

The Python Tutorial

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

- [1. Whetting Your Appetite](#)
- [2. Using the Python Interpreter](#)
 - [2.1. Invoking the Interpreter](#)
 - [2.1.1. Argument Passing](#)
 - [2.1.2. Interactive Mode](#)
 - [2.2. The Interpreter and Its Environment](#)

- 2.2.1. Source Code Encoding
- 3. An Informal Introduction to Python
 - 3.1. Using Python as a Calculator
 - 3.1.1. Numbers
 - 3.1.2. Strings
 - 3.1.3. Lists
 - 3.2. First Steps Towards Programming
- 4. More Control Flow Tools
 - 4.1. `if` Statements
 - 4.2. `for` Statements
 - 4.3. The `range()` Function
 - 4.4. `break` and `continue` Statements, and `else` Clauses on Loops
 - 4.5. `pass` Statements
 - 4.6. Defining Functions
 - 4.7. More on Defining Functions
 - 4.7.1. Default Argument Values
 - 4.7.2. Keyword Arguments
 - 4.7.3. Arbitrary Argument Lists
 - 4.7.4. Unpacking Argument Lists
 - 4.7.5. Lambda Expressions
 - 4.7.6. Documentation Strings
 - 4.7.7. Function Annotations
 - 4.8. Intermezzo: Coding Style
- 5. Data Structures
 - 5.1. More on Lists
 - 5.1.1. Using Lists as Stacks
 - 5.1.2. Using Lists as Queues
 - 5.1.3. List Comprehensions
 - 5.1.4. Nested List Comprehensions
 - 5.2. The `del` statement
 - 5.3. Tuples and Sequences
 - 5.4. Sets
 - 5.5. Dictionaries
 - 5.6. Looping Techniques
 - 5.7. More on Conditions
 - 5.8. Comparing Sequences and Other Types
- 6. Modules
 - 6.1. More on Modules
 - 6.1.1. Executing modules as scripts
 - 6.1.2. The Module Search Path
 - 6.1.3. “Compiled” Python files
 - 6.2. Standard Modules
 - 6.3. The `dir()` Function
 - 6.4. Packages
 - 6.4.1. Importing `*` From a Package

- 6.4.2. Intra-package References
 - 6.4.3. Packages in Multiple Directories
- 7. Input and Output
 - 7.1. Fancier Output Formatting
 - 7.1.1. Old string formatting
 - 7.2. Reading and Writing Files
 - 7.2.1. Methods of File Objects
 - 7.2.2. Saving structured data with `json`
- 8. Errors and Exceptions
 - 8.1. Syntax Errors
 - 8.2. Exceptions
 - 8.3. Handling Exceptions
 - 8.4. Raising Exceptions
 - 8.5. User-defined Exceptions
 - 8.6. Defining Clean-up Actions
 - 8.7. Predefined Clean-up Actions
- 9. Classes
 - 9.1. A Word About Names and Objects
 - 9.2. Python Scopes and Namespaces
 - 9.2.1. Scopes and Namespaces Example
 - 9.3. A First Look at Classes
 - 9.3.1. Class Definition Syntax
 - 9.3.2. Class Objects
 - 9.3.3. Instance Objects
 - 9.3.4. Method Objects
 - 9.3.5. Class and Instance Variables
 - 9.4. Random Remarks
 - 9.5. Inheritance
 - 9.5.1. Multiple Inheritance
 - 9.6. Private Variables
 - 9.7. Odds and Ends
 - 9.8. Iterators
 - 9.9. Generators
 - 9.10. Generator Expressions
- 10. Brief Tour of the Standard Library
 - 10.1. Operating System Interface
 - 10.2. File Wildcards
 - 10.3. Command Line Arguments
 - 10.4. Error Output Redirection and Program Termination
 - 10.5. String Pattern Matching
 - 10.6. Mathematics
 - 10.7. Internet Access
 - 10.8. Dates and Times
 - 10.9. Data Compression
 - 10.10. Performance Measurement

- 10.11. Quality Control
 - 10.12. Batteries Included
- 11. Brief Tour of the Standard Library — Part II
 - 11.1. Output Formatting
 - 11.2. Templating
 - 11.3. Working with Binary Data Record Layouts
 - 11.4. Multi-threading
 - 11.5. Logging
 - 11.6. Weak References
 - 11.7. Tools for Working with Lists
 - 11.8. Decimal Floating Point Arithmetic
- 12. Virtual Environments and Packages
 - 12.1. Introduction
 - 12.2. Creating Virtual Environments
 - 12.3. Managing Packages with pip
- 13. What Now?
- 14. Interactive Input Editing and History Substitution
 - 14.1. Tab Completion and History Editing
 - 14.2. Alternatives to the Interactive Interpreter
- 15. Floating Point Arithmetic: Issues and Limitations
 - 15.1. Representation Error
- 16. Appendix
 - 16.1. Interactive Mode
 - 16.1.1. Error Handling
 - 16.1.2. Executable Python Scripts
 - 16.1.3. The Interactive Startup File
 - 16.1.4. The Customization Modules