

---

# **Python Tutorial**

***Release 3.9.1***

**Guido van Rossum  
and the Python development team**

**January 12, 2021**

**Python Software Foundation  
Email: [docs@python.org](mailto:docs@python.org)**



# CONTENTS

<b>1</b>	<b>Whetting Your Appetite</b>	<b>3</b>
<b>2</b>	<b>Using the Python Interpreter</b>	<b>5</b>
2.1	Invoking the Interpreter . . . . .	5
2.1.1	Argument Passing . . . . .	6
2.1.2	Interactive Mode . . . . .	6
2.2	The Interpreter and Its Environment . . . . .	6
2.2.1	Source Code Encoding . . . . .	6
<b>3</b>	<b>An Informal Introduction to Python</b>	<b>9</b>
3.1	Using Python as a Calculator . . . . .	9
3.1.1	Numbers . . . . .	9
3.1.2	Strings . . . . .	11
3.1.3	Lists . . . . .	14
3.2	First Steps Towards Programming . . . . .	15
<b>4</b>	<b>More Control Flow Tools</b>	<b>17</b>
4.1	if Statements . . . . .	17
4.2	for Statements . . . . .	17
4.3	The range() Function . . . . .	18
4.4	break and continue Statements, and else Clauses on Loops . . . . .	19
4.5	pass Statements . . . . .	20
4.6	Defining Functions . . . . .	20
4.7	More on Defining Functions . . . . .	22
4.7.1	Default Argument Values . . . . .	22
4.7.2	Keyword Arguments . . . . .	23
4.7.3	Special parameters . . . . .	24
4.7.4	Arbitrary Argument Lists . . . . .	27
4.7.5	Unpacking Argument Lists . . . . .	27
4.7.6	Lambda Expressions . . . . .	27
4.7.7	Documentation Strings . . . . .	28
4.7.8	Function Annotations . . . . .	28
4.8	Intermezzo: Coding Style . . . . .	29
<b>5</b>	<b>Data Structures</b>	<b>31</b>
5.1	More on Lists . . . . .	31
5.1.1	Using Lists as Stacks . . . . .	32
5.1.2	Using Lists as Queues . . . . .	33
5.1.3	List Comprehensions . . . . .	33
5.1.4	Nested List Comprehensions . . . . .	34
5.2	The del statement . . . . .	35
5.3	Tuples and Sequences . . . . .	36
5.4	Sets . . . . .	37
5.5	Dictionaries . . . . .	37
5.6	Looping Techniques . . . . .	38

5.7	More on Conditions . . . . .	40
5.8	Comparing Sequences and Other Types . . . . .	40
<b>6</b>	<b>Modules</b>	<b>41</b>
6.1	More on Modules . . . . .	42
6.1.1	Executing modules as scripts . . . . .	43
6.1.2	The Module Search Path . . . . .	43
6.1.3	“Compiled” Python files . . . . .	44
6.2	Standard Modules . . . . .	44
6.3	The <code>dir()</code> Function . . . . .	45
6.4	Packages . . . . .	46
6.4.1	Importing <code>*</code> From a Package . . . . .	47
6.4.2	Intra-package References . . . . .	48
6.4.3	Packages in Multiple Directories . . . . .	48
<b>7</b>	<b>Input and Output</b>	<b>49</b>
7.1	Fancier Output Formatting . . . . .	49
7.1.1	Formatted String Literals . . . . .	50
7.1.2	The String <code>format()</code> Method . . . . .	51
7.1.3	Manual String Formatting . . . . .	52
7.1.4	Old string formatting . . . . .	52
7.2	Reading and Writing Files . . . . .	53
7.2.1	Methods of File Objects . . . . .	54
7.2.2	Saving structured data with <code>json</code> . . . . .	55
<b>8</b>	<b>Errors and Exceptions</b>	<b>57</b>
8.1	Syntax Errors . . . . .	57
8.2	Exceptions . . . . .	57
8.3	Handling Exceptions . . . . .	58
8.4	Raising Exceptions . . . . .	60
8.5	Exception Chaining . . . . .	61
8.6	User-defined Exceptions . . . . .	61
8.7	Defining Clean-up Actions . . . . .	62
8.8	Predefined Clean-up Actions . . . . .	63
<b>9</b>	<b>Classes</b>	<b>65</b>
9.1	A Word About Names and Objects . . . . .	65
9.2	Python Scopes and Namespaces . . . . .	66
9.2.1	Scopes and Namespaces Example . . . . .	67
9.3	A First Look at Classes . . . . .	68
9.3.1	Class Definition Syntax . . . . .	68
9.3.2	Class Objects . . . . .	68
9.3.3	Instance Objects . . . . .	69
9.3.4	Method Objects . . . . .	69
9.3.5	Class and Instance Variables . . . . .	70
9.4	Random Remarks . . . . .	71
9.5	Inheritance . . . . .	72
9.5.1	Multiple Inheritance . . . . .	73
9.6	Private Variables . . . . .	74
9.7	Odds and Ends . . . . .	74
9.8	Iterators . . . . .	75
9.9	Generators . . . . .	76
9.10	Generator Expressions . . . . .	77
<b>10</b>	<b>Brief Tour of the Standard Library</b>	<b>79</b>
10.1	Operating System Interface . . . . .	79
10.2	File Wildcards . . . . .	79
10.3	Command Line Arguments . . . . .	80
10.4	Error Output Redirection and Program Termination . . . . .	80

10.5	String Pattern Matching . . . . .	80
10.6	Mathematics . . . . .	81
10.7	Internet Access . . . . .	81
10.8	Dates and Times . . . . .	82
10.9	Data Compression . . . . .	82
10.10	Performance Measurement . . . . .	82
10.11	Quality Control . . . . .	83
10.12	Batteries Included . . . . .	83
<b>11</b>	<b>Brief Tour of the Standard Library — Part II</b>	<b>85</b>
11.1	Output Formatting . . . . .	85
11.2	Templating . . . . .	86
11.3	Working with Binary Data Record Layouts . . . . .	87
11.4	Multi-threading . . . . .	87
11.5	Logging . . . . .	88
11.6	Weak References . . . . .	88
11.7	Tools for Working with Lists . . . . .	89
11.8	Decimal Floating Point Arithmetic . . . . .	90
<b>12</b>	<b>Virtual Environments and Packages</b>	<b>91</b>
12.1	Introduction . . . . .	91
12.2	Creating Virtual Environments . . . . .	91
12.3	Managing Packages with pip . . . . .	92
<b>13</b>	<b>What Now?</b>	<b>95</b>
<b>14</b>	<b>Interactive Input Editing and History Substitution</b>	<b>97</b>
14.1	Tab Completion and History Editing . . . . .	97
14.2	Alternatives to the Interactive Interpreter . . . . .	97
<b>15</b>	<b>Floating Point Arithmetic: Issues and Limitations</b>	<b>99</b>
15.1	Representation Error . . . . .	101
<b>16</b>	<b>Appendix</b>	<b>105</b>
16.1	Interactive Mode . . . . .	105
16.1.1	Error Handling . . . . .	105
16.1.2	Executable Python Scripts . . . . .	105
16.1.3	The Interactive Startup File . . . . .	106
16.1.4	The Customization Modules . . . . .	106
<b>A</b>	<b>Glossary</b>	<b>107</b>
<b>B</b>	<b>About these documents</b>	<b>119</b>
B.1	Contributors to the Python Documentation . . . . .	119
<b>C</b>	<b>History and License</b>	<b>121</b>
C.1	History of the software . . . . .	121
C.2	Terms and conditions for accessing or otherwise using Python . . . . .	122
C.2.1	PSF LICENSE AGREEMENT FOR PYTHON 3.9.1 . . . . .	122
C.2.2	BEOPEN.COM LICENSE AGREEMENT FOR PYTHON 2.0 . . . . .	123
C.2.3	CNRI LICENSE AGREEMENT FOR PYTHON 1.6.1 . . . . .	124
C.2.4	CWI LICENSE AGREEMENT FOR PYTHON 0.9.0 THROUGH 1.2 . . . . .	125
C.2.5	ZERO-CLAUSE BSD LICENSE FOR CODE IN THE PYTHON 3.9.1 DOCUMENTA- TION . . . . .	125
C.3	Licenses and Acknowledgements for Incorporated Software . . . . .	126
C.3.1	Mersenne Twister . . . . .	126
C.3.2	Sockets . . . . .	127
C.3.3	Asynchronous socket services . . . . .	127
C.3.4	Cookie management . . . . .	128
C.3.5	Execution tracing . . . . .	128

C.3.6	UUencode and UUdecode functions . . . . .	129
C.3.7	XML Remote Procedure Calls . . . . .	129
C.3.8	test_epoll . . . . .	130
C.3.9	Select kqueue . . . . .	130
C.3.10	SipHash24 . . . . .	131
C.3.11	strtod and dtoa . . . . .	131
C.3.12	OpenSSL . . . . .	132
C.3.13	expat . . . . .	134
C.3.14	libffi . . . . .	134
C.3.15	zlib . . . . .	135
C.3.16	cfuhash . . . . .	135
C.3.17	libmpdec . . . . .	136
C.3.18	W3C C14N test suite . . . . .	136
<b>D</b>	<b>Copyright</b>	<b>139</b>
	<b>Index</b>	<b>141</b>

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 [library-index](#). [reference-index](#) gives a more formal definition of the language. To write extensions in C or C++, read [extending-index](#) and [c-api-index](#). 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 [library-index](#).

The [Glossary](#) is also worth going through.





## WHETTING YOUR APPETITE

If you do much work on computers, eventually you find that there's some task you'd like to automate. For example, you may wish to perform a search-and-replace over a large number of text files, or rename and rearrange a bunch of photo files in a complicated way. Perhaps you'd like to write a small custom database, or a specialized GUI application, or a simple game.

If you're a professional software developer, you may have to work with several C/C++/Java libraries but find the usual write/compile/test/re-compile cycle is too slow. Perhaps you're writing a test suite for such a library and find writing the testing code a tedious task. Or maybe you've written a program that could use an extension language, and you don't want to design and implement a whole new language for your application.

Python is just the language for you.

You could write a Unix shell script or Windows batch files for some of these tasks, but shell scripts are best at moving around files and changing text data, not well-suited for GUI applications or games. You could write a C/C++/Java program, but it can take a lot of development time to get even a first-draft program. Python is simpler to use, available on Windows, Mac OS X, and Unix operating systems, and will help you get the job done more quickly.

Python is simple to use, but it is a real programming language, offering much more structure and support for large programs than shell scripts or batch files can offer. On the other hand, Python also offers much more error checking than C, and, being a *very-high-level language*, it has high-level data types built in, such as flexible arrays and dictionaries. Because of its more general data types Python is applicable to a much larger problem domain than Awk or even Perl, yet many things are at least as easy in Python as in those languages.

Python allows you to split your program into modules that can be reused in other Python programs. It comes with a large collection of standard modules that you can use as the basis of your programs — or as examples to start learning to program in Python. Some of these modules provide things like file I/O, system calls, sockets, and even interfaces to graphical user interface toolkits like Tk.

Python is an interpreted language, which can save you considerable time during program development because no compilation and linking is necessary. The interpreter can be used interactively, which makes it easy to experiment with features of the language, to write throw-away programs, or to test functions during bottom-up program development. It is also a handy desk calculator.

Python enables programs to be written compactly and readably. Programs written in Python are typically much shorter than equivalent C, C++, or Java programs, for several reasons:

- the high-level data types allow you to express complex operations in a single statement;
- statement grouping is done by indentation instead of beginning and ending brackets;
- no variable or argument declarations are necessary.

Python is *extensible*: if you know how to program in C it is easy to add a new built-in function or module to the interpreter, either to perform critical operations at maximum speed, or to link Python programs to libraries that may only be available in binary form (such as a vendor-specific graphics library). Once you are really hooked, you can link the Python interpreter into an application written in C and use it as an extension or command language for that application.

By the way, the language is named after the BBC show “Monty Python’s Flying Circus” and has nothing to do with reptiles. Making references to Monty Python skits in documentation is not only allowed, it is encouraged!

Now that you are all excited about Python, you'll want to examine it in some more detail. Since the best way to learn a language is to use it, the tutorial invites you to play with the Python interpreter as you read.

In the next chapter, the mechanics of using the interpreter are explained. This is rather mundane information, but essential for trying out the examples shown later.

The rest of the tutorial introduces various features of the Python language and system through examples, beginning with simple expressions, statements and data types, through functions and modules, and finally touching upon advanced concepts like exceptions and user-defined classes.

## USING THE PYTHON INTERPRETER

### 2.1 Invoking the Interpreter

The Python interpreter is usually installed as `/usr/local/bin/python3.9` 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:

```
python3.9
```

to the shell.<sup>1</sup> 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 `python3.9` command will be available. If you have the `py.exe` launcher installed, you can use the `py` command. See [setting-envvars](#) 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 [using-on-general](#).

---

<sup>1</sup> 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.