upper case letters.

Enter program as shown.

TIp: App provides quick entry! Always watch the cursor state as you enter your program!



Alphabet characters on Keypad 	alpha toggles the insert cursor state in the Editor and Shell. _ non-alpha a lower case alpha A upper case ALPHA
Where is the equal sign?	Press sto→ when the cursor is _. 
Where are these located? input() print()	[Fns...] I/O 1:print() 2:input()
Where is double quote?	alpha ["] 
Where are (and)?	Use keypad when cursor is _. 

Try-It! [a A #] and 2nde [catalog] also are helpers for quick entry as needed!

Execute the program BONJOUR

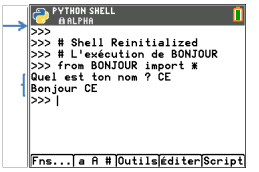
- From the Editor, press trace ([Éxec]) to execute your program in the Shell.
- Enter your name at the "Voltra nom ?" prompt.
- Output displays "Bonjour" with your name.

Note: At the Shell prompt >>>, you can execute a command, such as 2+3. If you use any functions from the math or random module, be sure to execute an import statement first as in any Python coding environment.

Shell cursor state indicator.

Input your name.

Output of BONJOUR displays.



Running the Python Adapter App with the TI-Python Adapter

Setting up a Python Session with your Programs

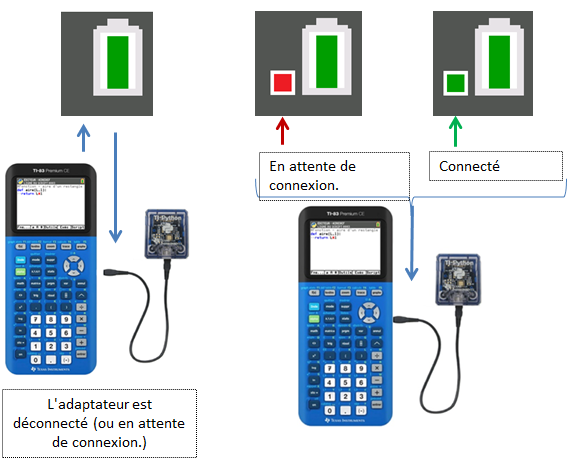
When the Python Adapter App is launched, the CE connection with the TI-Python Adapter will synchronize for your current Python session. You will see your list of programs, in RAM, as they synchronize to the Adapter. When complete, the File Manager will display. Do not disconnect during this Python session setup.

Note: If the Python Adapter App detects an incorrect version of TI-Python on the connected TI-Python Adapter, you may see a progress bar as the proper version is installed. Then, your Python AppVar programs will synchronize with the Adapter and finally display the File Manager. This is a fast process. Please do not disconnect during this process.

During your Python session, the Python programs will always synchronize with the Adapter so new programs created or programs edited will be available to execute. At times, you may see a full synchronization of your programs with the Adapter.

Disconnecting and Reconnecting the TI-Python Adapter

When the Python Adapter App is running, the status bar contains an indicator that signals whether the Adapter is connected and ready for use. Until the connection is established, the CE keypad may not respond. Also, the File Manager>Manage menu will display a static connection status message while open. Best practice is to use the status bar connection indicator while in your Python session.



Screen Captures

When you disconnect the Adapter, screen captures can be taken using TI-Connect CE v5.3.5.

When you reconnect the Adapter and the Python Adapter App is still running, you will return to about the same screen state as when you connected but remember the Shell will reinitialize. If the Python Adapter App is closed prior to reconnection, then a new Python session setup will occur.

Python Workspaces

The Python Adapter App contains three workspaces for your Python programming development.

- File Manager
- Editor
- Shell

Python File Manager

The File Manager lists the Python AppVars available in RAM on your calculator. You can create, edit, and run programs as well as navigate to the Shell.

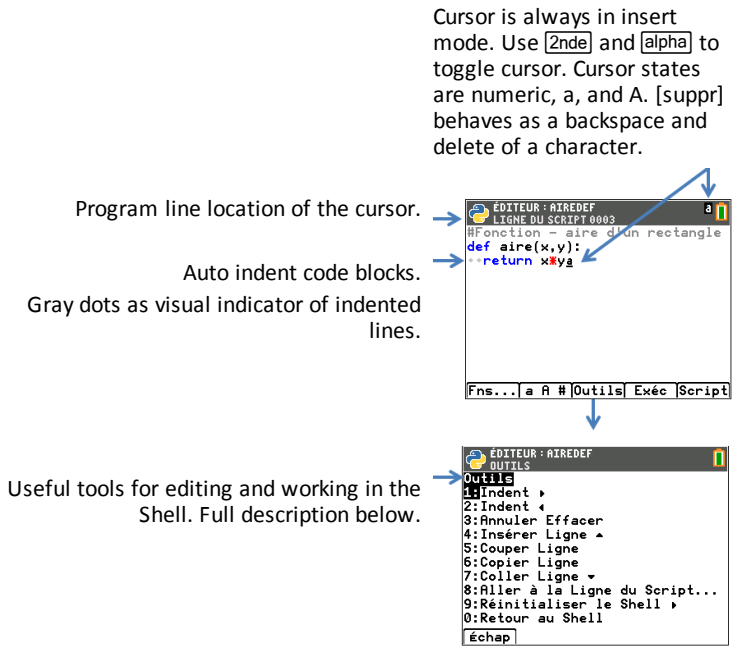
When in alpha state, press any letter on the keypad to jump to programs beginning with that letter.

Press [alpha] if needed when **A** indicator is not in the status bar.

File Manager shortcut keys and menus		
Menus	**Keypress**	**Description**
[Run]	[f(x)]	Select a program using ▲or ▼. Next, select [Run] to execute your program.
[Edit]	[fenétre]	Select a program using ▲or ▼. Next, select [Edit] to display the program in the Editor to edit your program.
[New]	[zoom]	Select [New] to enter a new program name and continue to the Editor to enter your new program.
[Shell]	[trace]	Select [Shell] to display the Shell prompt (Python interpreter). The Shell will be in the current state.
[Manage]	[graphe]	Select [Manage] to: • View connection state of the Adapter when the menu is open. For real time connection update, please observe the connection indicator in the Status Bar. See also: Disconnecting and Reconnecting the TI-Python Adapter • View version number. • Replicate, delete or rename a selected program. • View the About screen. • Quit the App. Also use [2nde] [quitter]

Python Editor

The Python Editor is displayed from a selected program in File Manager or from the Shell. The Editor displays keywords, operators, comments, strings and indents in color. Quick paste of common Python keywords and functions are available as well as direct keypad entry and [a A #] character entry . When pasting a code block such as if.. elif.. else, the Editor offers auto-indent which can be modified as needed as you write your program.



Python Editor shortcut keys and menus		
Menus	**Keypress**	**Description**
[Fns…]	f(x)	Select [Fns…] to access menus of commonly used functions, keywords, and operators. Also access selected contents of the math and random modules. **Note:** 2nde [catalog] is also helpful for quick paste.
[a A #]	fenêtre	Select [a A #] to access a character palette as an alternate way to enter many characters.
[Tools]	zoom	Select [Tools] to access features to assist in

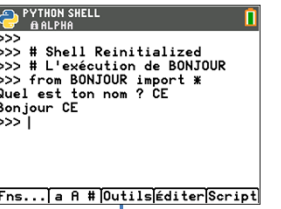
<table>
<tr><th colspan="3">Python Editor shortcut keys and menus</th></tr>
<tr><th>Menus</th><th>Keypress</th><th>Description</th></tr>
<tr><td></td><td></td><td>your editing or your interaction with the Shell.<table>
<tr><td>1: Indent ▸</td><td>Indents the program line to the right cursor moves to first character of the line.</td></tr>
<tr><td>2: Indent ◂</td><td>Reduces the indent of the program line to the left. Cursor moves to first character of the line.</td></tr>
<tr><td>3: Undo Clear</td><td>Pastes the last cleared line to a new line below the program line containing the cursor. Cursor displays at the end of the pasted line.</td></tr>
<tr><td>4: Insert Line Above</td><td>Inserts a line above the program line with the cursor. Line will indent and display indent dots when appropriate.</td></tr>
<tr><td>5: Cut Line</td><td>Current program line with cursor is cut. Cursor displays on program line below the cut line.</td></tr>
<tr><td>6: Copy Line</td><td>Copies current program line with cursor. A copied program line can be pasted to the Shell prompt. See Shell below.</td></tr>
<tr><td>7: Paste Line Below</td><td>Pastes the last stored program line to the line below the cursor position.</td></tr>
<tr><td>8: Go to Program Line...</td><td>Displays cursor at the beginning of the specified program line.</td></tr>
<tr><td>9: Go to New Shell</td><td>Displays reinitialized Shell.</td></tr>
<tr><td>0: Return to Shell</td><td>Displays Shell in current state.</td></tr>
</table></td></tr>
<tr><td>[Run]</td><td>trace</td><td>Select [Run] to execute your program.</td></tr>
<tr><td>[Files]</td><td>graphe</td><td>Select [Files] to display the File Manager.</td></tr>
</table>

Python Shell

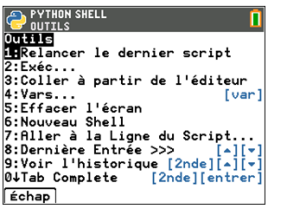
The Python Shell is the console where you can interact with the Python interpreter or run your Python programs. Quick paste of common Python keywords and functions is available as well as direct keypad entry and [a A #] character entry . The Shell prompt can be used to test one line of code pasted from the Editor. Multiple lines of code may also be entered and run at a Shell prompt >>>.

Shell cursor state indicator.

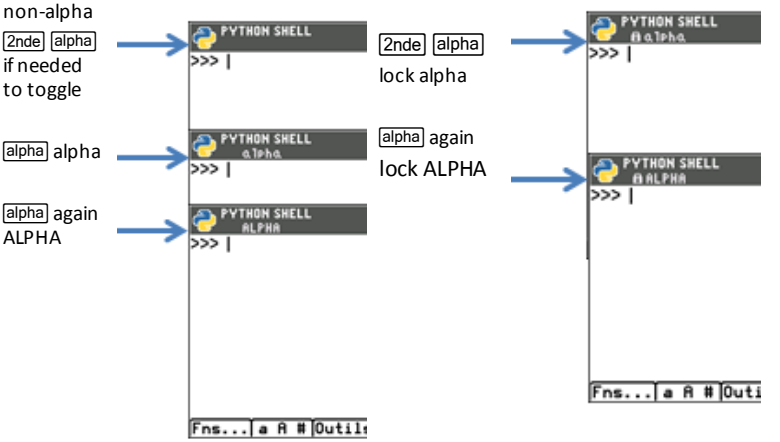
Shell reinitialize when a new program is executed.



Useful tools for working in the Shell. See details below.



Shell Cursor States



Python Shell shortcut keys and menus		
Menus	**Keypress**	**Description**
[Fns…]	f(x)	Select [Fns…] to access menus of commonly used functions, keywords, and operators. Also access selected contents of the math and random modules. **Note:** 2nde [catalog] is also helpful for quick paste.
[a A #]	fenêtre	Select [a A #] to access a character palette as an alternate way to enter many characters.
[Tools]	zoom	Select [Tools] to display the following menu items.

1: Rerun last program	Reruns last program which was executed in the Shell.
2: Run…	Displays a list of the Python programs available to run in Shell.
3: Paste from Editor	Pastes the last copied program line from the Editor to the Shell prompt.
4: Vars…	Displays the vars from the last program which ran. Does not display program defined vars from an imported program.
5: Clear	Clears the Shell screen. Does not

Python Shell shortcut keys and menus			
Menus	**Keypress**	**Description**	
		Screen	reinitialize a new Shell.
		6: New Shell	Reinitialize a new Shell.
		7: Go to Program Line...	Displays the Editor from the Shell with cursor on the specified program line.
		8: Last Entry>>> 2nde ▲ ▼	Displays up to the last 8 entries at the Shell prompt during a Shell session.
		9: View History 2nde ▲ 2nde ▼	Scroll the Shell screen to view up to the last 60 lines of output in the Shell during a Shell session.
		0: Tab Complete 2nde entrer	Displays the names of the variables and functions available for access in the current Shell session. When a letter of an available variable or function is entered, press 2nde entrer to auto-complete the name if a match is available in the current Shell session.
		A: from PROGRAM import *...	When first executed in a Shell session, PROGRAM will run and vars will only be viewable using Tab Complete. When executed again in the same Shell session, the execution will appear as no execution. This command can also be pasted from 2nde [catalog].
[Editor]	trace	Select [Editor] to display the Editor with the last programs in Editor. If Editor is empty, you can display File Manager.	
[Files]	graphe	Select [Files] to display the File Manager.	

Entries - Keypad, Catalog, Character Map, and Menus

Tips for fast entry

- Keypad
- Catalog
- [a A #] Character Map
- [Fns…] Menus

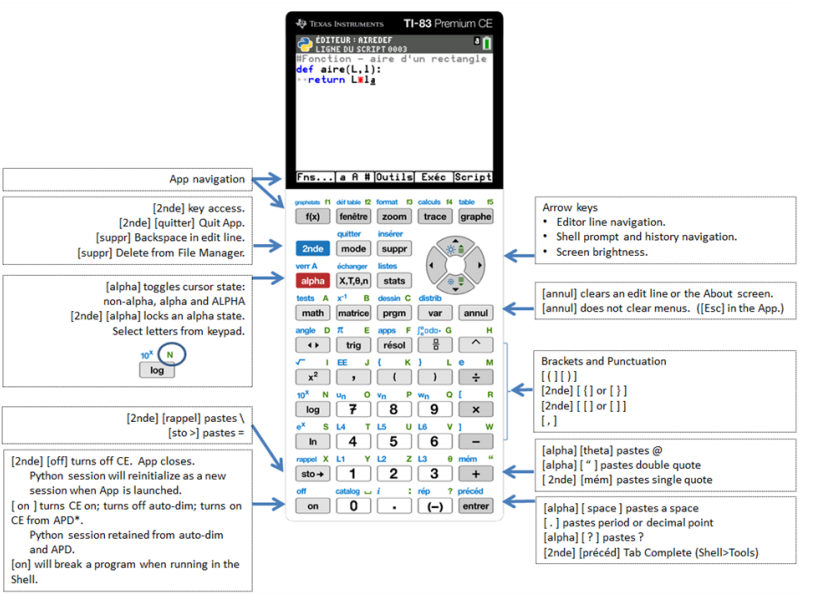
Using the Keypad, Catalog, [a A #], and Fns… menus

When entering code in the Editor or in the Shell, use the following entry methods to quickly paste to the edit line.

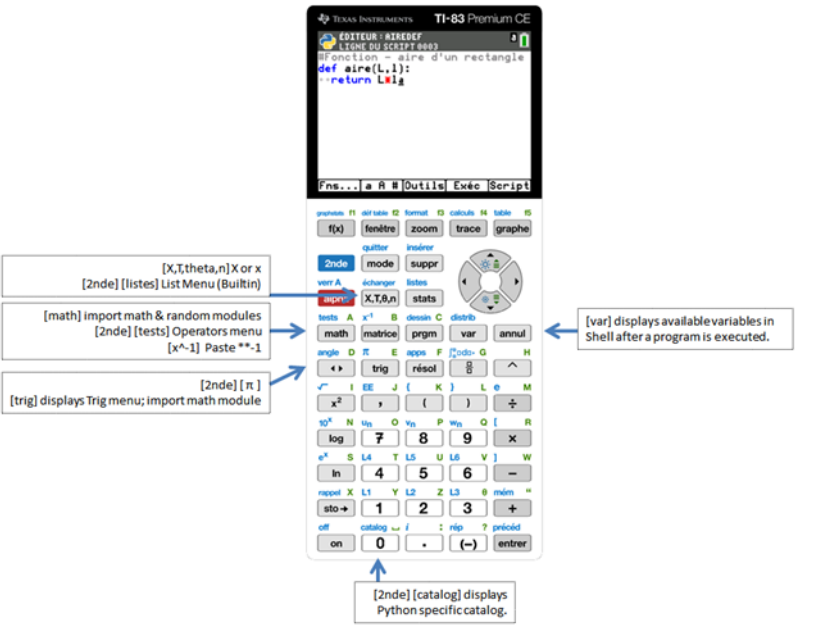
Keypad

When the Python App is running, the keypad is designed to paste the appropriate Python operations or open menus designed for easy entry of functions, keywords, methods, operators, etc. Pressing 2nde and alpha will access the second and third functions on a key as in the Operating System.

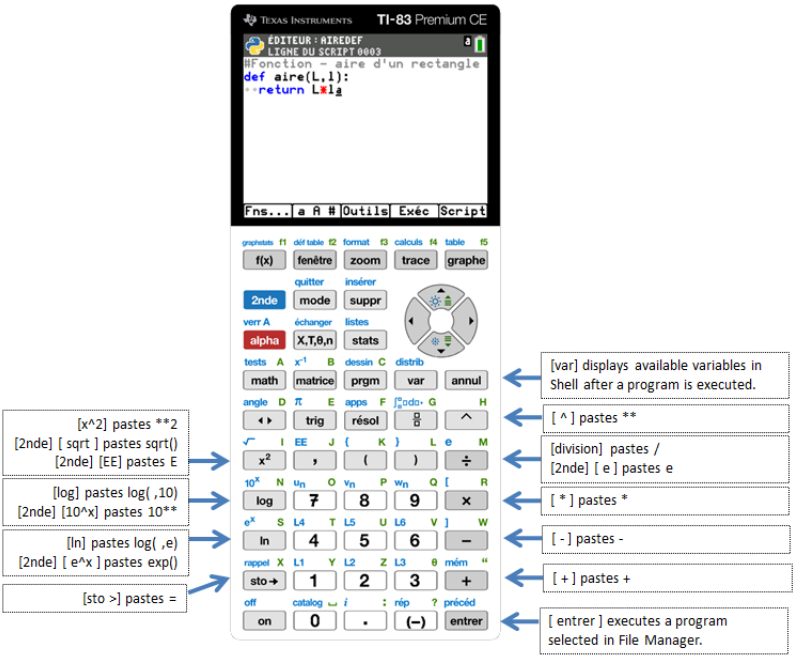
Python Adapter App Navigation, Editing, and Special Characters by Keypad Rows



Python Adapter App Specific Key Presses for Menus and Functions by Keypad Rows



Python Adapter App Specific Key Presses for Menus and Functions by Keypad Rows (Continued)



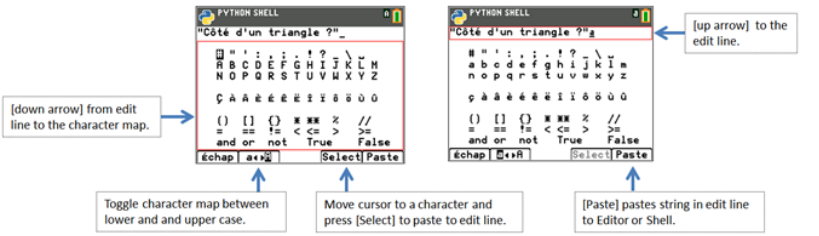
Catalog

When the Python Adapter App is running, 2nde [catalog] will display a list of frequently used delimiters, keywords, functions and operators to quickly paste to an edit line. 2nde [catalog] is available in Editor and Shell only. For a more detailed description of each Catalog item, please see the Reference Guide. From the top of the catalog menu, use ▲ for circular navigation of the catalog.

When in catalog, select alpha and a letter key to display the listing starting at that letter.

[a A #] Character Map

[a A #] shortcut tab to a character palette is a convenient feature to enter strings when in Editor or Shell.



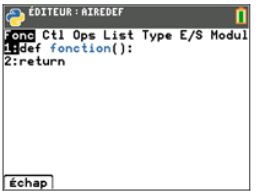
Note: When the cursor focus is in the [a A #] edit line, selected keypad keys are not available. When focus is in the character map, the keypad is restricted.

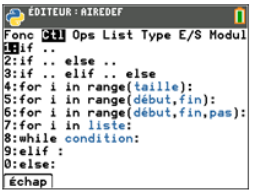
[Fns…] Menus

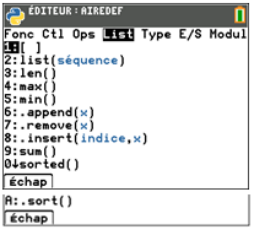
[Fns…] shortcut tab displays menus containing frequently used Python functions, keywords, and operators. The menus also provide access to the selected functions and constants from the math and random modules. While you can enter character by character from the keypad, these menus provide a quick way to paste in Editor or Shell. Press [Fns…] when in Editor or Shell. See also Catalog and Keypad for alternate entry methods.

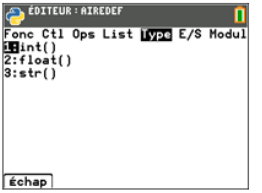
Functions and Modules Submenus

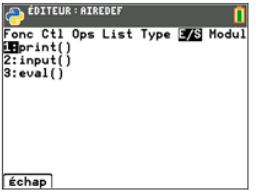
Built-in, Operators and Keywords











Module Submenus

When using a Python function or constant from a module, always use an import statement to indicate the module location of the function or constant. See Modules Included in the TI-83 Premium CE Python Experience

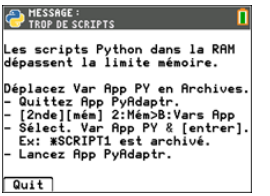
Python App Messages

There are several messages that may display while you are in a Python session. Some selected messages are given in the table. Please follow the instructions on the screen and navigate using [quitter], [Échap] or [Ok] as needed.

Memory Management

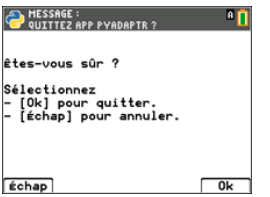
Python files synchronize with the Adapter. If you have too many Python AppVars in RAM in your CE for the Adapter memory*, when the Python Adapter App synchronizes with the TI-Python Adapter, you will be directed to move some of your programs from RAM to Archive memory.

*TI-Python Adapter memory may hold up to 40K or 80 Python programs whichever comes first.

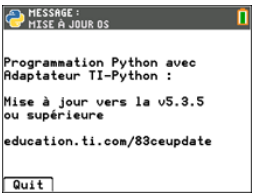


Use [2nde] [quitter] to quit the App

You will be prompted to make sure you want to quit the App. Quitting the App will stop your Python session. When you run the Python Adapter App again, your Python AppVar programs will synchronize with the Adapter. The Shell will reinitialize.



When you run the Python Adapter App, your TI-83 Premium CE OS must be v5.3.5 or higher. Please go to education.ti.com/83ceupdates to get all the latest calculator files for your CE.

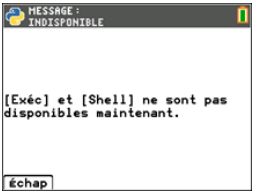


In File Manager, you press [suppr] on a selected Python program or you select from File Manager>Manage 2:Supprimer le script…

You will see a dialog to delete or escape back to the File Manager.

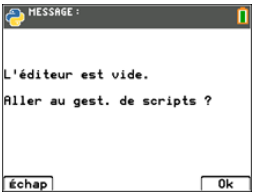
TI-Python Adapter is not connected.

The File Manager and Editor are available for use when the TI-Python Adapter is not connected. If you attempt to execute a program or go to the Shell, a reminder is given that the Adapter is not available now. Connect the TI-Python Adapter, wait for the connection to establish and then continue your work.

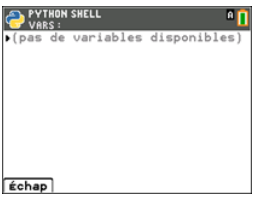


You attempt to create a new or duplicate a Python program that already exists on your CE either in Archive or disabled for exam mode. Enter a different name.

You attempt to navigate from the Shell to the Editor but the Editor is empty. Select an appropriate option for your work.



When you execute a Python program, defined variables from the last program executed are listed in the Shell>Tools> 4:Vars… menu to use again in the Shell. If no variables display, you may need to run your program again.



Errors

The TI-Python Adapter will display Python error messages in the Shell when code is executed. If an error is displayed when a program executes, a program line number will display. Use Shell>Tools 7:Go to Program Line… Enter the line number and press [Ok]. The cursor will display on the first character of the appropriate program line in the Editor. The program line number is displayed in the second line of the Status bar in the Editor.

See Selected Module Content in Builtins for the list of Python errors (exceptions) supported in this release.

How to use the Python Adapter App in TI-SmartView™ CE for TI-83 Premium CE

- Download the free TI-83 Premium CE emulator OS update, *.o83, file at education.ti.com/83ceupdate.
- Launch TI-SmartView™ CE v5.3.0.
- To update the TI-83 Premium CE emulator
  - Select Actions>Emulator OS Update...
  - Load the downloaded *.o83 file.
  - The 83CE emulator experience will be up to date with 83CE OS v5.3.5 and PyAdaptr App v 5.3.5.
- PyAdaptr App offers the File Manager and Editor only for class demonstrations of creating and editing a program when running in TI-SmartView™ CE.
- There is no computer connectivity with the TI-Python Adapter when using TI-SmartView™ CE.
- SmartPad App will remotely press the keypad when the PyAdaptr App is running.
- *.py files do not convert in TI-SmartView™ CE in this release. Please use TI Connect™ CE v5.3.5 for the conversion.
- Send PY AppVars to the Emulator Explorer as needed.
  - Please quit PyAdaptr App before switching to Emulator Explorer to send/recieve PY AppVars.

Using TI Connect™ CE to Convert Python Programs

Please see TI Connect™ CE v5.3.5 in the TI-83 Premium CE e-Guide for details on converting *.py program to a PY AppVar as the CE calculator file format.

What is the Python programming experience?

Python on the TI-83 Premium CE using the Python App is based on CircuitPython, a variant of Python designed to fit in small microcontrollers. The original CircuitPython implementation has been adapted for use by TI.

The internal storage of numbers for computation in this variant of Circuit Python is in limited-precision binary floats and thus cannot exactly represent all possible decimal values. The differences from actual decimal representations that arise when storing these values can lead to unexpected results in subsequent calculations.

- **For Floats** - Displays up to 16 significant digits of precision. Internally, values are stored using 53 bits of precision, which is roughly equivalent to 15-16 decimal digits.
- **For Integers** - The size of integers is limited only by the memory available at the time calculations are performed.

Modules Included in the TI-83 Premium CE Python Experience

- Built-ins
- math*
- random*

***Note:** If you have existing Python programs from another Python development environment, please modify your program to access only what is available on the TI-Python Adapter solution.

As with any version of Python, you will need to include “from math import *” and/or “from random import *” to use any functions, methods, or constants contained in the math module or the random module. For an example, to execute the cos() function, use import to import the math module for use.

See CATALOG Listing.

Example:

```
>>>from math import *
>>>cos(0)
1.0
```

Alternate Example:

```
>>>import math
>>>math.cos(0)
1.0
```

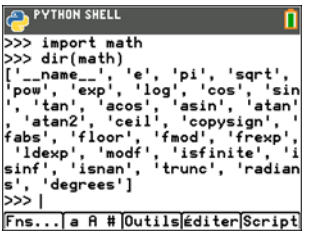
Modules available can be displayed in the Shell using the following command

```
>>> help("modules")
__main__ sys gc
random time array
math builtins collections
```

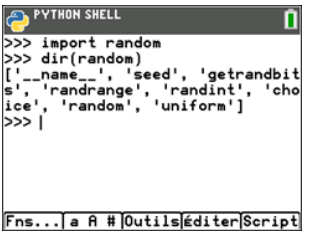
Content of modules can be viewed in the Shell as shown using “import module” and “dir(module).”

These screens display the module contents for math and random.

Not all module contents appear in the quick paste menus such as [Fns…] or 2nde [catalog].



math module



random module

Contents of selected modules and keywords

For list of the modules included in this release, please see:

Selected Module Content for Python Adapter App v5.3.5

Reference Guide for TI-Python Experience

The Python Adapter App contains menus of functions, classes, controls, operators and keywords for quick pasting in the Editor or Shell. The following reference table contains the listing of features in 2nde [catalog] when the App is running. For a complete listing of Python functions, classes, operators, and keywords available in this version, please see "Contents of selected modules and keywords."

This table is not intended to be an exhaustive list of Python available in this offering. Other functions supported in this Python offering can be entered using the alpha keys from the keypad.

Most examples given in this table run at the Shell prompt (>>>).

CATALOG Listing

Alphabetical List

- A
- B
- C
- D
- E
- F
- G
- I
- L
- M
- N
- O
- P
- R
- S
- T
- U
- W
- Y

#

Delimiter 2nde [catalog]

Syntax: #Your comment about your program.

Description: In Python, a comment begins with the hash tag character, #, and extends to the end of the line. [a A #]

Example:

```
#A short explanation of the code.
```

%

Operator 2nde [catalog]

Syntax: x%y or x % y

Description: Returns remainder of x/y. (Modulus) Preferred use is when x and y are integers. [a A #]

Example:

```
>>>57%2
1
```

See also fmod(x,y).

//

Operator 2nde [catalog]

Syntax: x//y or x // y

Description: Returns the floor division of x/y. [a A #]

Example:

```
>>>26//7
3
>>>65.4//3
21.0
```

[a A #]

Description: Launch [a A #] character palette.

Includes ç à â è é ê ë î ï ô ö ù û

[a A #] shortcut is on screen at [fenêtre] in the Editor or Shell

abs()

Module: Built-in

[2nde] [catalog]

Syntax: abs(x)

Description: Returns the absolute value of a number. In this release, the argument may be an integer or floating point number.

Note: fabs() is a function in the math module.

Example:

```
>>>abs(-35.4)
35.4
```

acos()

Module: math

[trig] 7:acos()

Syntax: acos(x)

Description: Returns arc cosine of x in radians.

[2nde] [catalog]

Example:

```
>>>from math import *
>>>acos(1)
0.0
```

[Fns…] Modul 1:math… > Trig 7:acos()

Alternate Example: [Tools] > 6:New Shell

```
>>>import math
>>>math.acos(1)
0.0
```

import commands can be found in [2nde] [catalog]

and

Keyword

2nde [tests]
Ops 8:and

Syntax: x and y

Description: May return True or False. Returns "x" if "x" is False and "y" otherwise. Pastes with space before and after and. Edit as needed.

[Fns…] > Ops 8:and

Example:

```
>>>2<5 and 5<10
True
>>>2<5 and 15<10
False
>>>{1} and 3
3
>>>0 and 5 < 10
0
```

2nde [catalog]

[a A #]

.append(x)

Module: Built-in

2nde [listes]
List
6: .append(x)

Syntax: listname.append(item)

Description: The method append() appends an item to a list.

2nde [catalog]

Example:

```
>>>listA = [2,4,6,8]
>>>listA.append(10)
>>>print(listA)
[2,4,6,8,10]
```

[Fns…] > List
6:.append(x)

as

Keyword

2nde [catalog]

Description: Use as to create an alias when importing a module. See Python documentation for more details.

asin()

Module: math

trig 6:asin()

Syntax: asin()

Description: Returns arc sine of x in radians.

2nde [catalog]

Example:

```
>>>from math import *
>>>asin(1)
1.570796326794897
```

[Fns…] > Modul 1:math… > Trig 6:asin()

Alternate Example:

```
>>>import math
>>>math.asin(1)
1.570796326794897
```

import commands can be found in 2nde [catalog]

assert

Keyword

2nde [catalog]

Description: Use assert to test a condition in your code. Returns None or if not, execution of the program will display an AssertionError.

atan()

Module: math

trig 8:atan()

Syntax: atan(x)

Description: Returns arc tangent of x in radians.

[Fns…] > Modul 1:math… > Trig 8 :atan()

Example:

```
>>>from math import *
>>>atan(1)*4
3.141592653589793
```

2nde [catalog]

Alternate Example:

```
>>>import math
>>>math.atan(1)*4
3.141592653589793
```

import commands can be found in 2nde [catalog]

atan2(y,x)

Module: math

Syntax: atan2(y,x)

Description: Returns arc tangent of y/x in radians. Result is in [-pi, pi].

Example:

```
>>>from math import *
>>>atan2(pi,2)
1.003884821853887
```

Alternate Example:

```
>>>import math
>>>math.atan2(math.pi,2)
1.003884821853887
```

trig 9:atan2 ()

[Fns…] > Modul 1:math… > Trig 9:atan2()

2nde [catalog]

import commands can be found in 2nde [catalog]

B

break

Keyword 2nde [catalog]

Description: Use break to break out of a for or while loop.

.

C

ceil()

Module: math

Syntax: ceil(x)

Description: Returns the smallest integer greater than or equal to x.

Example:

```
>>>from math import *
>>>ceil(34.46)
35
>>>ceil(678)
678
```

[math] Modul
1:math... Math
8:ceil()

[2nde] [catalog]

[Fns...] Modul
1:math...Math
8:ceil()

import commands can be found in
[2nde] [catalog]

choice(sequence)

Module: random

Syntax: choice(sequence)

Description: Returns a random element from a non-empty sequence.

Example:

```
>>>from random import *
>>>listA=[2,4,6,8]
>>>choice(listA)        #Your result may differ.
4
```

[math] Modul
2:random...
Random
5:choice(sequence)

[2nde] [catalog]

[Fns...] Modul
2:random...
Random
5:choice(sequence)

import commands can be found in
[2nde] [catalog]

class

Keyword 2nde [catalog]

Description: Use class to create a class. See Python documentation for more details.

continue

Keyword 2nde [catalog]

Description: Use continue in a for or while loop to end the current iteration. See Python documentation for more details.

cos()

Module: math

trig Trig 4: cos()

Syntax: cos(x)

Description: Returns cos of x. Angle argument is in radians.

2nde [catalog]

Example:

```
>>>from math import *
>>>cos(0)
1.0
>>>cos(pi/2)
6.123233995736767e-17
```

[Fns...] Modul 1:math... > Trig 4:cos()

Alternate Example:

```
>>>import math
>>>math.cos(0)
1.0
```

Note: Python displays scientific notation using e or E. Some math results in Python will be different than in the CE OS.

.count()

Module: Built-in

2nde [catalog]

Syntax: listname.count(item)

Description: count()is a method that returns the number of occurrences of an item in a list, tuple, bytes, str, bytearray, or array.array object.

Example:

```
>>>listA = [2,4,2,6,2,8,2,10]
>>>listA.count(2)
4
```

def function():

Keyword

2nde [catalog]

Syntax: def function(var, var,...)

Description: Define a function dependent on specified variables. Typically used with the keyword return.

[Fns...] > Func 1:def function ():

Example:

[Fns...] > Func 2:return

```
>>>    def f(a,b):
...          return a*b
...
...
...
>>>          f(2,3)
6
```

degrees()

Module: math

trig Trig 2:degrees()

Syntax: degrees(x)

Description: Converts angle x in radians to degrees.

2nde [catalog]

Example:

```
>>>from math import *
>>>degrees(pi)
180.0
>>>degrees(pi/2)
90.0
```

[Fns...] > Modul 1:math... > Trig 2:degrees()

del

Keyword

2nde [catalog]

Description: Use del to delete objects such as variables, lists, etc.
See Python documentation for more details.

E

e

Module: math

2nde [e] (above ÷)

Syntax: math.e or e if math module was imported

Description: Constant e displays as shown below.

Example:

[Fns…] > Modul 1:math… > Const 1:e

```
>>>from math import *
>>>e
2.718281828459045
```

Alternate Example:

```
>>>import math
>>>math.e
2.718281828459045
```

elif :

Keyword

2nde [catalog]

See if..elif..else.. for details.

[Fns…] > Ctl

1:if..

2:if..else..

3:if..elif..else

9:elif :

0:else:

else:

Keyword 2nde [catalog]

See if..elif..else.. for details.

[Fns…] > Ctl

1:if..

2:if..else..

3:if..elif..else

9:elif :

0:else:

eval()

Module: Built-in 2nde [catalog]

Syntax: eval(x)

Description: Returns the evaluation of the expression x. [Fns…] I/O 3:eval()

Example:

```
>>>a=7
>>>eval("a+9")
16
>>>eval('a+10')
17
```

except exception:

Keyword 2nde [catalog]

Description: Use except in a try..except code block.
See Python documentation for more details.

exp()

Module: math

Syntax: exp(x)

Description: Returns e**x.

Example:

```
>>>from math import *
>>>exp(1)
2.718281828459046
```

Alternate Example: [Tools] > 6:New Shell

```
>>>import math
 >>>math.exp(1)
2.718281828459046
```

2nde [e^x] (above ln)

2nde [catalog]

[Fns…] > Modul
1:math…
4:exp()

import commands can be found in 2nde [catalog].

.extend()

Module: Built-in

Syntax: listname.extend(newlist)

Description: The method extend() is a method to extend newlist to the end of a list.

Example:

```
>>>listA = [2,4,6,8]
>>>listA.extend([10,12])
>>>print(listA)
[2,4,6,8,10,12]
```

2nde [catalog]

fabs()

Module: math

2nde [catalog]

Syntax: fabs(x)

Description: Returns the absolute value of x

[Fns…] > Modul
1:math...
2:fabs()

Example:

```
>>>from math import *
>>>fabs(35-65.8)
30.8
```

import commands can be found in 2nde [catalog].

See also Built-in function abs().

False

Keyword

2nde [tests] (above math)

Description: Returns False when statement executed is False. "False" represents the false value of objects of type bool.

2nde [catalog]

Example:

```
>>>64<=32
False
```

[Fns…] > Ops
B:False

[a A #]

finally:

Keyword

2nde [catalog]

Description: Use finally in a try..except..finally code block. See Python documentation for more details.

float()

Module: Built-in

2nde [catalog]

Syntax: float(x)

Description: Returns x as a float.

Example:

[Fns…] > Type 2:float()

```
>>>float(35)
35.0
>>>float("1234")
1234.0
```

floor()

Module: math

math Modul 1:math 9:floor()

Syntax: floor(x)

Description: Returns the largest integer less than or equal to x.

2nde [catalog]

Example:

[Fns…] > Modul 1:math 9:floor()

import commands can be found in 2nde [catalog]

```
>>>from math import *
>>>floor(36.87)
36
>>>floor(-36.87)
-37
>>>floor(254)
254
```

fmod(x,y)

Module: math

Syntax: fmod(x,y)

Description: See Python documentation for more details. Preferred use is when x and y are floats.

May not return the same result as x%y.

Example:

```
>>>from math import *
>>>fmod(50.0,8.0)
2.0
>>>fmod(-50.0,8.0)
-2.0
>>>-50.0 - (-6.0)*8.0      #validation from description
-2.0
```

See also: x%y.

[math] Modul
1:math
7:fmod()

[2nde] [catalog]

[Fns…] >
Modul
1:math…
7:fmod()

import commands can be found in [2nde] [catalog]

for i in list:

Keyword

Syntax: for i in list:

Description: Used to iterate over list elements.

Example:

```
>>> for i in [2,4,6]:
…       print(i)
…
…
…
2
4
6
```

[Fns…] Ctl
7:for i in list:

[2nde] [catalog]

for i in range(size):

Keyword

Syntax: for i in range(size)

Description: Used to iterate over a range.

Example:

```
>>> for i in range(3):
…       print(i)
…
…
…
0
1
2
```

[Fns…] Ctl 4:for i in range (size):

2nde [catalog]

for i in range(start,stop):

Keyword

Syntax: for i in range(start,stop)

Description: Used to iterate over a range.

Example:

```
>>> for i in range(1,4):
…       print(i)
…
…
…
1
2
3
```

[Fns…] Ctl 5:for i in range (start,stop):

2nde [catalog]

for i in range(start,stop,step):

Keyword

Syntax: for i in range(start,stop,step)

Description: Used to iterate over a range.

Example:

```
>>> for i in range(1,8,2):
…       print(i)
…
…
…
1
3
4
7
```

[Fns…] Ctl
6:for i in range
(start,stop,step):

2nde [catalog]

frexp()

Module: math

Syntax: frexp(x)

Description: Returns a pair (y,n) where x == y * 2**n. y is float where 0.5<abs(y)<1; and n is integer.

Example:

```
>>>from math import *
>>>frexp(2000.0)
(0.9765625, 11)
>>>0.9765625 * 2**11      #validate description
2000.0
```

math Modul
1:math
A:frexp()

2nde [catalog]

[Fns…] >
Modul
1:math
A:frexp()

import
commands
can be found
in
2nde [catalog]

from PROGRAM import *

Keyword

Syntax: from PROGRAM import *

Description: Used to import a program. Imports the public attributes of a Python module into the current namespace.

Shell [Tools]
A:from
PROGRAM
import *

2nde [catalog]

from math import *

Keyword

Syntax: from math import *

Description: Used to import all functions and constants from the math module.

math Modul
1:math…
1:from math
import *

[Fns..] > Modul
1:math…
1:from math
import *

2nde [catalog]

from random import *

Keyword

Syntax: from random import *

Description: Used to import all functions from the random module.

math Modul
2:random…
1:from
random
import *

[Fns..] >
Modul
2:random…
1:from
random
import *

2nde [catalog]

G

global

Keyword 2nde [catalog]

Description: Use global to create global variables inside a function.

See CircuitPython documentation for more details.

if :

See if..elif..else.. for details.

2nde [catalog]

[Fns…] > Ctl

1:if..

2:if..else..

3:if..elif..else

9:elif :

0:else:

if..elif..else..

Keyword

2nde [catalog]

Syntax: ••Gray indent identifiers automatically provided in the Python App for ease of use.

if :

••

elif :

••

else:

[Fns…] > Ctl

1:if..

2:if..else..

3:if..elif..else

9:elif :

0:else:

Description: if..elif..else is a conditional statement. The Editor provides automatic indents as gray dots to assist your correct programming indents.

Example: Create and run this program, say S01, from the Editor

```
def f(a):
••if a>0:
••••print(a)
••elif a==0:
••••print("zero")
••else:
••••a=-a
••••print(a)
```

Shell interaction

```
>>> # Shell Reinitialized
>>> # Running S01
>>>from S01 import *      #automatically pastes
>>>f(5)
5
>>>f(0)
zero
>>>f(-5)
5
```

if..else..

Keyword 2nde [catalog]

See if..elif..else.. for details.

[Fns…] > Ctl

1:if..

2:if..else..

3:if..elif..else

9:elif :

0:else:

import math

Keyword

Syntax: import math 2nde [catalog]

Description: The math module is accessed using this command. This instruction imports the public attributes of the "math" module within its own namespace.

import random

Keyword

Syntax: import random 2nde [catalog]

Description: The random module is accessed using this command. This instruction imports the public attributes of the "random" module within its own namespace.

in

Keyword 2nde [catalog]

Description: Use in to check if a value is in a sequence or to iterate a sequence in a for loop.

input()

Module : Built-in

2nde [catalog]

Syntax: input()

Description: Prompt for input

[Fns…] I/O
2:input()

Example:

```
>>>input("Name? ")
Name? Me
'Me'
```

Alternate Example:

```
Create Program A
len=float(input("len: "))
print(len)



Run Program A
>>> # Shell Reinitialized
>>> # Running A
>>>from A import *
len: 15        (enter 15)
15.0          (output float 15.0)
```

.insert(index,x)

Module : Built-in

2nde [listes] List
8:.insert
(index,x)

Syntax: listname.insert(index,x)

Description: The method insert() inserts an item x after index within a sequence.

2nde [catalog]

Example:

```
>>>listA = [2,4,6,8]
>>>listA.insert(3,15)
>>>print(listA)
[2,4,6,15,8]
```

[Fns…] > List
8:.insert
(index,x)

int()

Module : Built-in

2nde [catalog]

Syntax: int(x)

Description: Returns x as an integer object.

[Fns…] > Type 1:int()

Example:

```
>>>int(34.67)
34
>>>int(1234.56)
1234
```

is

Keyword

2nde [catalog]

Description: Use is to test if two objects are the same object.

lambda

Keyword

2nde [catalog]

Syntax: lambda arguments : expression

Description: Use lambda to define an anonymous function. See Python documentation for details.

len()

Module: Built-in

2nde [listes]
(above stats)
List
3:len()

Syntax: len(sequence)

Description: Returns the number of items in the argument. The argument may be a sequence or a collection. See Python documentation for more details.

2nde [catalog]

Example:

```
>>>mylist=[2,4,6,8,10]
>>>len(mylist)
5
```

[Fns…] > List
3:len()

list(sequence)

Module: Built-in

2nde [listes] (above stats) List 2:list(sequence)

Syntax: list(sequence)

Description: Mutable sequence of items of the save type.

list()" converts its argument into the "list" type. Like many other sequences, the elements of a list do not need to be of the same type.

2nde [catalog]

Example:

[Fns…] > List 2:list(sequence)

```
>>>mylist=[2,4,6,8]
>>>print(mylist)
[2,4,6,8]
```

Example:

```
>>>mylist=[2,4,6,8]
>>>print(mylist)
[2,4,6,8]
>>> list({1,2,"c", 7})
[7, 1, 2, 'c']
>>> list("foobar")
['f', 'o', 'o', 'b', 'a', 'r']
```

log(x,base)

Module: math

Syntax: log(x,base)

Description: log(x) with no base returns the natural logarithm x.

Example:

```
 >>>from math import *
>>>log(e)
1.0
>>>log(100,10)
2.0
>>>log(32,2)
5.0
```

2nde log for log(x,10)

2nde ln for log(x) (natural log)

math Modul 1:math… 6:log(x,base)

2nde [catalog]

[Fns…] > Modul 1:math… 6:log(x,base)

import commands can be found in 2nde [catalog]

math.function

Module: math

2nde [catalog]

Syntax: math.function

Description: Use after import math command to use a function in the math module.

Example:

```
>>>import math
>>>math.cos(0)
1.0
```

max()

Module: Built-in

2nde [listes] (above stats) List 4:max()

Syntax: max(sequence)

Description: Returns the maximum value in the sequence. See Python documentation for more information on max().

2nde [catalog]

Example:

```
>>>listA=[15,2,30,12,8]
>>>max(listA)
30
```

[Fns...] > List 4:max()

min()

Module: Built-in

2nde [listes] (above stats) List 5:min()

Syntax: min(sequence)

Description: Returns the minimum value in the sequence. See Python documentation for more information on min().

2nde [catalog]

Example:

```
>>>listA=[15,2,30,12,8]
>>>min(listA)
2
```

[Fns...] > List 5:min()

None

Keyword 2nde [catalog]

Description: None represents the absence of a value.

Example: [a A #]

```
>>> def f(x):
…       x
…
…
…
>>> print(f(2))
None
```

nonlocal

Keyword 2nde [catalog]

Syntax: nonlocal

Description: Use nonlocal to declare a variable is not local. See Python documentation for more details.

not

Keyword 2nde [tests] Ops 0:not

Syntax: not x

Description: Evaluates to True if x is False and False otherwise. Pastes with space before and after the keyword not. Edit as needed. [Fns…] > Ops 0:not

Example:

```
>>> not 2<5      #edit the space before not
False
>>>3<8 and not 2<5
False
```

2nde [catalog]

[a A #]

or

Keyword

2nde [tests] Ops 9:or

Syntax: x or y

[Fns…] > Ops 9:or

Description: May return True or False. Returns x if x evaluates as True and y otherwise. Pastes with space before and after or. Edit as needed.

Example:

2nde [catalog]

```
>>>2<5 or 5<10
True
>>>2<5 or 15<10
True
>>>12<5 or 15<10
False
>>> 3 or {}
3
>>> [] or {2}
{2}
```

[a A #]

pass

Keyword 2nde [catalog]

Description: Use pass in an empty function or class definition as a placeholder for future code as you build out your program. Empty definitions will not cause an error when program is executed.

pi

Module: math 2nde [π] (above trig)

Syntax: math.pi or pi if math module imported.

Description: Constant pi displays as shown below.

Example: [Fns…] > Modul 1:math… > Const 2:pi

```
>>>from math import *
>>>pi
3.141592653589793
```

Alternate Example:

```
>>>import math
>>>math.pi
3.141592653589793
```

pow(x,y)

Module: math

Syntax: pow(x,y)

Description: Returns x raised to the power y. Converts both x and y to float. See Python documentation for more information.

Use the built-in pow(x,y) function or ** for computing exact integer powers.

Example:

```
>>>from math import *
>>>pow(2,3)
>>>8.0
```

Example using: Built-in:

[Tools] > 6:New Shell

```
>>>pow(2,3)
8
>>>2**3
8
```

math Modul
1:math
5:pow(x,y)

2nde [catalog]

[Fns…] >
Modul 1:math
5:pow(x,y)

import commands can be found in 2nde [catalog]

print()

Module: Built-in

Syntax: print(argument)

Description: Displays argument as string.

Example:

```
>>>x=57.4
>>>print("my number is =", x)
my number is= 57.4
```

2nde [catalog]

[Fns…] > I/O
1:print()

radians()

Module: math

trig Trig
1:radians()

Syntax: radians(x)

Description: Converts angle x in degrees to radians.

2nde [catalog]

Example:

```
>>>from math import *
>>>radians(180.0)
3.141592653589793
>>>radians(90.0)
1.570796326794897
```

[Fns…] >
Modul
1:math… >
Trig
1:radians()

raise

Keyword

2nde [catalog]

Syntax: raise exception

Description: Use raise to raise a specified exception and stop your program.

randint(min,max)

Module: random

Syntax: randint(min,max)

Description: Returns a random integer between min and max.

Example:

```
>>>from random import *
>>>randint(10,20)
>>>15
```

Alternate Example:

```
>>>import random
>>>random.randint(200,450)
306
```

Results will vary given a random output.

[math] Modul
2:random
4:randint
(min,max)

[Fns…] >
Modul
2:random…
4:randint
(min,max)

[2nde] [catalog]

import
commands
can be found
in
[2nde] [catalog]

random()

Module: random

Syntax: random()

Description: Returns a floating point number from 0 to 1.0. This function takes no arguments.

Example:

```
>>>from random import *
>>>random()
0.5381466990230621
```

Alternate Example:

```
>>>import random
>>>random.random()
0.2695098437037318
```

Results will vary given a random output.

[math] Modul 2:random… Random 2:random()

[Fns…] > Modul 2:random… Random 2:random()

[2nde] [catalog]

import commands can be found in [2nde] [catalog]

random.function

Module: random

Syntax: random.function

Description: Use after import random to access a function in the random module.

Example:

```
>>>import random
>>>random.randint(1,15)
2
```

Results will vary given a random output.

[2nde] [catalog]

randrange(start,stop,step)

Module: random

Syntax: randrange(start,stop,step)

Description: Returns a random number from start to stop by step.

Example:

```
>>>from random import *
>>>randrange(10,50,2)
12
```

Alternate Example:

```
>>>import random
>>>random.randrange(10,50,2)
48
```

Results will vary given a random output.

math Modul 2:random… Random 6:randrange (start,stop,step)

math Modul 2:random… Random 6:randrange (start,stop,step)

2nde [catalog]

import commands can be found in 2nde[catalog]

range(start,stop,step)

Module: Built in

2nde [catalog]

Syntax: range(start,stop,step)

Description: Use range function to return a sequence of numbers. All arguments are optional. Start default is 0, step default is 1 and sequence ends at stop.

Example:

```
>>> x = range(2,10,3)
>>> for i in x
…       print(i)
…
…
2
5
8
```

.remove(x)

Module: Built-in

2nde [listes]
List
7:.remove(x)

Syntax: listname.remove(item)

Description: The method remove() removes the first instance of item from a sequence.

2nde [catalog]

Example:

```
>>>listA = [2,4,6,8,6]
>>>listA.remove(6)
>>>print(listA)
[2,4,8,6]
```

[Fns…] > List
7:.remove(x)

return

Module: Built-in

2nde [catalog]

Syntax: return expression

Description: A return statement defines the value produced by a function. Python functions return None by default. See also: def function():

[Fns…] > Func
1:def function():

Example:

[Fns…] > Func
2:return

```
>>>  def f(a,b):
…         return a*b
…
…
…
>>>       f(2,3)
6
```

.reverse()

Module: Built-in

2nde [catalog]

Syntax: listname.reverse()

Description: Reverses the order of items in a sequence.

Example:

```
>>>list1=[15,-32,4]
>>>list1.reverse()
>>>print(list1)
[4,-32,15]
```

round()

Module: Built in

2nde [catalog]

Syntax: round(number, digits)

Description: Use round function to return a floating point number rounded to the specified digits. Default digit is 0 and returns the nearest integer.

Example:

```
>>>round(23.12456)
23
>>>round(23.12456,3)
23.125
```

seed()

Module: random

Syntax: seed() or seed(x) where x is integer

Description: Initialize random number generator.

Example:

```
>>>from random import *
>>>seed(12)
>>>random()
0.9079708720366826
>>>seed(10)
>>>random()
0.9063990882481896
>>>seed(12)
>>>random()
0.9079708720366826
```

Results will vary given a random output.

[math] Modul 2:random… Random 7:seed()

[Fns…] > Modul 2:random… Random 7:seed()

[2nde] [catalog]

import commands can be found in [2nde] [catalog]

sin()

Module: math

Syntax: sin()

Description: Returns sine of x. Argument angle is in radians.

Example:

```
>>>from math import *
>>>sin(pi/2)
1.0
```

[trig] 3:sin()

[2nde] [catalog]

[Fns…] > Modul 1:math… > Trig 3:sin()

import commands can be found in [2nde] [catalog]

.sort()

Module: Built-in

Syntax: listname.sort()

Description: The method sorts a list in place. See Python documentation for more details.

Example:

```
>>>listA=[4,3,6,2,7,4,8,9,3,5,4,6]
>>>listA.sort()
>>>print(listA)      #listA updated to a sorted list
[2,3,3,4,4,4,5,6,6,7,8,9]
```

2nde [listes]
(above stats
List A:.sort()

2nde [catalog]

[Fns…] >
List
A:sort()

sorted()

Module: Built-in

Syntax: sorted(sequence)

Description: Returns a sorted list from sequence.

Example:

```
>>>listA=[4,3,6,2,7,4,8,9,3,5,4,6]
>>>sorted(listA)
[2,3,3,4,4,4,5,6,6,7,8,9]
>>>print(listA)      #listA did not change
[4,3,6,2,7,4,8,9,3,5,4,6]
```

2nde [listes]
(above stats)
List
0:sorted()

2nde [catalog]

[Fns…] > List
0:sorted()

sqrt()

Module: math

Syntax: sqrt(x)

Description: Returns square root of x.

Example:

```
>>>from math import *
>>>sqrt(25)
5.0
```

[math] Modul
1:math 3:sqrt()

[2nde] [catalog]

[Fns…] > Modul
1:math 3:sqrt()

import commands can be found in [2nde] [catalog].

str()

Module: Built-in

Syntax: str(argument)

Description: Converts "argument" to a string.

Example:

```
>>>x=2+3
>>>str(x)
'5'
```

[2nde] [catalog]

[Fns…]
> Type
3 :str()

sum()

Module: Built-in

Syntax: sum(sequence)

Description: Returns the sum of the items in a sequence.

Example:

```
>>>listA=[2,4,6,8,10]
>>>sum(listA)
30
```

[2nde] [listes]
(above [stats])
List
9:sum()

[2nde] [catalog]

[Fns…] > List
9:sum()

tan()

Module: math

Syntax: tan(x)

Description: Returns tangent of x. Angle argument is in radians.

Example:

```
>>>from math import *
>>>tan(pi/4)
1.0
```

[trig] 5:tan()

[Fns…] > Modul 1:math… > Trig 5:tan()

[2nde] [catalog]

import commands can be found in [2nde] [catalog]

True

Keyword

Description: Returns True when statement executed is True. "True" represents the true value of objects of type bool.

Example:

```
>>>64>=32
True
```

[2nde] [tests] (above [math])

[2nde] [catalog]

[Fns…] > Ops A:True

[a A #]

trunc()

Module: math

Syntax: trunc(x)

Description: Returns the real value x truncated to an integer.

Example:

```
>>>from math import *
>>>trunc(435.867)
435
```

math Modul
1:math…
0:trunc()

2nde [catalog]

[Fns…] >
Modul
1:math…
0:trunc()

import commands can be found in 2nde [catalog]

try:

Keyword

2nde [catalog]

Description: Use try code block to test the code block for errors. Also used with except and finally. See Python documentation for more details.

uniform(min,max)

Module: random

Syntax: uniform(min,max)

Description: Returns a random number x (float) such that min <= x <= max.

Example:

```
>>>from random import *
>>>uniform(0,1)
0.476118
>>>uniform(10,20)
16.2787
```

Results will vary given a random output.

math Modul 2:random… Random 3:uniform (min,max)

2nde [catalog]

[Fns…] > Modul 2:random… Random 3:uniform (min,max)

import commands can be found in 2nde [catalog]

while condition:

Keyword

Syntax: while condition:

Description: Executes the statements in the following code block until "condition" evaluates to False.

Example:

```
>>> x=5
>>> while x<8:
…        x=x+1
…        print(x)
…
…
6
7
8
```

[Fns…] Ctl 8:while condition:

2nde [catalog]

@

Operator

Description: Decorator – See general Python documentation for details.

alpha [θ] (above 3)

2nde [catalog]

<<

Operator

Syntax: x<<n

Description: Bitwise left shift by n bits.

2nde [catalog]

>>

Operator

Syntax: x>>n

Description: Bitwise right shift by n bits.

2nde [catalog]

|

Operator 2nde [catalog]

Syntax: x|y

Description: Bitwise or.

&

Operator 2nde [catalog]

Syntax: x&y

Description: Bitwise and.

^

Operator 2nde [catalog]

Syntax: x^y

Description: Bitwise exclusive or.

~

Operator 2nde [catalog]

Syntax: ~x

Description: Bitwise not; the bits of x inverted.

x<=y

Operator

Syntax: x<=y

Description: Comparison; x less than or equal to y.

Example:

```
>>>2<=5
True
>>>3<=0
False
```

math
1:math > Ops
7:x<=y

2nde [catalog]

[Fns…] > Ops
7:x<=y

[a A #]

x<y

Operator

Syntax: x<y

Description: Comparison; x strictly less than y.

Example:

```
>>>6<10
True
>>>12<-15
False
```

math
1:math > Ops
6:x<y

2nde [catalog]

[Fns…] > Ops
6:x<y

[a A #]

x>=y

Operator

Syntax: x>=y

Description: Comparison; x greater than or equal to y.

Example:

```
>>>35>=25
True
>>>14>=65
False
```

math
1:math > Ops
5:x>=y

2nde [catalog]

[Fns…] > Ops
5:x>=y

[a A #]

x>y

Operator

Syntax: x>y

Description: Comparison; x strictly greater than y.

Example:

```
>>>35>25
True
>>>14>65
False
```

math
1:math > Ops
4:x>y

2nde [catalog]

[Fns…] > Ops
4:x>y

[a A #]

x!=y

Operator

Syntax: x!=y

Description: Comparison; x not equal to y.

Example:

```
>>>35!=25
True
>>>14!=10+4
False
```

math
1:math > Ops
3:x!=y

2nde [catalog]

[Fns…] > Ops
3:x!=y

[a A #]

x==y

Operator

Syntax: x==y

Description: Comparison; x is equal to y.

Example:

```
>>>75==25+50
True
>>>1/3==0.333333
False
>>>1/3==0.3333333      #equal to stored Python value
True
```

math
1:math > Ops
2:x==y

2nde [catalog]

[Fns…] > Ops
2:x==y

[a A #]

x=y

Operator

Syntax: x=y

Description:y is stored in variable x

Example:

```
>>>A=5.0
>>>print(A)
5.0
>>>B=2**3       #Use [ ^ ] on keypad for **
>>>print(B)
8
```

sto→

math
1:math > Ops
1:x=y

2nde [catalog]

[Fns…] > Ops
1:x=y

[a A #]

\

Delimiter

Description: Backslash character.

2nde [catalog]

[a A #]

\t

Delimiter

Description: Tab space between strings or characters.

2nde [catalog]

\n

Delimiter

Description: New line to display string neatly on the screen.

2nde [catalog]

' '

Delimiter

Description: Two single quotes paste.

Example:

```
>>>eval('a+10')
17
```

2nde [mém]
(above +)

2nde [catalog]

[a A #]

" "

Delimiter

Description: Two double quotes paste.

Example:

```
>>>print("Ok")
```

alpha ["]
(above +)

2nde [catalog]

[a A #]

with

Keyword

Description: See Python documentation for more details.

2nde [catalog]

yield

Keyword 2nde [catalog]

Description: Use yield to end a function. Returns a generator. See Python documentation for more details.

Appendix

Selected Module Content for Python Adapter App v5.3.5

Selected Module Content for Python Adapter App v5.3.5

Built-ins	math	random	keywords
__name__	__name__	__name__	False
__build_class__ -- <function>	e -- 2.71828	seed -- <function>	None
__import__ -- <function>	pi -- 3.14159	getrandbits -- <function>	True
__repl_print__ -- <function>	sqrt -- <function>	randrange -- <function>	and
bool -- <class 'bool'>	pow -- <function>	randint -- <function>	as
bytes -- <class 'bytes'>	exp -- <function>	choice -- <function>	assert
bytearray -- <class 'bytearray'>	log -- <function>	random -- <function>	break
dict -- <class 'dict'>	cos -- <function>	uniform -- <function>	class
enumerate -- <class 'enumerate'>	sin -- <function>		continue
filter -- <class 'filter'>	tan -- <function>		def
float -- <class 'float'>	acos -- <function>		del
int -- <class 'int'>	asin -- <function>		elif
list -- <class 'list'>	atan -- <function>		else
map -- <class 'map'>	atan2 -- <function>		except
memoryview -- <class 'memoryview'>	ceil -- <function>		finally
object -- <class 'object'>	copysign -- <function>		for
property -- <class 'property'>	fabs -- <function>		from
range -- <class 'range'>	floor -- <function>		global
set -- <class 'set'>	fmod -- <function>		if
slice -- <class 'slice'>	frexp -- <function>		import

Built-ins	math	random	keywords
str -- <class 'str'>	ldexp -- <function>		in
super -- <class 'super'>	modf -- <function>		is
tuple -- <class 'tuple'>	isfinite -- <function>		lambda
type -- <class 'type'>	isinf -- <function>		nonlocal
zip -- <class 'zip'>	isnan -- <function>		not
classmethod -- <class 'classmethod'>	trunc -- <function>		or
staticmethod -- <class 'staticmethod'>	radians -- <function>		pass
Ellipsis -- Ellipsis	degrees -- <function>		raise
abs -- <function>			return
all -- <function>			try
any -- <function>			while
bin -- <function>			with
callable -- <function>			yield
chr -- <function>			
dir -- <function>			
divmod -- <function>			
eval -- <function>			
exec -- <function>			
getattr -- <function>			
setattr -- <function>			
globals -- <function>			
hasattr -- <function>			
hash -- <function>			

Built-ins	math	random	keywords
help -- <function>			
hex -- <function>			
id -- <function>			
input -- <function>			
isinstance -- <function>			
issubclass -- <function>			
iter -- <function>			
len -- <function>			
locals -- <function>			
max -- <function>			
min -- <function>			
next -- <function>			
oct -- <function>			
ord -- <function>			
pow -- <function>			
print -- <function>			
repr -- <function>			
round -- <function>			
sorted -- <function>			
sum -- <function>			
BaseException -- <class 'BaseException'>			
ArithmeticError -- <class 'ArithmeticError'>			
AssertionError -- <class 'AssertionError'>			

Built-ins	math	random	keywords
AttributeError -- <class 'AttributeError'>			
EOFError -- <class 'EOFError'>			
Exception -- <class 'Exception'>			
GeneratorExit -- <class 'GeneratorExit'>			
ImportError -- <class 'ImportError'>			
IndentationError -- <class 'IndentationError'>			
IndexError -- <class 'IndexError'>			
KeyboardInterrupt -- <class 'KeyboardInterrupt'>			
ReloadException -- <class 'ReloadException'>			
KeyError -- <class 'KeyError'>			
LookupError -- <class 'LookupError'>			
MemoryError -- <class 'MemoryError'>			
NameError -- <class 'NameError'>			
NotImplementedError -- <class 'NotImplementedError'>			
OSError -- <class 'OSError'>			
OverflowError -- <class 'OverflowError'>			
RuntimeError -- <class 'RuntimeError'>			
StopIteration -- <class 'StopIteration'>			
SyntaxError -- <class 'SyntaxError'>			
SystemExit -- <class 'SystemExit'>			
TypeError -- <class 'TypeError'>			

Built-ins	math	random	keywords
UnicodeError -- <class 'UnicodeError'>			
ValueError -- <class 'ValueError'>			
ZeroDivisionError -- <class 'ZeroDivisionError'>			
help -- <function>			
input -- <function>			
open -- <function>			

.

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.