

Python is an easy to learn, powerful programming language. It has efficient high-level data structures and a simple but effective approach to object-oriented programming. Python's elegant syntax and dynamic typing, together with its interpreted nature, make it an ideal language for scripting and rapid application development in many areas on most platforms.

The Python interpreter and the extensive standard library are freely available in source or binary form for all major platforms from the Python web site, <https://www.python.org/>, and may be freely distributed. The same site also contains distributions of and pointers to many free third party Python modules, programs and tools, and additional documentation.

The Python interpreter is easily extended with new functions and data types implemented in C or C++ (or other languages callable from C). Python is also suitable as an extension language for customizable applications.

This tutorial introduces the reader informally to the basic concepts and features of the Python language and system. It helps to have a Python interpreter handy for hands-on experience, but all examples are self-contained, so the tutorial can be read off-line as well.

For a description of standard objects and modules, see The Python Standard Library. The Python Language Reference gives a more formal definition of the language. To write extensions in C or C++, read Extending and Embedding the Python Interpreter and Python/C API Reference Manual. There are also several books covering Python in depth.

This tutorial does not attempt to be comprehensive and cover every single feature, or even every commonly used feature. Instead, it introduces many of Python's most noteworthy features, and will give you a good idea of the language's flavor and style. After reading it, you will be able to read and write Python modules and programs, and you will be ready to learn more about the various Python library modules described in The Python Standard Library. 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: