Go

Table of Contents

Previous topic

1. Whetting Your Appetite

Next topic

3. An Informal Introduction to Python

This Page

- Report a Bug
- Show Source

Navigation

- index
- modules |
- next |
- previous |
-
- Python »
-
-
- 3.10.2 Documentation »
- The Python Tutorial »

2. Using the Python Interpreter

2.1. Invoking the Interpreter

The Python interpreter is usually installed as `/usr/local/bin/python3.10` on those machines where it is available; putting `/usr/local/bin` in your Unix shell's search path makes it possible to start it by typing the command:

```
python3.10
```

to the shell. [1] Since the choice of the directory where the interpreter lives is an installation option, other places are possible; check with your local Python guru or system administrator. (E.g., `/usr/local/python` is a popular alternative location.)

On Windows machines where you have installed Python from the Microsoft Store, the `python3.10` command will be available. If you have the py.exe launcher installed, you can use the `py` command. See Excursus: Setting environment variables for other ways to launch Python.

Typing an end-of-file character (`Control-D` on Unix, `Control-Z` on Windows) at the primary prompt causes the interpreter to exit with a zero exit status. If that doesn't work, you can exit the interpreter by typing the following command: `quit()`.

The interpreter's line-editing features include interactive editing, history substitution and code completion on systems that support the GNU Readline library. Perhaps the quickest check to see whether command line editing is supported is typing `Control-P` to the first Python prompt you get. If it beeps, you have command line editing; see Appendix Interactive Input Editing and History Substitution for an introduction to the keys. If nothing appears to happen, or if `^P` is echoed, command line editing isn't available; you'll only be able to use backspace to remove characters from the current line.

The interpreter operates somewhat like the Unix shell: when called with standard input connected to a tty device, it reads and executes commands interactively; when called with a file name argument or with a file as standard input, it reads and executes a *script* from that file.

A second way of starting the interpreter is `python -c command [arg] ...`, which executes the statement(s) in *command*, analogous to the shell's `-c` option. Since Python statements often contain spaces or other characters that are special to the shell, it is usually advised to quote *command* in its entirety with single quotes.

Some Python modules are also useful as scripts. These can be invoked using `python -m module [arg] ...`, which executes the source file for *module* as if you had spelled out its full name on the command line.

When a script file is used, it is sometimes useful to be able to run the script and enter interactive mode afterwards. This can be done by passing `-i` before the script.

All command line options are described in Command line and environment.

2.1.1. Argument Passing

When known to the interpreter, the script name and additional arguments thereafter are turned into a list of strings and assigned to the `argv` variable in the `sys` module. You can access this list by executing `import sys`. The length of the list is at least one; when no script and no arguments are given, `sys.argv[0]` is an empty string. When the script name is given as `'-'` (meaning standard input), `sys.argv[0]` is set to `'-'`. When `-c` *command* is used, `sys.argv[0]` is set to `'-c'`. When `-m` *module* is used, `sys.argv[0]` is set to the full name of the located module. Options found after `-c` *command* or `-m` *module* are not consumed by the Python interpreter's option processing but left in `sys.argv` for the command or module to handle.

2.1.2. Interactive Mode

When commands are read from a tty, the interpreter is said to be in *interactive mode*. In this mode it prompts for the next command with the *primary prompt*, usually three greater-than signs (>>>);

for continuation lines it prompts with the *secondary prompt*, by default three dots (`...`). The interpreter prints a welcome message stating its version number and a copyright notice before printing the first prompt:

```
$ python3.10
Python 3.10 (default, June 4 2019, 09:25:04)
[GCC 4.8.2] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Continuation lines are needed when entering a multi-line construct. As an example, take a look at this `if` statement:

```
>>> the_world_is_flat = True
>>> if the_world_is_flat:
...     print("Be careful not to fall off!")
...
Be careful not to fall off!
```

For more on interactive mode, see Interactive Mode.

2.2. The Interpreter and Its Environment

2.2.1. Source Code Encoding

By default, Python source files are treated as encoded in UTF-8. In that encoding, characters of most languages in the world can be used simultaneously in string literals, identifiers and comments — although the standard library only uses ASCII characters for identifiers, a convention that any portable code should follow. To display all these characters properly, your editor must recognize that the file is UTF-8, and it must use a font that supports all the characters in the file.

To declare an encoding other than the default one, a special comment line should be added as the *first* line of the file. The syntax is as follows:

```
# -*- coding: encoding -*-
```

where *encoding* is one of the valid `codecs` supported by Python.

For example, to declare that Windows-1252 encoding is to be used, the first line of your source code file should be:

```
# -*- coding: cp1252 -*-
```

One exception to the *first line* rule is when the source code starts with a UNIX “shebang” line. In this case, the encoding declaration should be added as the second line of the file. For example:

```
#!/usr/bin/env python3
# -*- coding: cp1252 -*-
```

Footnotes

[1] On Unix, the Python 3.x interpreter is by default not installed with the executable named `python`, so that it does not conflict with a simultaneously installed Python 2.x executable.

Navigation

- index
- modules |
- next |
- previous |
-
- Python »
-
-
- 3.10.2 Documentation »
- The Python Tutorial »