

# Chapter 17 Python (fx-CG50, fx-CG50 AU only)

The **Python** mode provides a runtime environment for the Python programming language. You can use the **Python** mode to create, save, edit, and run Python files.

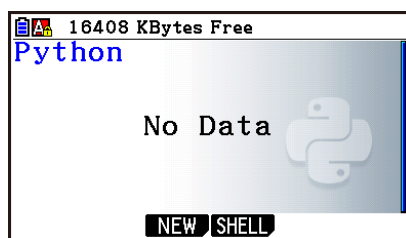
## **Important!**

- The **Python** mode supports a version of MicroPython Version 1.9.4, which has been adapted to run on this calculator. Note that generally, MicroPython is different from the Python that runs on a computer. Also, the **Python** mode does not support all of the functions, commands, modules, and libraries of MicroPython.
- MicroPython is an open-source project. For license information, refer to “MicroPython license information” (page 7-1).
- The **Python** mode performs executions using the MicroPython processing system. Because of this, calculation results and other data produced by this mode may differ from execution results of other function modes.
- Python is a registered trademark of the Python Software Foundation. Trademark (™) and registered trademark (®) symbols are not used in this manual.

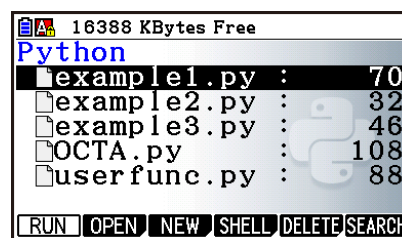
## 1. Python Mode Overview

### ■ File List Screen

The first thing that appears when you select the **Python** mode on the main menu is the file list screen.



When no py file\* or folder is in memory



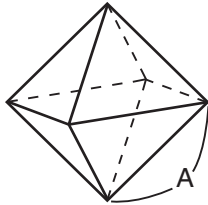
When there are py files or folders in memory

\* In this manual, a file created in the **Python** mode (file name extension py) is referred to as a “py file”.

## ■ Flow from py File Creation to Running the File

The example below explains the operation flow from creation of a new py file up to running it.

**Example:** To create a py file that obtains the surface area and volume of a regular octahedron and to run it to calculate the surface area and volume when the length of one side is 10. The file name is OCTA.



You can obtain the surface area (S) and volume (V) of a regular octahedron when the length of one side (A) is known using the formulas below.

$$S = 2\sqrt{3} A^2, \quad V = \frac{\sqrt{2}}{3} A^3$$

Here we will write a program that, when the py file is run, prompts for input of A, which is then used in the above formulas to output the calculation results. In this manual, a program written in Python (and stored in a py file) is called a “py script”.

For this example, we will input a py script like the one shown in the screen shot to the right.

A screenshot of a text editor window titled 'OCTA.py' with a line counter '001/006'. The code is as follows:

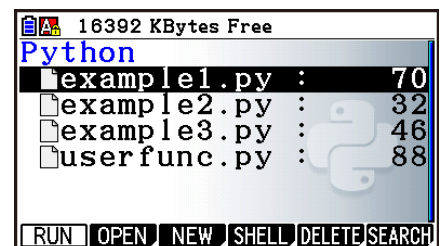
```
import math
A=int(input("A= "))
S=2*math.sqrt(3)*A**2
V=math.sqrt(2)/3*A**3
print("S=",S)
print("V=",V)
```

At the bottom, there are menu buttons: FILE, RUN, SYMBOL, CHAR, A↔a, and a play button.

### Procedure

1. From the main menu, enter the **Python** mode.

- This displays the file list screen.
- File names are listed in alphabetical order.
- The values on the right side of the file list indicate the number of bytes used by each py file.



2. Create a new file and register a file name\*.

Here we will use the procedure below to create a new py file named “OCTA”.

**[F3] (NEW) [F5] (A↔a) [9] (O) [In] (C) [÷] (T) [X,θ,T] (A) [EXE]**

- This displays the script editor screen.

#### \* File Names

- You can input up to eight letters (eight bytes) for a file name.
- A file name can be made up of single-byte alphanumeric characters (A to Z, a to z, 0 to 9). This calculator does not distinguish between uppercase and lower case letters.

### Important!

Note that a file with a name that starts with a numeral or a name that is a Python reserved word will not run.

3. Perform the key operations below to input each line of the py script.

- You can use the **Python** mode Catalog Function (page 17-9) for more efficient input of functions and commands. In the key operations below, text strings that are underlined and included in parentheses indicate function and command names input with the Catalog Function.

Perform this key operation:	To input this:
<u>SHIFT</u> <u>4</u> (CATALOG) <u>F6</u> (CAT) <u>3</u> (math) <u>(</u> <u>(</u> <u>import</u> <u>math</u> ) <u>EXE</u>	import math
<u>ALPHA</u> <u>F5</u> (A $\leftrightarrow$ a) <u>X,θ,T</u> (A) <u>SHIFT</u> <u>=</u> ( <u>=</u> ) <u>SHIFT</u> <u>4</u> (CATALOG) <u>F6</u> (CAT) <u>2</u> (Built-in) <u>(</u> <u>(</u> <u>int</u> ) <u>EXE</u> <u>SHIFT</u> <u>4</u> (CATALOG) <u>▲</u> <u>▲</u> (input()) <u>EXE</u> <u>ALPHA</u> <u>x10<sup>-1</sup></u> (") <u>ALPHA</u> <u>F5</u> (A $\leftrightarrow$ a) <u>X,θ,T</u> (A) <u>SHIFT</u> <u>=</u> ( <u>=</u> ) <u>ALPHA</u> <u>=</u> (SPACE) <u>ALPHA</u> <u>x10<sup>-1</sup></u> (") <u>▶</u> <u>▶</u> <u>EXE</u>	A=int(input("A= "))
<u>ALPHA</u> <u>F5</u> (A $\leftrightarrow$ a) <u>X</u> (S) <u>SHIFT</u> <u>=</u> ( <u>=</u> ) <u>2</u> <u>X</u> <u>SHIFT</u> <u>4</u> (CATALOG) <u>F6</u> (CAT) <u>3</u> (math) <u>7</u> (M)(math.) <u>EXE</u> <u>SHIFT</u> <u>x<sup>2</sup></u> ( <u>√</u> ) <u>3</u> <u>▶</u> <u>X</u> <u>ALPHA</u> <u>F5</u> (A $\leftrightarrow$ a) <u>X,θ,T</u> (A) <u>x<sup>2</sup></u> <u>EXE</u>	S=2*math.sqrt(3)*A**2
<u>ALPHA</u> <u>F5</u> (A $\leftrightarrow$ a) <u>2</u> (V) <u>SHIFT</u> <u>=</u> ( <u>=</u> ) <u>SHIFT</u> <u>4</u> (CATALOG) <u>(math.)</u> <u>EXE</u> <u>SHIFT</u> <u>x<sup>2</sup></u> ( <u>√</u> ) <u>2</u> <u>▶</u> <u>÷</u> <u>3</u> <u>X</u> <u>ALPHA</u> <u>F5</u> (A $\leftrightarrow$ a) <u>X,θ,T</u> (A) <u>▲</u> <u>3</u> <u>EXE</u>	V=math.sqrt(2)/3*A**3
<u>SHIFT</u> <u>4</u> (CATALOG) <u>F6</u> (CAT) <u>2</u> (Built-in) <u>4</u> (P) <u>6</u> (R)(print()) <u>EXE</u> <u>ALPHA</u> <u>x10<sup>-1</sup></u> (") <u>ALPHA</u> <u>F5</u> (A $\leftrightarrow$ a) <u>X</u> (S) <u>SHIFT</u> <u>=</u> ( <u>=</u> ) <u>ALPHA</u> <u>x10<sup>-1</sup></u> (") <u>▶</u> <u>ALPHA</u> <u>F5</u> (A $\leftrightarrow$ a) <u>X</u> (S) <u>▶</u> <u>EXE</u>	print("S=",S)
<u>SHIFT</u> <u>4</u> (CATALOG) <u>(print())</u> <u>EXE</u> <u>ALPHA</u> <u>x10<sup>-1</sup></u> (") <u>ALPHA</u> <u>F5</u> (A $\leftrightarrow$ a) <u>2</u> (V) <u>SHIFT</u> <u>=</u> ( <u>=</u> ) <u>ALPHA</u> <u>x10<sup>-1</sup></u> (") <u>▶</u> <u>ALPHA</u> <u>F5</u> (A $\leftrightarrow$ a) <u>2</u> (V)	print("V=",V)

4. Perform the key operation below to run the currently displayed py script.

F2 (RUN) F1 (Yes)

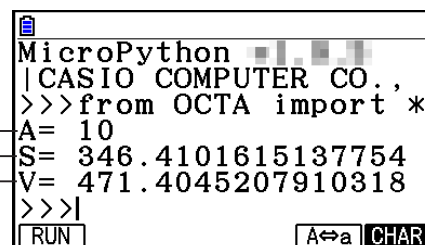
(Saves the script to a file before running it.)

The operations below are performed after the script is running.

1 0 (Inputs the value of A)

EXE

Value input for A —  
Execution result (S value) —  
Execution result (V value) —



- Following the operation above, you can re-run the same py script by performing the operation below.

- Press EXIT to return to the script editor screen.
- Press F2 (RUN).

---

## ■ SHELL Screen

Pressing **[F2]** (RUN) in step 4 of the procedure above starts up the **Python** mode SHELL, which can be used for running py scripts. The screen that appears at this time is called the “SHELL screen”. The SHELL screen not only lets you run py scripts that have been saved as files, you can also directly input expressions and commands and execute them one line at a time. For details about SHELL, see “Using the SHELL” (page 17-14).

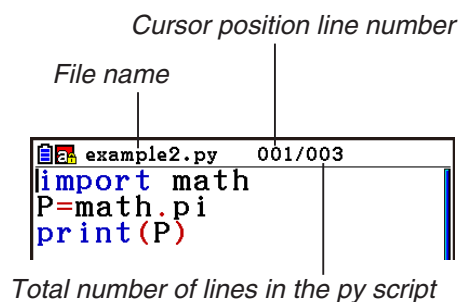
- If a py script does not work normally due to input error, running the script in step 4 will display an error message in red letters. Press **[EXIT]** to return to the script editor screen from the SHELL screen. For information about how to correct a py script, see “Editing a py File” (page 17-24).

---

## ■ Script Editor Screen

You can use the script editor screen that appears in step 2 above to input up to 300 lines, each of which can contain up to 255 characters.

The editor screen status bar shows the name of the currently open py file, the total number of lines in the py script, and line number of the current cursor position.



For information about how to open a py file and check its contents, and how to debug a py script and edit it, see “Editing a py File” (page 17-24).

## 2. Python Function Menu

---

### ■ File List Screen Function Menu

If there are no py files in memory, only the {NEW} and {SHELL} options will be available in the menu below.

- **{RUN}/{OPEN}** ... runs or opens a saved py file for editing
- **{NEW}** ... displays a file name registration screen for creating a new py file
- **{SHELL}** ... runs SHELL and displays the SHELL screen
- **{DELETE}** ... deletes the specified py file
- **{SEARCH}** ... searches for a file name

---

## ■ Function Menu for Registering a Name for a New py File

- {A⇌a} ... toggles between upper-case and lower-case input

---

## ■ Script Editor Screen Function Menu

- {FILE}
  - {SAVE} ... overwrites the currently open py file
  - {SAVE•AS} ... saves the currently open py file under a different name
- {RUN} ... displays the SHELL screen and runs the currently displayed py script
- {SYMBOL} ... displays a symbol input function menu
- {CHAR} ... displays an alphanumeric character, symbol, and operator input menu
- {A⇌a} ... toggles between upper-case and lower-case input
- {COMMAND} ... displays a conditional branch and loop command menu  
See “Using the Function Menu to Input Commands (Conditional Branches or Loops) as Statement Blocks” (page 17-8).
- {OPERAT} ... displays an operator (= != > < % | ^ & ~) input menu
- {JUMP} ... displays a line jump function menu
  - {TOP} ... jumps to the top line of a py script
  - {BOTTOM} ... jumps to the bottom line of a py script
  - {LINE} ... displays a line specification dialog box and jumps to the specified line of a py script
- {SEARCH} ... searches for the specified string

---

## ■ SHELL Screen Function Menu

- {RUN} ... runs the expression or command input in the last line (prompt line) of the SHELL screen
- {A⇌a} ... toggles between upper-case and lower-case input
- {CHAR} ... displays an alphanumeric character, symbol, and operator input menu

### 3. Inputting Text and Commands

There are three ways to input text and commands in the **Python** mode.

- Using the keyboard to input alpha characters, symbols, and functions (See the procedure below.)
- Function menu input
  - Alphanumeric character, symbol, and operator input (page 17-7)
  - Conditional branch command and loop command input (page 17-8)
- Using the catalog (function or command list) to select an item and input it (page 17-9)

#### ■ Using the Keyboard to Directly Input Commands

From the script editor screen or SHELL screen, you can use the calculator's keyboard to input numbers, alpha characters, and the functions ( $\sqrt{\quad}$ , log, etc.) assigned to each key.

#### • Using Keys to Input Numbers, Operators, Parentheses, and Functions

The table below shows what is input (number, operator, parentheses, or function) when you press a key, or press **SHIFT** and then a key.

Performing this key operation:	Inputs this:
<b>0</b> to <b>9</b>	0 to 9
<b>x<sup>2</sup></b>	**2
<b>^</b>	**
<b>X,θ,T</b>	x
<b>log</b>	log10()
<b>ln</b>	log()
<b>sin</b>	sin()
<b>cos</b>	cos()
<b>tan</b>	tan()
<b>(</b>	(
<b>)</b>	)
<b>.</b>	.
<b>,</b>	,
<b>×</b>	*
<b>÷</b>	/

Performing this key operation:	Inputs this:
<b>+</b>	+
<b>-</b>	-
<b>x10<sup>x</sup></b>	e
<b>SHIFT</b> <b>x<sup>2</sup></b> ( $\sqrt{\quad}$ )	sqrt()
<b>SHIFT</b> <b>ln</b> ( $e^x$ )	exp()
<b>SHIFT</b> <b>sin</b> ( $\sin^{-1}$ )	asin()
<b>SHIFT</b> <b>cos</b> ( $\cos^{-1}$ )	acos()
<b>SHIFT</b> <b>tan</b> ( $\tan^{-1}$ )	atan()
<b>SHIFT</b> <b>)</b> ( $x^{-1}$ )	** -1
<b>SHIFT</b> <b>×</b> ( { )	{
<b>SHIFT</b> <b>÷</b> ( } )	}
<b>SHIFT</b> <b>+</b> ( [ )	[
<b>SHIFT</b> <b>-</b> ( ] )	]
<b>SHIFT</b> <b>0</b> (i)	1j
<b>SHIFT</b> <b>.</b> (=)	=
<b>SHIFT</b> <b>x10<sup>x</sup></b> ( $\pi$ )	pi

## Important!

Among the text strings above that are input using key operations,  $\log()$  and other functions that are followed by parentheses,  $e$  (base of a natural logarithm), and  $\pi$  are math module functions. To use these functions, you first need to import the math module.\* For details, see “Command Categories” (page 17-10) and “Operation Example: To use math module functions” (page 17-13).

\* If you use *import* instead of *from* to input the module, you need to append “math.” before each function you use. See “Using Modules (*import*)” (page 17-12 for more information).

---

### • Alphabet Key Input

The first input immediately after **[ALPHA]** is pressed or if input was put into alpha lock by pressing **[SHIFT] [ALPHA]** (page 1-2), pressing a key will input the character marked in red on a key, a space or quotation marks (").

- Selecting {NEW} or {OPEN} from the file list screen will display the script editor screen and automatically puts input into lowercase alpha lock.

---

### • Auto Indent when a Newline is Input

From the **Python** mode script editor screen, pressing **[EXE]** will input a newline.

- Pressing **[EXE]** after a line that ends with a colon (:) automatically indents the new line two spaces more than the line above it (Auto Indent).
- Pressing **[EXE]** while the cursor is located in an indented line will indent the line following the newline by the same amount as the indented line above it.
- To perform a newline without indenting the new line, press **[SHIFT] [EXE]**.

Newline codes are not displayed in the **Python** mode.

---

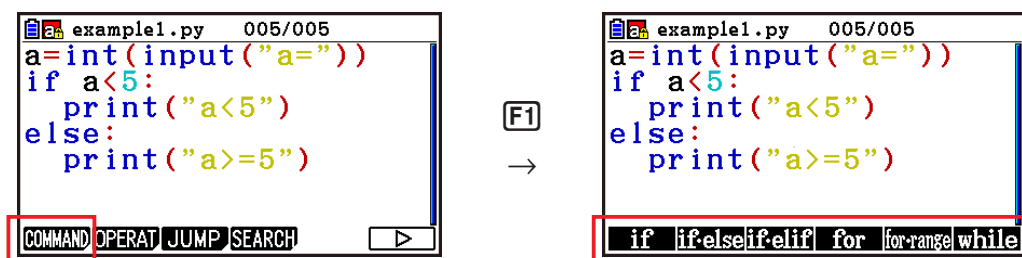
## ■ Using the Function Menu to Input Text (Alphanumeric Characters, Symbols, Operators)

Use the function menus shown in the table below to input alphanumeric characters, symbols, or operators.

Key Operation		Inputtable Characters (Alphanumeric Characters, Symbols, Operators)
Script Editor Screen	SHELL Screen	
<b>[F3]</b> (SYMBOL) (symbols)	—	, ( ) [ ] : ; # ' " \ _
<b>[F4]</b> (CHAR) (alphanumeric characters, symbols, operators)	<b>[F6]</b> (CHAR) (alphanumeric characters, symbols, operators)	! " # \$ % & ' ( ) * + , - . / 0 1 2 3 4 5 6 7 8 9 : ; < = > ? @ A B C D E F G H I J K L M N O P Q R S T U V W X Y Z [ \ ] ^ _ ` a b c d e f g h i j k l m n o p q r s t u v w x y z {   } ~
<b>[F6]</b> (▷) <b>[F2]</b> (OPERAT) (operators)	—	= != > < %   ^ & ~

## ■ Using the Function Menu to Input Commands (Conditional Branches or Loops) as Statement Blocks

From the script editor screen, you can use the function menu {COMMAND} menu to input conditional branch command and loop command statement blocks.



Perform this key operation:	To input this statement block:*	Perform this key operation:	To input this statement block:*
<b>[F6] (▷) [F1] (COMMAND)</b> <b>[F1] (if)</b>	if□l: □□	<b>[F6] (▷) [F1] (COMMAND)</b> <b>[F4] (for)</b>	for□i□in□l: □□
<b>[F6] (▷) [F1] (COMMAND)</b> <b>[F2] (if·else)</b>	if□l: □□ else: □□	<b>[F6] (▷) [F1] (COMMAND)</b> <b>[F5] (for·range)</b>	for□i□in□range(l): □□
<b>[F6] (▷) [F1] (COMMAND)</b> <b>[F3] (if·elif)</b>	if□l: □□ elif: □□ else: □□	<b>[F6] (▷) [F1] (COMMAND)</b> <b>[F6] (while)</b>	while□l: □□

\* The box (□) symbols in the above tables represent blank spaces. Box symbols do not appear on the display. The vertical lines (l) are cursor locations immediately after input. The vertical line (l) character is not inserted.

- In addition to the six statement blocks above, you can also use the catalog (page 17-9) to input the statement blocks shown below.
  - **for:range(,)**
  - **for:range(,,)**
  - **if·and:else**
  - **if·or:else**
  - **def:return**
- The SHELL screen allows one-line input only, so block statement input is not allowed. From the SHELL screen, selecting a menu that inputs statement blocks will input the first line of the block only.

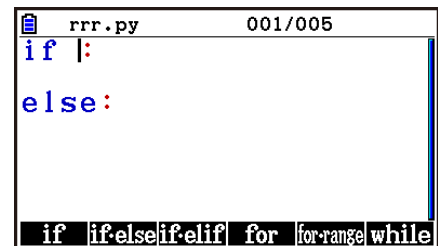


---

### • Example: To input an if...else statement

1. On the script editor screen, move the cursor to the line where you want to input the statement block and then press **F6**(▷) **F1**(COMMAND) **F2**(if·else).

- This inputs the if...else statement block, with the cursor positioned for input of the if condition.
- Lines 2 and 4 are indented two spaces automatically.



```
rrr.py 001/005
if |:
else:

```

if if-else if-elif for for-range while

---

## ■ Inputting a Command from the Catalog (Catalog Function)

The catalog is a list of functions and commands. You can perform input by displaying the catalog screen and then selecting the desired function or command. This operation is possible on both the script editor screen and the SHELL screen.\*

\* Only when the cursor is in the prompt line.

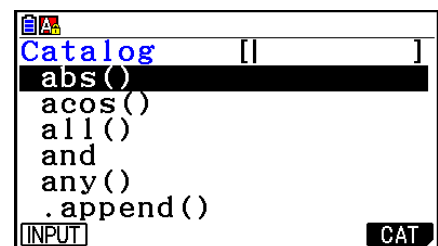
In the operations in this section, functions, commands, and other items that can be input from the catalog are collectively referred to as “commands”.

---

### • To input a command from the catalog

1. On the script editor screen or SHELL screen press **SHIFT** **4** (CATALOG).

- This displays the catalog command list screen.
- If you want to select a command on this screen for input, go to step 4 of this procedure. If you want to select a category, go to step 2 of this procedure.

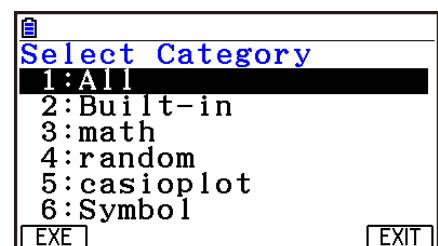


```
Catalog [ | ]
abs()
acos()
all()
and
any()
.append()
INPUT CAT

```

2. Press **F6**(CAT).

- This displays the category list.
- For detailed information about each category, see “Command Categories” (page 17-10).



```
Select Category
1:All
2:Built-in
3:math
4:random
5:casio plot
6:Symbol
EXE EXIT

```

3. Press a number key (from **1** to **6**) that corresponds to the category you want to select. Or you could use **▲** and **▼** to move the highlighting to the category and then press **EXE**.

- This returns to the command list screen, which will now show only the commands inside the category you selected.

4. Use  $\blacktriangle$  and  $\blacktriangledown$  to select the command you want to input.
  - You can scroll between screens by pressing  $\text{SHIFT}$   $\blacktriangle$  or  $\text{SHIFT}$   $\blacktriangledown$ .
  - You can also input a string up to eight characters long to search for commands that begin with the characters you input. For details about how to input these characters, see “Using the Catalog to Search for and Input a Command” (page 17-11).
5. After selecting the command you want to input, press  $\text{F1}$  (INPUT) or  $\text{EXE}$ .
  - When you enter the **Python** mode and display the catalog, the command that was selected the last time you displayed the catalog will appear first.

### Command Categories

The contents of each category of the **Python** mode catalog are described in the table below.

Category Name	Description
All	Shows a list of all functions and commands included in the <b>Python</b> mode catalog.
Built-in	Shows a list of Python built-in functions and commands. The functions and commands included in this category can be used without importing a module.*
math	Shows a list of commands that import the Python math module* (math functions), and the functions included in the math module.
random	Shows a list of commands that import the Python random module* (random number functions), and the functions included in the random module.
casioplot	Shows a list of commands that import the casioplot module,* and draw functions included in the casioplot module. The casioplot module is a CASIO-original module. For details, see “Using Draw Functions (casioplot Module)” (page 17-18).
Symbol	Shows a list of symbols and operators.

- \* For details about modules, see “Using Modules (*import*)” (page 17-12).
- Command search cannot be performed while the command list produced by selecting “Symbol” is displayed.
  - Unlike the catalogs of other modes (page 1-12), there is no command history function or QR Code function in the **Python** mode.

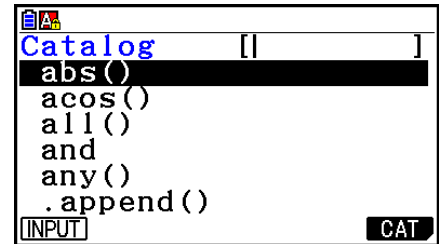
---

## • Using the Catalog to Search for and Input a Command

1. On the script editor screen or SHELL screen press

**[SHIFT] [4]** (CATALOG).

- This displays the catalog command list screen. The currently selected command is highlighted.
- As required, perform steps 2 and 3 under “To input a command from the catalog” (page 17-9) to select a category (besides “Symbol”).



2. Input some of the letters in the command name.

- You can input up to eight letters.
- With each letter you input, the highlighting will move to the first command name that matches.

3. Use **[▲]** and **[▼]** as required to move the highlighting to the command you want to input and then press **[F1]** (INPUT) or **[EXE]**.

---

## • Input Example 1: To use the Catalog Function to input “print()”

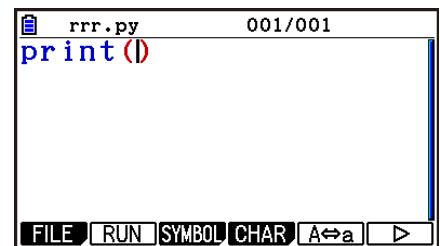
1. On the script editor screen, move the cursor to the line where you want to input the command and then press **[SHIFT] [4]** (CATALOG).

2. Press **[F6]** (CAT) to display the category screen, and then press **[2]** (Built-in).

3. Press **[4]** (P) **[6]** (R) to search for commands that begin with “pr”.

4. After confirming that “print()” is selected, press **[EXE]**.

- For an example that uses “print()”, see the examples at the beginning of this chapter and samples 1, 2, and 4 under “Sample Scripts” (page 17-29).



---

## • Input Example 2: To use the Catalog Function to input “import math”

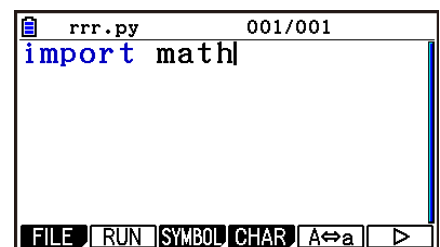
1. On the script editor screen, move the cursor to the line where you want to input the command and then press **[SHIFT] [4]** (CATALOG).

2. Press **[F6]** (CAT) to display the category screen, and then press **[3]** (math).

3. Press **[I]** searches for commands that begin with “i”.

4. After confirming that “import math” is selected, press **[EXE]**.

- For details about “import math”, see “Using Modules (*import*)” (page 17-12).



---

## ■ Using Modules (*import*)

In the **Python** mode, you can use the functions below in addition to Python built-in functions.

- Python standard math module and random module functions
- CASIO-original casioplot module functions (see page 17-18)

However, to use a function contained in a module you have to first import (*import*) the module.

<i>import</i> Syntax	Description
<code>import &lt;module name&gt;</code>	Imports the module (py file) specified by <module name>.
<code>from &lt;module name&gt; import *</code>	Imports all of the elements* included in the module specified by <module name>.
<code>from &lt;module name&gt; import &lt;element&gt; [, &lt;element&gt;]</code>	Imports the specified elements (functions, etc.) included in the module specified by <module name>.

\* An element with a name that starts with an underscore (\_) character cannot be imported.

- A single py file written by a py script is called a “module”. py files are imported using the same syntax as *import*.
- For an example of importing and using a py file, see “Sample 4: Importing a py File” in “Sample Scripts” (page 17-31).

### Notation Examples:

`import math` (Imports the math module.)

`from math import pi, sqrt` (From the math module, imports *pi* and *sqrt* only.)

- If you use *import* to import a module, you need to include “<module name>.” before a function in order to use that function. To use *pi* within the math module, for example, it would need to be written as “math.pi”.
- When using *from* to import a module, do not use the syntax below.

`<module name>.<function name>`

Executing an import operation with this syntax will cause an error.

## • Operation Example: To use math module functions

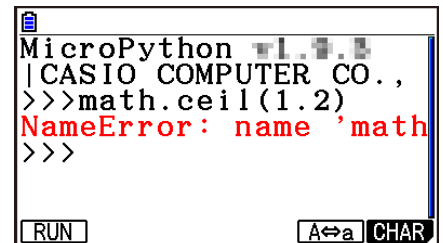
1. From the file list screen, press **[F4]** (SHELL).
2. Press **[SHIFT]** **[4]** (CATALOG) to display the catalog.
3. Press **[F6]** (CAT) to display the category screen, and then press **[3]** (math).
4. Perform the key operation sequence below.

**[7]** (M) **[EXE]** (Inputs “math.”)

**[SHIFT]** **[4]** (CATALOG) **[In]** (C) **[EXE]** (Inputs “ceil()”)

5. Press **[1]** **[.]** **[2]** **[EXE]**.

- The math module is not imported, so the “ceil()” math module function causes an error.

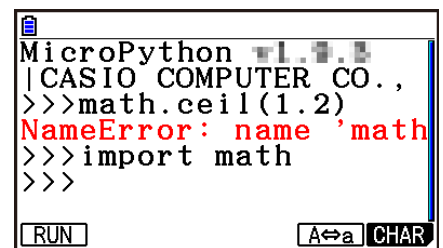


```
MicroPython 1.17.0
|CASIO COMPUTER CO.,
>>>math.ceil(1.2)
NameError: name 'math'
>>>
```

**[RUN]** **[A⇌a]** **[CHAR]**

6. Press **[SHIFT]** **[4]** (CATALOG) to display the catalog again, press **[C]** (I) to select “import math”, and then press **[EXE]**.
7. Press **[EXE]** again to execute “import math”.

- This imports the math module.



```
MicroPython 1.17.0
|CASIO COMPUTER CO.,
>>>math.ceil(1.2)
NameError: name 'math'
>>>import math
>>>
```

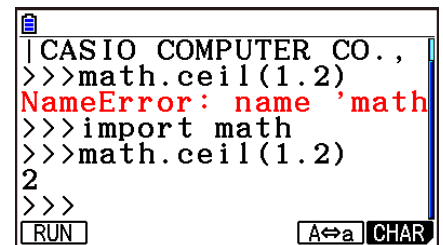
**[RUN]** **[A⇌a]** **[CHAR]**

8. Use **[▲]** to select the “math.ceil(1.2)” line you input in step 5 above, and then press **[EXE]**.

- This will copy the selected line to the prompt line.

9. Press **[EXE]**.

- This displays the execution result of “math.ceil(1.2)”.



```
|CASIO COMPUTER CO.,
>>>math.ceil(1.2)
NameError: name 'math'
>>>import math
>>>math.ceil(1.2)
2
>>>
```

**[RUN]** **[A⇌a]** **[CHAR]**

- The above operation can be performed by directly executing commands on the SHELL screen. For details about SHELL, see “Using the SHELL” (page 17-14).
- To use a math module, random module and/or casioplot module function in a py script, the applicable module import command must be written once in a line before the first use of the function.



---

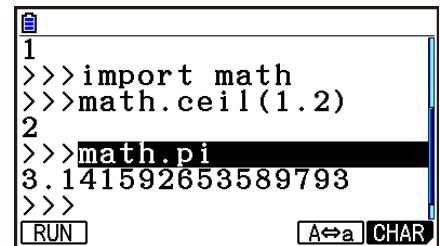
- **To execute a command from the SHELL screen**

See “Inputting a Command Directly on the SHELL Screen and Executing It” (page 17-16).

---

- **To scroll the SHELL screen vertically (to display history lines)**

Press ▲ or ▼. The currently selected history line is the one that is highlighted.

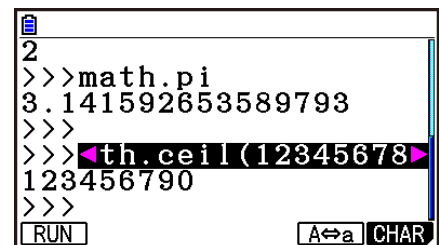


```
1
>>>import math
>>>math.ceil(1.2)
2
>>>math.pi
3.141592653589793
>>>
RUN A↔a CHAR
```

---

- **To scroll one line of the SHELL screen (history line or prompt line) horizontally**

1. Use ▲ and ▼ to move the highlighting to the line you want to scroll.
2. Press ◀ or ▶.
  - A history line that is too long to display completely is indicated by arrows (◀ and ▶) that show in which direction there are additional characters. Arrows (◀ and ▶) are not displayed in the prompt line even if its contents are too long to display completely.



```
2
>>>math.pi
3.141592653589793
>>>
>>>th.ceil(12345678)▶
123456790
>>>
RUN A↔a CHAR
```

---

- **To copy one SHELL screen history line to the prompt line**

Use ▲ and ▼ to move the highlighting to the line you want to copy, and then press [EXE]. For an actual operation example, see step 8 under “Operation Example: To use math module functions” (page 17-13).

---

- **To return to the file list screen from the SHELL screen**

Press [EXIT].

- If the currently displayed SHELL screen was arrived at by pressing [F2] (RUN) to run a py script from the script editor screen, the first press of [EXIT] will return to the script editor screen. In this case, press [EXIT] again to return to the file list screen.
- Changing from the SHELL screen to another **Python** mode screen will cause any variables used by the SHELL to be initialized. See “Initializing the SHELL” (page 17-17) for more information.

## ■ Inputting a Command Directly on the SHELL Screen and Executing It

You can input a single-line expression or command into the SHELL screen prompt line and execute it. The example operations below all start with the SHELL screen already displayed.

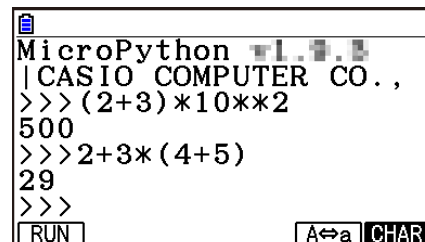
### • Operation Example 1: To perform simple arithmetic operations

$$(2+3) \times 10^2 = 500$$

( 2 + 3 ) × 1 0  $x^2$  EXE

$$2+3 \times (4+5) = 29$$

2 + 3 × ( 4 + 5 ) EXE



```
MicroPython 1.17.0
|CASIO COMPUTER CO.,
>>>(2+3)*10**2
500
>>>2+3*(4+5)
29
>>>
[RUN] A↔a CHAR
```

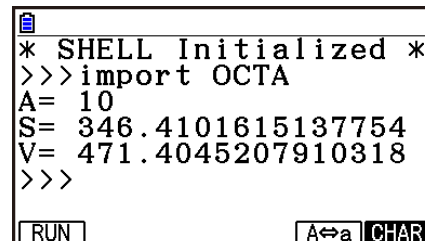
Note the important points below.

- Use the  $\boxed{-}$  key, not the  $\boxed{\leftarrow}$  key, to input a minus sign.
- Calculation accuracy in the **Python** mode is different from calculations performed in the **Run-Matrix** mode.

### • Operation Example 2: To recall and run a py file from the SHELL screen

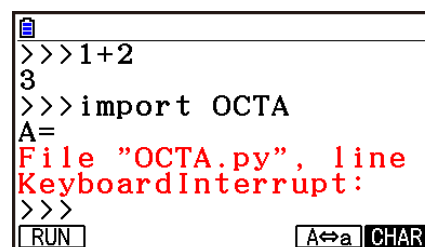
The operation below uses the “OCTA.py” file created using the example under “Flow from py File Creation to Running the File” (page 17-2). It assumes that the SHELL is already running. If you want to call the “OCTA.py” file from the SHELL, the SHELL must have been started up while the file list that contains the “OCTA.py” file was on the display.

SHIFT 4 (CATALOG) F6 (CAT) 2 (Built-in)  
( I ) 7 (M) 4 (P)(import) EXE  
SHIFT ALPHA F5 (A↔a) 9 (O) In (C)  $\div$  (T) X,θ,T (A) EXE  
ALPHA 1 0 (Input of value for A) EXE



```
* SHELL Initialized *
>>>import OCTA
A= 10
S= 346.4101615137754
V= 471.4045207910318
>>>
[RUN] A↔a CHAR
```

- To stop a running script, press  $\boxed{AC}$ .  
This will cause the message “KeyboardInterrupt:” to appear, with the cursor flashing in the bottom line (prompt line) of the display.



```
>>>1+2
3
>>>import OCTA
A=
File "OCTA.py", line
KeyboardInterrupt:
>>>
[RUN] A↔a CHAR
```



- ***input* Operation in the Python mode**

`input` is a built-in Python function that accepts user input while a py script is running.

<i>input</i> Syntax	Description
input([prompt text string])	<p>While a py script is running, <i>input</i> writes the [prompt text string] of the argument into the SHELL result output line, and stands by for user input.</p> <p>A string variable name or a character string enclosed in double quotation marks (") or single quotation marks (') can be specified for the [prompt text string].</p>

In the case of a string variable up to 16 characters long, all of the characters of the string variable name specified by *input* will be displayed as the prompt when the function is executed in the **Python** mode. In the case of a string variable longer than 16 characters, the first 15 characters of the string variable followed by the similarity symbol (~) will be displayed as the prompt.

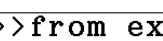
### *input* execution example

Prompt text string up to 16 characters long ("123?" input as the prompt text string.)	<pre>&gt;&gt;&gt;a=input("123?") 123? </pre>
Prompt text string longer than 16 characters ("12345678901234567" input as the prompt text string.)	<pre>&gt;&gt;&gt;a=input("12345678901234567") 123456789012345~ </pre>

## ■ Initializing the SHELL

Functions and variables that are defined, modules that are imported, and the results of other SHELL operations are stored in the SHELL heap area (temporary storage memory area) while the SHELL is running. Whenever the SHELL is exited (by going to a different **Python** mode screen), the SHELL heap area contents up to that point are cleared. This clearing of SHELL heap area contents is called “SHELL initialization”.

- When you restart the SHELL in the **Python** mode, the message “\* SHELL Initialized \*” will appear in the line above the bottom line (prompt line) of the SHELL screen.
- This message will appear only if you re-display the SHELL screen, but it won’t appear the first time you display the SHELL screen after entering the **Python** mode.



```
>>>from example1 impo
a=45
a>5
>>>a
45
* SHELL Initialized *
>>>|
```

- If the SHELL is restarted by running a py script from the file list screen or script editor screen, the SHELL will be initialized before the py script is run. Because of this, the SHELL screen will appear as shown in the screen shot below.

```

>>>f=60
>>>d+f*2
160
* SHELL Initialized *
>>>from example3 impo
6
>>>|
  
```

## 5. Using Draw Functions (casioplot Module)

The casioplot module is a CASIO-original module that includes draw functions for drawing pixels and character strings in the **Python** mode.

- However, to use the draw functions in the casioplot module you have to first import (*import*) the casioplot module. For details about the import procedure, see “Using Modules (*import*)” (page 17-12).
- The casioplot category of the Catalog Function (page 17-9) comes in handy when inputting the draw functions in the casioplot module. Input of commands that import the casioplot module is also simple.
- For details about the format and other information for each draw function, see “Draw Function Details” below.

### ■ Draw Function Details

This section provides a description, information about py script syntax and arguments, simple examples, and supplementary information about draw functions.

#### • Draw Function List

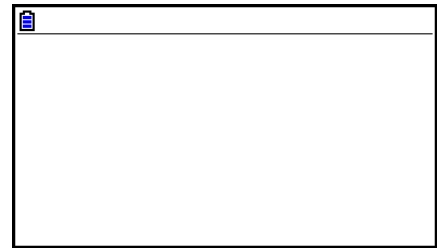
- Arguments enclosed in square brackets ([ ]) in a function syntax can be omitted. Omission of other arguments is not allowed.
- All function examples are executed from the SHELL screen.

**show\_screen()**

**Description:** Displays the drawing screen.

**Syntax:**        show\_screen()  
                  (No argument)

**Example:** To display the drawing screen  
from casioplot import \*  
show\_screen()



**Note:**

- The above shows an example in which a blank screen with nothing drawn on it is displayed when *show\_screen* is executed from the SHELL screen. If something is already drawn on the screen, that content will be shown when the function is executed.
- To exit the drawing screen and return to the SHELL screen, press **EXIT**, **AC**, or **SHIFT** **EXIT** (QUIT).
- For details about displaying the drawing screen, drawing screen refresh and clear timing, and other information, see “Drawing Screen” (page 17-23).

**clear\_screen()**

**Description:** Clears all draw contents from the drawing screen.

**Syntax:** clear\_screen()  
(No argument)

**Example:** To clear the character string drawn by the operation in the *draw\_string* example (page 17-22).

Following the draw operation, press **EXIT** to return to the SHELL screen and then execute the function below.

clear\_screen()  
show\_screen()



**Note:** This function is executed regardless of whether or not there are any draw contents on the drawing screen.

## set\_pixel()

**Description:** Draws a pixel of the specified color at the specified coordinates.

**Syntax:** `set_pixel(x, y[, color])`

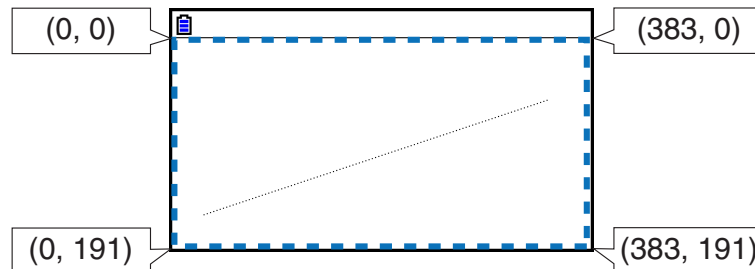
- *x* argument, *y* argument

Specifies the *x*- and *y*-coordinates of the pixel to be drawn.

Only int type values within the following ranges can be specified:

$0 \leq x \leq 383$ ,  $0 \leq y \leq 191$ .

The figure below shows the relationship between coordinate values and locations on the drawing screen.



- color argument

The color of the pixel to be drawn can be specified as a 256-gradation RGB value (0, 0, 0 to 255, 255, 255). For details about this argument, see “Draw Function color Argument” (page 17-22).

**Example:** To draw a black pixel at coordinates (10, 10) and display the drawing screen

```
from casioplot import *  
set_pixel(10,10,(0,0,0))  
show_screen()
```

Black pixel—



**Note:** If either the *x*- or *y*-coordinate value is outside of the allowable range, function execution will be ignored (nothing drawn, no error).

## get\_pixel()

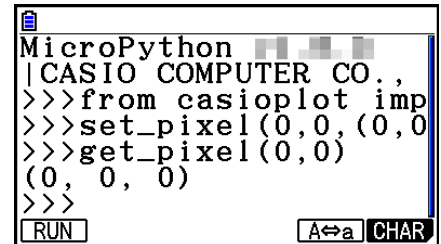
**Description:** Gets color information at the specified coordinates on the drawing screen.

**Syntax:** get\_pixel(x, y)

- *x* argument, *y* argument  
Specifies the *x*- and *y*-coordinates of the pixel whose color information is to be gotten. The range and type of value that can be specified are the same as the *x*-argument and *y*-argument of *set\_pixel* (page 17-20).

**Example:** To get color information (0, 0, 0) of coordinates (0, 0)

```
from casioplot import *  
set_pixel(0,0,(0,0,0))  
get_pixel(0,0)
```



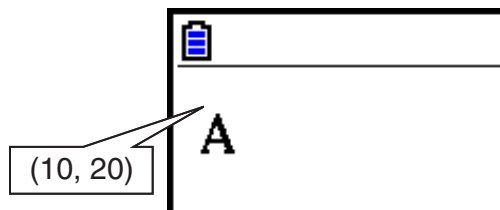
**Note:** If both the *x*- and *y*-coordinate values are within the allowable range, this function will return the RGB value for the color argument (page 17-22). If either the *x*- or *y*-coordinate value is outside of the allowable range, nothing is returned.

## draw\_string()

**Description:** Draws a character string of the specified color at the specified coordinates.

**Syntax:** draw\_string(x, y, s[, color[, size]])

- *x* argument, *y* argument  
Specifies the *x*- and *y*-coordinates of the first character of the character string to be drawn. The figure below shows the result when *x*=10, *y*=20 is specified for drawing the character “A”.

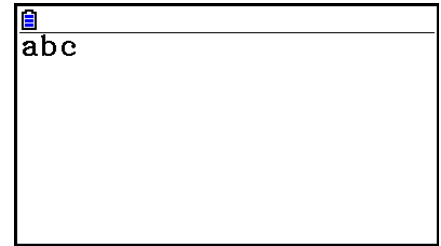


The range and type of value that can be specified are the same as the *x*-argument and *y*-argument of *set\_pixel* (page 17-20).

- *s* argument  
Specifies, as a str type, the character string to be drawn.
- color argument  
Specifies, as a 256-gradation RGB value (0, 0, 0 to 255, 255, 255), the character string to be drawn. For details about this argument, see “Draw Function color Argument” (page 17-22).
- size argument  
Specifies one of the following as the character size of the character string to be drawn: “large”, “medium”, “small”. “medium” is applied when this argument is omitted.

**Example:** To draw large size “abc” in black at coordinates (0, 0) and display the drawing screen

```
from casioplot import *  
draw_string(0,0,"abc",(0,0,0),"large")  
show_screen()
```



**Note:**

- If both the  $x$ - and  $y$ -coordinate values are within the allowable range, the drawn character string will be displayed within the drawing screen range, even if it partially runs off of the drawing screen.  
If either the  $x$ - or  $y$ -coordinate value is outside of the allowable range, function execution will be ignored (nothing drawn, no error).
- Only ASCII characters (“py Files Created and Saved with This Calculator”, page 17-27) can be displayed on the drawing screen.

---

## • Draw Function color Argument

The color argument of each function specifies the color of a pixel or character string.

### • Specifying a Color

Color is specified as a 256-gradation RGB value.

Example: (0,0,0)                      Black  
(255,255,255)                      White

### • Data Types

Data can be specified as a list[R,G,B] or tuple(R,G,B) type.

Only int type values within the range of 0 to 255 can be specified for each element (R, G, B).

### • Omitting Argument Specification

Omitting a color argument specification in a function causes (0,0,0) to be applied.

### color Argument Precautions

There may be some tonal variation between the specified RGB value and this product’s display screen. This is due to hardware limitations and does not indicate malfunction. Specified colors that cannot be reproduced are replaced by pseudo colors that can be displayed by this product.

---

## ■ Drawing Screen

The drawing screen is a special screen for drawing.

---

### ● Drawing Screen Refresh Timing

To refresh the drawing screen while it is displayed, execute the *show\_screen* function.

If you put *show\_screen* outside of a loop statement, executing the py script will cause only the final result to appear on the drawing screen. Putting *show\_screen* inside of a loop statement will display the result of each draw operation until the final result is reached.

#### (a) Example: *show\_screen* outside of a py script loop statement (display of the final result)

```
from casioplot import *  
for i in range(60):  
    set_pixel(i,i)  
show_screen()
```

#### (b) Example: *show\_screen* inside of a py script loop statement (display of each draw operation)

```
from casioplot import *  
for i in range(60):  
    set_pixel(i,i)  
    show_screen()
```

**Note:** The currently displayed drawing screen is refreshed after execution of the py script is complete.

---

### ● To return to the SHELL screen from the drawing screen

Press **EXIT**, **AC**, or **SHIFT** **EXIT** (QUIT).

**Note:** The display will change from the drawing screen to the SHELL screen if any one of the events below occurs during py script execution.

- Execution of *input*
  - Generation of an error
  - Pressing of **AC**
- 

### ● Drawing Screen Content Clear Timing

Drawing screen contents are cleared at the timing below.

- When *clear\_screen* is executed (page 17-19)
- When SHELL is initialized (page 17-17)

## 6. Editing a py File



---

### ■ Displaying and Editing a py File

You can use the procedure below to open a stored py file and display its contents in the script editor screen, where you can edit them, if you want.

---

#### ● To open a py file and display the script editor screen

1. From the main menu, enter the **Python** mode.
  2. On the file list screen that appears, use  and  to move the highlighting to the py file you want to open and then press **F2** (OPEN).
    - This opens the selected py file and displays the script editor screen.
    - Take care not to press **EXE** by mistake while the file list screen is displayed. Doing so will execute the py file and display the SHELL screen.
- 

#### ● To jump to the first line or the last line on the script editor screen

- To jump to the first line on the script editor screen, press **F6** (▷) **F3** (JUMP) **F1** (TOP).
  - To jump to the last line on the script editor screen, press **F6** (▷) **F3** (JUMP) **F2** (BOTTOM).
- 

#### ● To jump to a specific line number on the script editor screen

1. From the script editor screen, press **F6** (▷) **F3** (JUMP) **F3** (LINE).
  2. On the dialog box that appears, input the number of line to which you want to jump and then press **EXE**.
- 

#### ● To search for text on the script editor screen

1. From the script editor screen, press **F6** (▷) **F4** (SEARCH).
2. On the screen that appears, enter the character string you want to search for and then press **EXE**.
  - This starts searching from the top of the py script, and moves the cursor to the left of the first character of the first matching character string that is found. If there is no matching character string found, the message “Not Found” will appear. If this happens, press **EXIT**.
  - To resume the search using the same character string, press **F1** (SEARCH).
  - You will be able to resume a search operation only while “SEARCH” is shown for the **F1** function menu key, which indicates that there is at least one other matching character strings in the script. To cancel a search operation part way through, press **EXIT**. The search operation will end automatically if there are no more matches to the text string you specified.



---

- **To copy or cut a script editor screen text string and store it on the clipboard**

1. On the script editor screen, move the cursor to the beginning of the range you want to copy or cut and then press **[SHIFT] [8]** (CLIP).
2. Move the cursor to the end of the range you want to copy or cut.
  - This causes the selected range to become highlighted.
  - It makes no difference whether you select from the beginning to the end of a range, or from the end to the beginning.
3. Press **[F1]** (COPY) or **[F2]** (CUT).

---

- **To paste a character string that is on the clipboard**

1. On the script editor screen, move the cursor to the location where you want to paste the text string.
2. Press **[SHIFT] [9]** (PASTE).

---

## ■ Debugging a py Script

If a py file does not run as you expected it to, it may be due to a bug (error) in the py script.

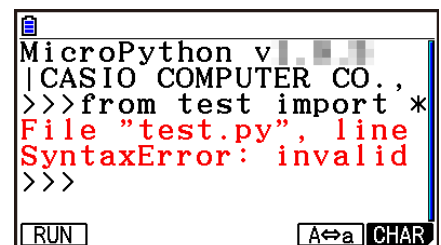
The symptoms below indicate that a py file needs to be debugged.

- When running a py file script produces an error message.
- When running a py file does not produce the desired operations or results.

---

- **Using Error Messages for Debugging**

If a red text error message appears on the SHELL screen when you run a py file, perform the steps below.



1. Use **[▲]** to move the highlighting to the error message line, and then use **[◀]** and **[▶]** to check the details of the error message.
2. Press **[EXIT]**.
  - This returns to the screen from which the py file was run (script editor screen or file list screen). Open the py file where the error occurred and check the contents of the line for which an error message was displayed. Make corrections as required.
  - Note that an error message may not necessarily identify the actual problem.
  - Note that an error message will also appear if there is a SHELL input problem, making it appear that there is an error in the py file. For example, if input data does not match the data type specified by *input*, etc. If you are unable to find a problem with the line for which an error message was displayed, check if SHELL input is correct.

The function menu {JUMP} function comes in handy when you need to jump to a particular line on the script editor screen. See “To jump to a specific line number on the script editor screen” (page 17-24).

---

### • Debugging Based on py File Running Results

If running a py file produces an unexpected result, check the entire content of the py file and make corrections as required.

## 7. File Management (Searching For and Deleting Files)

You can use the file list screen to search for saved py files by their file names, and to delete files.

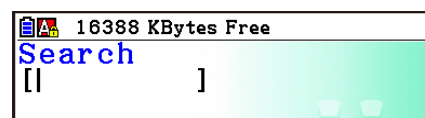
- py files you create in the **Python** mode are stored in the calculator's storage memory.
- In addition to file operations described in this section, you can also use Memory Manager to create folders and perform other folder operations. See “Chapter 11 Memory Manager” for more information.

---

### • To search for a py file by its file name

1. From the file list screen, press **[F6]** (SEARCH).

- This displays a search text input screen.



2. Enter part or the entire name of the file you want to find.

- You can input uppercase alphabetic characters only. Searches are not case-sensitive.
- File name characters are searched from left to right. This means that if you enter “IT” here, names such as ITXX, ITABC, and IT123 will be regarded as hits, but names like XXIT and ABITC will not be hits.

3. Press **[EXE]**.

- If a file name matches the character string you input in step 2, that file will be selected on the file list screen.
- The message “Not Found” will appear if a matching file name cannot be found. Press **[EXIT]** to close the message dialog box.

---

### • To delete a py file

1. From the file list screen, use **[▼]** and **[▲]** to highlight the file you want to delete, and then press **[F5]** (DELETE).

- This causes a delete confirmation message to appear.

2. Press **[F1]** (Yes) to delete or **[F6]** (No) to cancel the delete operation.

## 8. File Compatibility

py files can be shared between your calculator and a computer. A py file created with the calculator can be transferred to a computer for editing with a text editor or other software. A py file created on a computer can be transferred to and run on the calculator.

- py files you create in the **Python** mode are stored in the calculator's storage memory (with file name extension py).
- For information about the procedure for transferring files between the calculator and a computer, see "Performing Data Communication between the Calculator and a Personal Computer" (page 13-3).

---

### ■ py Files Created and Saved with This Calculator

The formats of py files created and saved with this calculator are shown below.

Character Code:     ASCII code

Characters used:    ASCII\*

Newline code:       CR+LF

Indent:             Spaces (two spaces for auto indent)

\* ASCII characters are those shown below.

A-Z   a-z   0-9   !   "   #   \$   %   &   '   (   )   \*   +   ,   -   .   /   :   ;   <   =   >   ?   @  
[   \   ]   ^   \_   `   {   |   }   ~   space

---

### ■ Precautions when Using an Externally Created py File on this Calculator

The restrictions below apply whenever you are trying to use the calculator's **Python** mode to display (file name or file content), edit, or run a py file that was transferred to the computer from an external source.

---

#### ● File Name Display

- Only py files whose file names consist of ASCII characters\* are displayed on the **Python** mode file list screen.
- Files with file names that include non-ASCII characters are not displayed.

\* However, the characters below are not allowed in file names.

\   /   :   \*   ?   "   <   >   |   .

- If the name of a py file transferred to storage memory from a computer or other source has a file name that is more than eight characters long, its name will abbreviated to eight characters when displayed on the storage memory information screen. (Example: AAAABBBBCC.py will become AAAABB~1.py.)

---

## • File Content Display and Editing

Opening a py file that satisfies conditions (A) and (B) below in the **Python** mode will produce a normal display of all of the file contents. A py file that shows contents can be displayed normally and edited in the **Python** mode.

(A) py file written in ASCII characters only and saved using UTF-8 or other ASCII-compatible codes

- If a file saved with character codes that are not compatible with ASCII, none of its contents will be displayed if you open it in the **Python** mode. All of the character will be replaced by spaces, or appear garbled.

(B) py file with up to 300 lines, each line containing up to 255 characters

- The contents of a py file that exceeds the number of characters and/or number of lines specified above cannot be displayed in the **Python** mode. Attempting to open such a file will display an “Invalid Data Size” error.
  - Though the contents of a py file that exceeds the number of characters and/or number of lines specified above cannot be displayed or edited in the **Python** mode, you may be able to run it. See “Running a py File” (page 17-28).
- All tab codes in a py file will be replaced by two spaces when the file is opened in the **Python** mode.
  - No type of newline codes (LF, CR, CR+LF) have an effect on **Python** mode display contents. All newline codes in a py file will be replaced by CR+LF (Windows standard newline code) when the file is opened in the **Python** mode. Before transferring a py file that was edited and saved in the **Python** mode to an external device for use on that device, replace its newline codes with the type that is appropriate for the environment where the file will be used.

---

## • Running a py File

You may be able to run a py file if that file is displayed on the file list screen in the **Python** mode. See “File Name Display” (page 17-27). Note the important points below.

- Running a py file that includes commands not supported by the calculator’s **Python** mode will result in an error.
- Using the **Python** mode to open a py file created on an external device will cause characters and newline codes to be replaced. For details, see “File Content Display and Editing” (page 17-28). Because of this, opening a py file in the **Python** mode, saving it, and running it, will change the content from the original py file, which may affect the running results.

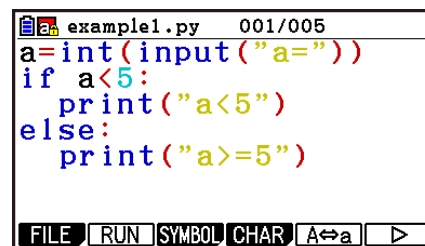
## 9. Sample Scripts

### Sample 1: Conditional Branching

#### Purpose

With conditional branching a condition is evaluated and then processing follows one of multiple paths in accordance with the evaluation result.

The example below is for a simple “if... else...” statement.



```
example1.py 001/005
a=int(input("a="))
if a<5:
    print("a<5")
else:
    print("a>=5")
```

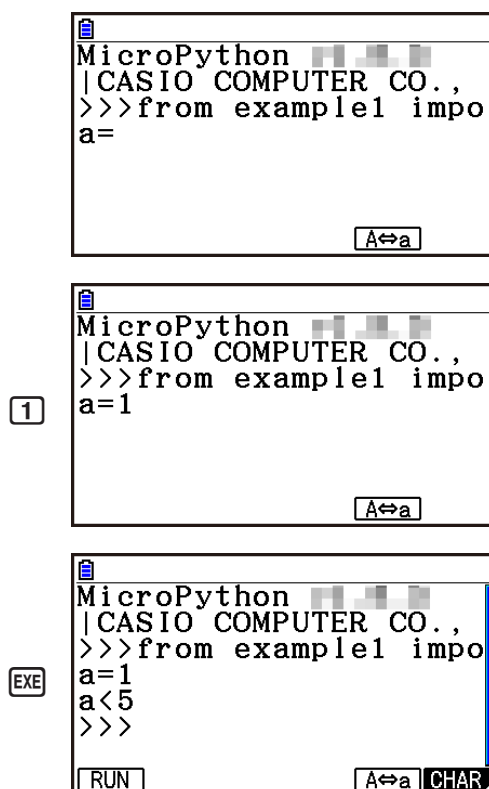
#### Description

a=int(input("a="))	Accepts user input while the py script is running. Input values are converted to integers and define variable a.
if a<5:	If the variable a is less than 5,
print("a<5")	outputs the text string a<5.
else:	Otherwise (if variable a is 5 or greater),
print("a>=5")	outputs the text string a>=5.

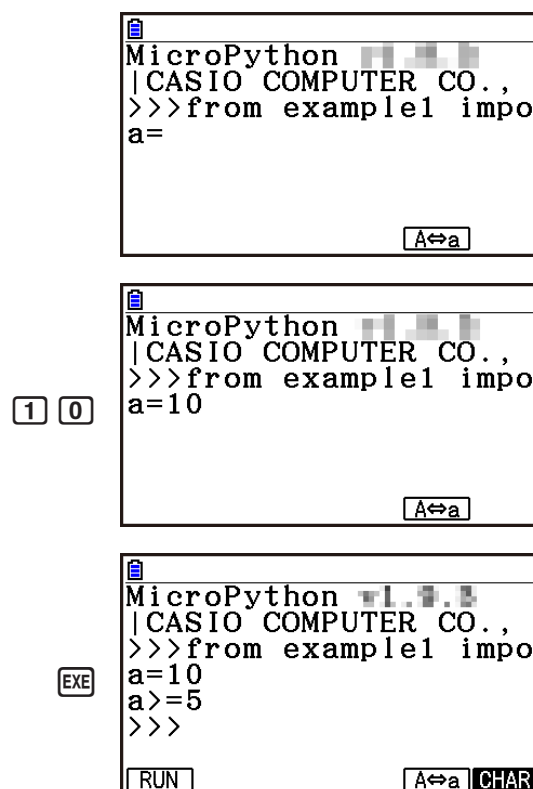
#### Execution Result (when a=1 and a=10 are input)

(1) If you input a = 1

(2) If you input a = 10



```
MicroPython
|CASIO COMPUTER CO.,
>>>from example1 impo
a=
[1]
[1]
>>>from example1 impo
a=1
a<5
>>>
[EXE] [RUN] [A↔a] [CHAR]
```



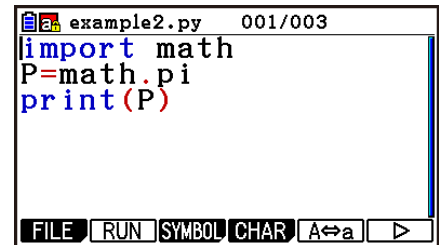
```
MicroPython
|CASIO COMPUTER CO.,
>>>from example1 impo
a=
[1] [0]
>>>from example1 impo
a=10
a>=5
>>>
[EXE] [RUN] [A↔a] [CHAR]
```

## Sample 2: Importing a Module

### Purpose

*import* imports a module and makes it possible to run the functions defined within it. Use the syntax below to execute function within the module.

<module name>.<function name>

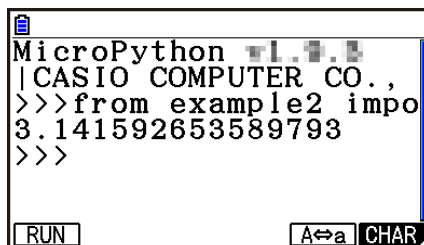


```
example2.py 001/003
import math
P=math.pi
print(P)
```

### Description

import math	Imports the math module and makes it possible to run the function defined by it.
P=math.pi	Defines variable P as <i>pi</i> , which is defined in the math module.
print(P)	Outputs the value stored in variable P.

### Execution Result



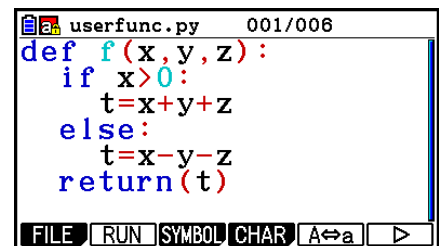
```
MicroPython 1.17.0
|CASIO COMPUTER CO.,
|>>>from example2 impo
|3.141592653589793
|>>>
```

## Sample 3: Defining a User-defined Function

### Purpose

*def* defines a user-defined function.

The script below recalls and uses the py script created under “Sample 4: Importing a py File”.



```
userfunc.py 001/006
def f(x,y,z):
    if x>0:
        t=x+y+z
    else:
        t=x-y-z
    return(t)
```

## Description

def f(x,y,z):	Defines a user-defined function with function name f, and arguments x, y, and z.
if x>0:	If variable x is greater than 0,
t=x+y+z	defines variable t as the execution result of x+y+z.
else:	Otherwise (if variable x is 0 or less),
t=x-y-z	defines variable t as the execution result of x-y-z.
return(t)	Makes t the return value.

Running this py script as a standalone script will only define the user defined function. The function will not be executed so the py script will end without output.

## Execution Result

```
MicroPython v1.17.0
|CASIO COMPUTER CO.,
>>>from userfunc impo
>>>
```

## Sample 4: Importing a py File

### Purpose

*import* can be used to import py files into other py files and run the processes written in the imported py files.

This makes it possible to use user-defined functions and variables across multiple py files.

Use the syntax below for execution of a module function or variable.


<py file (module) name>.<function name or variable name>

```
example3.py 001/003
import userfunc
a=userfunc.f(1,2,3)
print(a)
```

## Description

import userfunc	Imports userfunc.py and runs the written process.
a=userfunc.f(1,2,3)	Inputs arguments 1, 2, and 3 to the user defined function f defined by userfunc.py, executes the function f, and defines variable a as the result value.
print(a)	Outputs the value stored in the variable a.

## Execution Result



The screenshot shows a MicroPython shell window. The title bar reads "MicroPython 1.11.0". The window content displays the following text:  
|CASIO COMPUTER CO.,  
>>>from example3 impo  
6  
>>>  
At the bottom of the window, there are two buttons: "RUN" and "A↔a CHAR".

### ***Important!***

- To import py files into other py file or files, all of the files must be in the same directory (folder).
  - py files that can be imported with the SHELL screen are those described below.
    - If the SHELL is started up by a file list screen operation,\* importable files are py files in the directory displayed on the file list screen.
    - If the SHELL is started up by a script editor screen operation,\* importable files are py files in the same directory recalled with the script editor screen.
- \* For actual operations, see “To display the SHELL screen” (page 17-14).



# Chapter 18 Distribution (fx-CG50, fx-CG50 AU only)

18

In the **Distribution** mode, you can perform the eight distribution calculations below.

Discrete distributions: Binomial, Poisson, Geometric, Hypergeometric

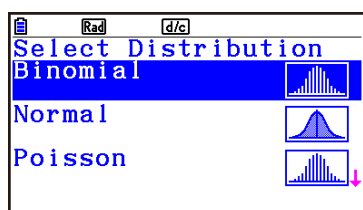
Continuous distributions: Normal, Student- $t$ ,  $\chi^2$ ,  $F$

To calculate probability values and draw a distribution graph, select a distribution type and then input parameter values. You can also perform inverse calculation to determine the value of  $x$  from a probability value.

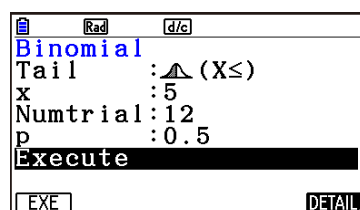
## 1. Distribution Mode Overview

### ■ Operation Flow

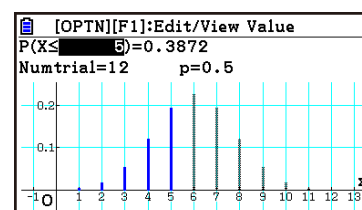
You can use the three steps below to display distribution calculation results (probability values) and a distribution graph.



1. Select a distribution type.  
(Distribution selection screen)



2. Input parameters.  
(Parameter input screen)



3. Perform the calculation.  
(Graph screen)

### Operation Example:

- **Determine the cumulative probability of a success count of two or less for a binomial distribution where the number of trials is five with a probability of 0.5.**

In the **Distribution** mode, select Binomial and then input the following parameters.

$x$ : 2 (Data value)

Numtrial: 5 (Number of trials)

$p$ : 0.5 (Probability of success)

- **Next, change data value  $x$  to 4, and then re-calculate the cumulative probability.**

1. From the Main Menu, enter the **Distribution** mode.

- This displays the distribution selection screen.

