WIKIPEDIA

Python (programming language)

Python is an interpreted, high-level, general-purpose programming language. Created by Guido van Rossum and first released in 1991. Python's design philosophy emphasizes code readability with its notable use of significant whitespace. Its language constructs and object-oriented approach aim to help programmers write clear, logical code for small and large-scale projects.[30]

Python is dynamically typed and garbage-collected. It supports multiple programming paradigms, including structured (particularly, procedural), object-oriented, and functional programming. Python is often described as a "batteries included" language due to its comprehensive standard library.[31]

Python was conceived in the late 1980s as a successor to the ABC language. Python 2.0, released in 2000, introduced features like list comprehensions and a garbage collection system capable of collecting reference cycles. Python 3.0, released in 2008, was a major revision of the language that is not completely backward-compatible, and much Python 2 code does not run unmodified on Python 3.

The Python 2 language, i.e. Python 2.7.x, was officially discontinued on 1 January 2020 (first planned for 2015) after which security patches and other improvements will not be released for it.[32][33] With Python 2's end-of-life, only Python 3.5.x[34] and later are supported.

Python interpreters are available for many operating systems. A global community of programmers develops and maintains CPython, an open source[35] reference implementation. A non-profit organization, the Python Software Foundation, manages and directs resources for Python and CPython development.

Python

Paradigm	Multi-paradigm: functional, imperative, object-oriented, structured, reflective
Designed by	Guido van Rossum
Developer	Python Software Foundation
First appeared	1990[1]
Stable release	3.8.2 / 24 February 2020[2]
Preview release	3.9.0a5 / 23 March 2020[3] and 2.7.18rc1[4][5]
Typing discipline	Duck, dynamic, gradual (since 3.5)[6]
License	Python Software Foundation License
Filename extensions	.py, .pyi, .pyc, .pyd, .pyo (prior to 3.5),[7] .pyw, .pyz (since 3.5)[8]
Website	www.python.org (https://www.python.org/)

Major implementations

CPython, PyPy, Stackless Python, MicroPython, CircuitPython, IronPython, Jython, RustPython

Dialects

Cython, RPython, Starlark[9]

Influenced by

ABC,[10], Ada[11], ALGOL 68,[12]

Contents

History
Features and philosophy
Syntax and semantics
- Indentation
- Statements and control flow
- Expressions
- Methods
- Typing

Mathematics
Python programming examples
Libraries
Development environments
Implementations
Reference implementation
Other implementations
Unsupported implementations
Cross-compilers to other languages
Performance
Development
Naming
API documentation generators
Uses
Languages influenced by Python
See also
References
Sources
Further reading
External links

APL,[13] C,[14] C++,[15] CLU,[16] Dylan,[17] Haskell,[18] Icon,[19] Java,[20] Lisp,[21] Modula-3,[15] Perl, Standard ML[13]
Influenced
Apache Groovy, Boo, Cobra, CoffeeScript,[22] D, F#, Genie,[23] Go, JavaScript,[24][25] Julia,[26] Nim, Ring,[27] Ruby,[28] Swift[29]
Python Programming at Wikibooks

History

Guido van Rossum at OSCON 2006

Python was conceived in the late 1980s[36] by Guido van Rossum at Centrum Wiskunde & Informatica (CWI) in the Netherlands as a successor to the ABC language (itself inspired by SETL),[37] capable of exception handling and interfacing with the Amoeba operating system.[10] Its implementation began in December 1989.[38] Van Rossum shouldered sole responsibility for the project, as the lead developer, until 12 July 2018, when he announced his "permanent vacation" from his responsibilities as Python's *Benevolent Dictator For Life*, a title the Python community bestowed upon him to reflect his long-term commitment as the project's chief decision-maker.[39] He now shares his leadership as a member of a five-person steering council.[40][41][42] In January 2019, active Python core developers elected Brett Cannon, Nick Coghlan, Barry Warsaw, Carol Willing and Van Rossum to a five-member "Steering Council" to lead the project.[43]

Python 2.0 was released on 16 October 2000 with many major new features, including a cycle-detecting garbage collector and support for Unicode.[44]

Python 3.0 was released on 3 December 2008. It was a major revision of the language that is not completely backward-compatible.[45] Many of its major features were backported to Python 2.6.x[46] and 2.7.x version series. Releases of Python 3 include the `2to3` utility, which automates (at least partially) the translation of Python 2 code to Python 3.[47]

Python 2.7's end-of-life date was initially set at 2015 then postponed to 2020 out of concern that a large body of existing code could not easily be forward-ported to Python 3.[48][49]

Features and philosophy

Python is a multi-paradigm programming language. Object-oriented programming and structured programming are fully supported, and many of its features support functional programming and aspect-oriented programming (including by metaprogramming[50] and metaobjects (magic methods)).[51] Many other paradigms are supported via extensions, including design by contract[52][53] and logic programming.[54]

Python uses dynamic typing and a combination of reference counting and a cycle-detecting garbage collector for memory management. It also features dynamic name resolution (late binding), which binds method and variable names during program execution.

Python's design offers some support for functional programming in the Lisp tradition. It has `filter`, `map`, and `reduce` functions; list comprehensions, dictionaries, sets, and generator expressions.[55] The standard library has two modules (itertools and functools) that implement functional tools borrowed from Haskell and Standard ML.[56]

The language's core philosophy is summarized in the document *The Zen of Python* (*PEP 20*), which includes aphorisms such as:[57]

- Beautiful is better than ugly.
- Explicit is better than implicit.
- Simple is better than complex.
- Complex is better than complicated.
- Readability counts.

Rather than having all of its functionality built into its core, Python was designed to be highly extensible. This compact modularity has made it particularly popular as a means of adding programmable interfaces to existing applications. Van Rossum's vision of a small core language with a large standard library and easily extensible interpreter stemmed from his frustrations with ABC, which espoused the opposite approach.[36]

Python strives for a simpler, less-cluttered syntax and grammar while giving developers a choice in their coding methodology. In contrast to Perl's "there is more than one way to do it" motto, Python embraces a "there should be one—and preferably only one—obvious way to do it" design philosophy.[57] Alex Martelli, a Fellow at the Python Software Foundation and Python book author, writes that "To describe something as 'clever' is *not* considered a compliment in the Python culture."[58]

Python's developers strive to avoid premature optimization, and reject patches to non-critical parts of the CPython reference implementation that would offer marginal increases in speed at the cost of clarity.[59] When speed is important, a Python programmer can move time-critical functions to extension modules written in languages such as C, or use PyPy, a just-in-time compiler. Cython is also available, which translates a Python script into C and makes direct C-level API calls into the Python interpreter.

An important goal of Python's developers is keeping it fun to use. This is reflected in the language's name—a tribute to the British comedy group Monty Python[60]—and in occasionally playful approaches to tutorials and reference materials, such as examples that refer to spam and eggs (from a famous Monty Python sketch) instead of the standard foo and bar.[61][62]

A common neologism in the Python community is *pythonic*, which can have a wide range of meanings related to program style. To say that code is pythonic is to say that it uses Python idioms well, that it is natural or shows fluency in the language, that it conforms with Python's minimalist philosophy and emphasis on readability. In contrast, code that is difficult to understand or reads like a rough transcription from another programming language is called *unpythonic*.

Users and admirers of Python, especially those considered knowledgeable or experienced, are often referred to as *Pythonistas*.[63][64]

Syntax and semantics

Python is meant to be an easily readable language. Its formatting is visually uncluttered, and it often uses English keywords where other languages use punctuation. Unlike many other languages, it does not use curly brackets to delimit blocks, and semicolons after statements are optional. It has fewer syntactic exceptions and special cases than C or Pascal.[65]

Indentation

Python uses whitespace indentation, rather than curly brackets or keywords, to delimit blocks. An increase in indentation comes after certain statements; a decrease in indentation signifies the end of the current block.[66] Thus, the program's visual structure accurately represents the program's semantic structure.[1] This feature is sometimes termed the off-side rule, which some other languages share, but in most languages indentation doesn't have any semantic meaning.

Statements and control flow

Python's statements include (among others):

- The assignment statement (token '=', the equals sign). This operates differently than in traditional imperative programming languages, and this fundamental mechanism (including the nature of Python's version of *variables*) illuminates many other features of the language. Assignment in C, e.g., `x = 2`, translates to "typed variable name *x* receives a copy of numeric value 2". The (right-hand) value is copied into an allocated storage location for which the (left-hand) variable name is the symbolic address. The memory allocated to the variable is large enough (potentially quite large) for the declared type. In the simplest case of Python assignment, using the same example, `x = 2`, translates to "(generic) name x receives a reference to a separate, dynamically allocated object of numeric (int) type of value 2." This is termed *binding* the name to the object. Since the name's storage location doesn't *contain* the indicated value, it is improper to call it a *variable*. Names may be subsequently rebound at any time to objects of greatly varying types, including strings, procedures, complex objects with data and methods, etc. Successive assignments of a common value to multiple names, e.g., `x = 2`; `y = 2`; `z = 2` result in allocating storage to (at most) three names and one numeric object, to which all three names are bound. Since a name is a generic reference holder it is unreasonable to associate a fixed data type with it. However at a given time a name will be bound to *some* object, which **will** have a type; thus there is dynamic typing.
- The `if` statement, which conditionally executes a block of code, along with `else` and `elif` (a contraction of else-if).
- The `for` statement, which iterates over an iterable object, capturing each element to a local variable for use by the attached block.
- The `while` statement, which executes a block of code as long as its condition is true.

- The `try` statement, which allows exceptions raised in its attached code block to be caught and handled by `except` clauses; it also ensures that clean-up code in a `finally` block will always be run regardless of how the block exits.
- The `raise` statement, used to raise a specified exception or re-raise a caught exception.
- The `class` statement, which executes a block of code and attaches its local namespace to a class, for use in object-oriented programming.
- The `def` statement, which defines a function or method.
- The `with` statement, from Python 2.5 released in September 2006,[67] which encloses a code block within a context manager (for example, acquiring a lock before the block of code is run and releasing the lock afterwards, or opening a file and then closing it), allowing Resource Acquisition Is Initialization (RAII)-like behavior and replaces a common try/finally idiom.[68]
- The `break` statement, exits from the loop.
- The `continue` statement, skips this iteration and continues with the next item.
- The `pass` statement, which serves as a NOP. It is syntactically needed to create an empty code block.
- The `assert` statement, used during debugging to check for conditions that ought to apply.
- The `yield` statement, which returns a value from a generator function. From Python 2.5, `yield` is also an operator. This form is used to implement coroutines.
- The `import` statement, which is used to import modules whose functions or variables can be used in the current program. There are three ways of using import: `import <module name> [as <alias>]` or `from <module name> import *` or `from <module name> import <definition 1> [as <alias 1>], <definition 2> [as <alias 2>], ....`
- The `print` statement was changed to the `print()` function in Python 3.

Python does not support tail call optimization or first-class continuations, and, according to Guido van Rossum, it never will.[69][70] However, better support for coroutine-like functionality is provided in 2.5, by extending Python's generators.[71] Before 2.5, generators were lazy iterators; information was passed unidirectionally out of the generator. From Python 2.5, it is possible to pass information back into a generator function, and from Python 3.3, the information can be passed through multiple stack levels.[72]

Expressions

Some Python expressions are similar to languages such as C and Java, while some are not:

- Addition, subtraction, and multiplication are the same, but the behavior of division differs. There are two types of divisions in Python. They are floor division (or integer division) `//` and floating point`/`division.[73] Python also added the `**` operator for exponentiation.
- From Python 3.5, the new `@` infix operator was introduced. It is intended to be used by libraries such as NumPy for matrix multiplication.[74][75]
- From Python 3.8, the syntax `:=`, called the 'walrus operator' was introduced. It assigns values to variables as part of a larger expression.[76]
- In Python, `==` compares by value, versus Java, which compares numerics by value[77] and objects by reference.[78] (Value comparisons in Java on objects can be performed with the `equals()` method.) Python's `is` operator may be used to compare object identities (comparison by reference). In Python, comparisons may be chained, for example `a <= b <= c`.
- Python uses the words `and`, `or`, `not` for its boolean operators rather than the symbolic `&&`, `||`, `!` used in Java and C.

- Python has a type of expression termed a *list comprehension*. Python 2.4 extended list comprehensions into a more general expression termed a *generator expression*.[55]
- Anonymous functions are implemented using lambda expressions; however, these are limited in that the body can only be one expression.
- Conditional expressions in Python are written as `x if c else y`[79] (different in order of operands from the `c ? x : y` operator common to many other languages).
- Python makes a distinction between lists and tuples. Lists are written as `[1, 2, 3]`, are mutable, and cannot be used as the keys of dictionaries (dictionary keys must be immutable in Python). Tuples are written as `(1, 2, 3)`, are immutable and thus can be used as the keys of dictionaries, provided all elements of the tuple are immutable. The + operator can be used to concatenate two tuples, which does not directly modify their contents, but rather produces a new tuple containing the elements of both provided tuples. Thus, given the variable `t` initially equal to `(1, 2, 3)`, executing `t = t + (4, 5)` first evaluates `t + (4, 5)`, which yields `(1, 2, 3, 4, 5)`, which is then assigned back to `t`, thereby effectively "modifying the contents" of `t`, while conforming to the immutable nature of tuple objects. Parentheses are optional for tuples in unambiguous contexts.[80]
- Python features *sequence unpacking* wherein multiple expressions, each evaluating to anything that can be assigned to (a variable, a writable property, etc.), are associated in the identical manner to that forming tuple literals and, as a whole, are put on the left hand side of the equal sign in an assignment statement. The statement expects an *iterable* object on the right hand side of the equal sign that produces the same number of values as the provided writable expressions when iterated through, and will iterate through it, assigning each of the produced values to the corresponding expression on the left.[81]
- Python has a "string format" operator %. This functions analogous to `printf` format strings in C, e.g. `"spam=%s eggs=%d" % ("blah", 2)` evaluates to `"spam=blah eggs=2"`. In Python 3 and 2.6+, this was supplemented by the `format()` method of the `str` class, e.g. `"spam={0} eggs={1}".format("blah", 2)`. Python 3.6 added "f-strings": `blah = "blah"; eggs = 2; f'spam={blah} eggs={eggs}'`.[82]
- Python has various kinds of string literals:
  - Strings delimited by single or double quote marks. Unlike in Unix shells, Perl and Perl-influenced languages, single quote marks and double quote marks function identically. Both kinds of string use the backslash (\) as an escape character. String interpolation became available in Python 3.6 as "formatted string literals".[82]
  - Triple-quoted strings, which begin and end with a series of three single or double quote marks. They may span multiple lines and function like here documents in shells, Perl and Ruby.
  - Raw string varieties, denoted by prefixing the string literal with an `r`. Escape sequences are not interpreted; hence raw strings are useful where literal backslashes are common, such as regular expressions and Windows-style paths. Compare "@-quoting" in C#.
- Python has array index and array slicing expressions on lists, denoted as `a[key]`, `a[start:stop]` or `a[start:stop:step]`. Indexes are zero-based, and negative indexes are relative to the end. Slices take elements from the *start* index up to, but not including, the *stop* index. The third slice parameter, called *step* or *stride*, allows elements to be skipped and reversed. Slice indexes may be omitted, for example `a[:]` returns a copy of the entire list. Each element of a slice is a shallow copy.

In Python, a distinction between expressions and statements is rigidly enforced, in contrast to languages such as Common Lisp, Scheme, or Ruby. This leads to duplicating some functionality. For example:

- List comprehensions vs. `for`-loops

- Conditional expressions vs. `if` blocks
- The `eval()` vs. `exec()` built-in functions (in Python 2, `exec` is a statement); the former is for expressions, the latter is for statements.

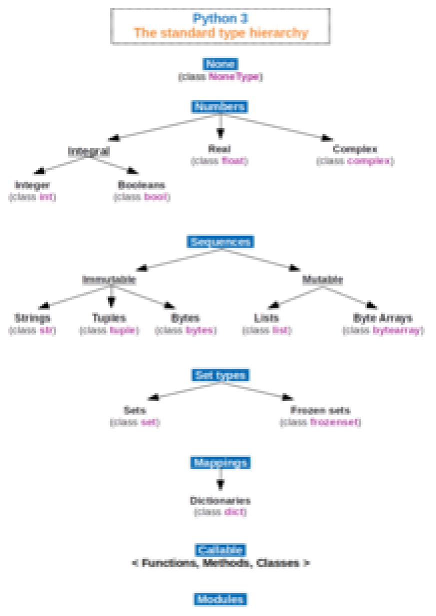
Statements cannot be a part of an expression, so list and other comprehensions or lambda expressions, all being expressions, cannot contain statements. A particular case of this is that an assignment statement such as `a = 1` cannot form part of the conditional expression of a conditional statement. This has the advantage of avoiding a classic C error of mistaking an assignment operator `=` for an equality operator `==` in conditions: `if (c = 1) { ... }` is syntactically valid (but probably unintended) C code but `if c = 1: ...` causes a syntax error in Python.

Methods

Methods on objects are functions attached to the object's class; the syntax `instance.method(argument)` is, for normal methods and functions, syntactic sugar for `Class.method(instance, argument)`. Python methods have an explicit `self` parameter to access instance data, in contrast to the implicit `self` (or `this`) in some other object-oriented programming languages (e.g., C++, Java, Objective-C, or Ruby).[83]

Typing

Python uses duck typing and has typed objects but untyped variable names. Type constraints are not checked at compile time; rather, operations on an object may fail, signifying that the given object is not of a suitable type. Despite being dynamically typed, Python is strongly typed, forbidding operations that are not well-defined (for example, adding a number to a string) rather than silently attempting to make sense of them.



The standard type hierarchy in Python 3

Python allows programmers to define their own types using classes, which are most often used for object-oriented programming. New instances of classes are constructed by calling the class (for example, `SpamClass()` or `EggsClass()`), and the classes are instances of the metaclass `type` (itself an instance of itself), allowing metaprogramming and reflection.

Before version 3.0, Python had two kinds of classes: *old-style* and *new-style*.[84] The syntax of both styles is the same, the difference being whether the class `object` is inherited from, directly or indirectly (all new-style classes inherit from `object` and are instances of `type`). In versions of Python 2 from Python 2.2 onwards, both kinds of classes can be used. Old-style classes were eliminated in Python 3.0.

The long term plan is to support gradual typing[85] and from Python 3.5, the syntax of the language allows specifying static types but they are not checked in the default implementation, CPython. An experimental optional static type checker named *mypy* supports compile-time type checking.[86]

Summary of Python 3's built-in types

Type	Mutability	Description	Syntax examples
`bool`	immutable	Boolean value	`True` `False`

`bytearray`	mutable	Sequence of bytes	`bytearray(b'Some ASCII')`
			`bytearray(b"Some ASCII")`
			`bytearray([119, 105, 107, 105])`

`bytes`	immutable	Sequence of bytes	`b'Some ASCII'`
			`b"Some ASCII"`
			`bytes([119, 105, 107, 105])`

`complex`	immutable	Complex number with real and imaginary parts	`3+2.7j`

`dict`	mutable	Associative array (or dictionary) of key and value pairs; can contain mixed types (keys and values), keys must be a hashable type	`{'key1': 1.0, 3: False}`
			`{}`

`ellipsis`[a]	immutable	An ellipsis placeholder to be used as an index in NumPy arrays	`...`
			`Ellipsis`

`float`	immutable	Double precision floating point number. The precision is machine dependent but in practice is 64 bits.	`3.1415927`
`frozenset`	immutable	Unordered set, contains no duplicates; can contain mixed types, if hashable	`frozenset([4.0, 'string', True])`
`int`	immutable	Integer of unlimited magnitude[87]	`42`

`list`	mutable	List, can contain mixed types	`[4.0, 'string', True]`
			`[]`

`NoneType`[a]	immutable	An object representing the absence of a value, often called Null in other languages	`None`
`NotImplementedType`[a]	immutable	A placeholder that can be returned from overloaded operators to indicate unsupported operand types.	`NotImplemented`

`range`	immutable	A Sequence of numbers commonly	`range(1, 10)`
			`range(10, -5, -2)`

		used for looping specific number of times in `for` loops[88]	
`set`	mutable	Unordered set, contains no duplicates; can contain mixed types, if hashable	`{4.0, 'string', True}` `set()`
`str`	immutable	A character string: sequence of Unicode codepoints	`'Wikipedia'` `"Wikipedia"` `"""Spanning multiple lines"""`
`tuple`	immutable	Can contain mixed types	`(4.0, 'string', True)` `('single element',)` `()`

^a Not directly accessible by name

Mathematics

Python has the usual symbols for arithmetic operators (`+`, `-`, `*`, `/`), the floor division operator `//` and the remainder operator `%` (where the remainder can be negative, e.g. `4 % -3 == -2`). It also has `**` for exponentiation, e.g. `5**3 == 125` and `9**0.5 == 3.0`, and a matrix multiply operator `@` .[89] These operators work like in traditional math; with the same precedence rules, the operators infix (`+` and `-` can also be unary to represent positive and negative numbers respectively). Additionally, it has a unary operator (`~`), which essentially inverts all the bits of its one argument. For integers, this means `~x=-x-1`.[90] Other operators include bitwise shift operators `x << y`, which shifts `x` to the left `y` places, the same as `x*(2**y)` , and `x >> y`, which shifts `x` to the right `y` places, the same as `x//(2**y)`.[91]

Division between integers produces floating point results. The behavior of division has changed significantly over time:[92]

- Python 2.1 and earlier used C's division behavior. The `/` operator is integer division if both operands are integers, and floating-point division otherwise. Integer division rounds towards 0, e.g. `7/3 == 2` and `-7/3 == -2`.
- Python 2.2 changed integer division to round towards negative infinity, e.g. `7/3 == 2` and `-7/3 == -3`. The floor division `//` operator was introduced. So `7//3 == 2`, `-7//3 == -3`, `7.5//3 == 2.0` and `-7.5//3 == -3.0`. Adding `from __future__ import division` causes a module to use Python 3.0 rules for division (see next).
- Python 3.0 changed `/` to always be floating-point division, e.g. `5/2 == 2.5`.

In Python terms, `/` is *true division* (or simply *division*), and `//` is *floor division.* `/` before version 3.0 is *classic division*.[92]

Rounding towards negative infinity, though different from most languages, adds consistency. For instance, it means that the equation `(a + b)//b == a//b + 1` is always true. It also means that the equation `b*(a//b) + a%b == a` is valid for both positive and negative values of `a`. However, maintaining the validity of this equation means that while the result of `a%b` is, as expected, in the half-open interval [0, *b*), where `b` is a positive integer, it has to lie in the interval (*b*, 0] when `b` is negative.[93]

Python provides a `round` function for rounding a float to the nearest integer. For tie-breaking, Python 3 uses round to even: `round(1.5)` and `round(2.5)` both produce `2`.[94] Versions before 3 used round-away-from-zero: `round(0.5)` is `1.0`, `round(-0.5)` is `−1.0`.[95]

Python allows boolean expressions with multiple equality relations in a manner that is consistent with general use in mathematics. For example, the expression `a < b < c` tests whether `a` is less than `b` and `b` is less than `c`.[96] C-derived languages interpret this expression differently: in C, the expression would first evaluate `a < b`, resulting in 0 or 1, and that result would then be compared with `c`.[97]

Python uses arbitrary-precision arithmetic for all integer operations. The `Decimal` type/class in the `decimal` module provides decimal floating point numbers to a pre-defined arbitrary precision and several rounding modes.[98] The `Fraction` class in the `fractions` module provides arbitrary precision for rational numbers.[99]

Due to Python's extensive mathematics library, and the third-party library NumPy that further extends the native capabilities, it is frequently used as a scientific scripting language to aid in problems such as numerical data processing and manipulation.[100][101]

Python programming examples

Hello world program:

```python
print('Hello, world!')
```

Program to calculate the factorial of a positive integer:

```python
n = int(input('Type a number, then its factorial will be printed: '))

if n < 0:
    raise ValueError('You must enter a positive number')

fact = 1
i = 2
while i <= n:
    fact = fact * i
    i += 1

print(fact)
```

Libraries

Python's large standard library, commonly cited as one of its greatest strengths,[102] provides tools suited to many tasks. For Internet-facing applications, many standard formats and protocols such as MIME and HTTP are supported. It includes modules for creating graphical user interfaces, connecting to relational databases, generating pseudorandom numbers, arithmetic with arbitrary-precision decimals,[103] manipulating regular expressions, and unit testing.

Some parts of the standard library are covered by specifications (for example, the Web Server Gateway Interface (WSGI) implementation `wsgiref` follows PEP 333[104]), but most modules are not. They are specified by their code, internal documentation, and test suites. However, because most of the standard library is cross-platform Python code, only a few modules need altering or rewriting for variant implementations.

As of November 2019, the Python Package Index (PyPI), the official repository for third-party Python software, contains over 200,000[105] packages with a wide range of functionality, including:

- Graphical user interfaces
- Web frameworks
- Multimedia
- Databases
- Networking
- Test frameworks
- Automation
- Web scraping[106]
- Documentation
- System administration
- Scientific computing
- Text processing
- Image processing
- Machine learning
- Data analytics

Development environments

Most Python implementations (including CPython) include a read–eval–print loop (REPL), permitting them to function as a command line interpreter for which the user enters statements sequentially and receives results immediately.

Other shells, including IDLE and IPython, add further abilities such as improved auto-completion, session state retention and syntax highlighting.

As well as standard desktop integrated development environments, there are Web browser-based IDEs; SageMath (intended for developing science and math-related Python programs); PythonAnywhere, a browser-based IDE and hosting environment; and Canopy IDE, a commercial Python IDE emphasizing scientific computing.[107]

Implementations

Reference implementation

CPython is the reference implementation of Python. It is written in C, meeting the C89 standard with several select C99 features.[108] It compiles Python programs into an intermediate bytecode[109] which is then executed by its virtual machine.[110] CPython is distributed with a large standard library written in a mixture of C and native Python. It is available for many platforms, including Windows and most modern Unix-like systems. Platform portability was one of its earliest priorities.[111]

Other implementations

PyPy is a fast, compliant interpreter of Python 2.7 and 3.6.[112] Its just-in-time compiler brings a significant speed improvement over CPython but several libraries written in C cannot be used with it.[113][114]

Stackless Python is a significant fork of CPython that implements microthreads; it does not use the C memory stack, thus allowing massively concurrent programs. PyPy also has a stackless version.[115]

MicroPython and CircuitPython are Python 3 variants optimized for microcontrollers. This includes Lego Mindstorms EV3.[116]

Unsupported implementations

Other just-in-time Python compilers have been developed, but are now unsupported:

- Google began a project named Unladen Swallow in 2009, with the aim of speeding up the Python interpreter five-fold by using the LLVM, and of improving its multithreading ability to scale to thousands of cores,[117] while ordinary implementations suffer from the global interpreter lock.
- Psyco was a just-in-time specializing compiler that integrates with CPython and transforms bytecode to machine code at runtime. The emitted code is specialized for certain data types and is faster than standard Python code.

In 2005, Nokia released a Python interpreter for the Series 60 mobile phones named PyS60. It includes many of the modules from the CPython implementations and some additional modules to integrate with the Symbian operating system. The project has been kept up-to-date to run on all variants of the S60 platform, and several third-party modules are available. The Nokia N900 also supports Python with GTK widget libraries, enabling programs to be written and run on the target device.[118]

Cross-compilers to other languages

There are several compilers to high-level object languages, with either unrestricted Python, a restricted subset of Python, or a language similar to Python as the source language:

- Jython enables the use of the Java class library from a Python program.
- IronPython follows a similar approach in order to run Python programs on the .NET Common Language Runtime.
- The RPython language can be compiled to C, and is used to build the PyPy interpreter of Python.
- Pyjs compiles Python to JavaScript.
- Cython compiles Python to C and C++.
- Numba uses LLVM to compile Python to machine code.
- Pythran compiles Python to C++.[119][120]
- Somewhat dated Pyrex (latest release in 2010) and Shed Skin (latest release in 2013) compile to C and C++ respectively.
- Google's Grumpy compiles Python to Go.[121][122]
- MyHDL compiles Python to VHDL.
- Nuitka compiles Python into C++.[123]

Performance

A performance comparison of various Python implementations on a non-numerical (combinatorial) workload was presented at EuroSciPy '13.[124]

Development

Python's development is conducted largely through the *Python Enhancement Proposal* (PEP) process, the primary mechanism for proposing major new features, collecting community input on issues and documenting Python design decisions.[125] Python coding style is covered in PEP 8.[126] Outstanding PEPs are reviewed and commented on by the Python community and the steering council.[125]

Enhancement of the language corresponds with development of the CPython reference implementation. The mailing list python-dev is the primary forum for the language's development. Specific issues are discussed in the Roundup bug tracker hosted at bugs.python.org (https://bugs.python.org).[127] Development originally took place on a self-hosted source-code repository running Mercurial, until Python moved to GitHub in January 2017.[128]

CPython's public releases come in three types, distinguished by which part of the version number is incremented:

- Backward-incompatible versions, where code is expected to break and need to be manually ported. The first part of the version number is incremented. These releases happen infrequently—for example, version 3.0 was released 8 years after 2.0.
- Major or "feature" releases, about every 18 months, are largely compatible but introduce new features. The second part of the version number is incremented. Each major version is supported by bugfixes for several years after its release.[129]
- Bugfix releases, which introduce no new features, occur about every 3 months and are made when a sufficient number of bugs have been fixed upstream since the last release. Security vulnerabilities are also patched in these releases. The third and final part of the version number is incremented.[130]

Python 3.9 alpha1 was announced in November 2019[131] and with the adoption of a new yearly release cadence,[132][133] the first release of 3.9 is slated for November 2020.[134]

Many alpha, beta, and release-candidates are also released as previews and for testing before final releases. Although there is a rough schedule for each release, they are often delayed if the code is not ready. Python's development team monitors the state of the code by running the large unit test suite during development, and using the BuildBot continuous integration system.[135]

The community of Python developers has also contributed over 206,000[136] software modules (as of 29 November 2019) to the Python Package Index (PyPI), the official repository of third-party Python libraries.

The major academic conference on Python is PyCon. There are also special Python mentoring programmes, such as Pyladies.

Naming

Python's name is derived from the British comedy group Monty Python, whom Python creator Guido van Rossum enjoyed while developing the language. Monty Python references appear frequently in Python code and culture;[137] for example, the metasyntactic variables often used in Python literature are *spam* and *eggs* instead of the traditional *foo and bar*.[137][138] The official Python documentation also contains various references to Monty Python routines.[139][140]

The prefix *Py-* is used to show that something is related to Python. Examples of the use of this prefix in names of Python applications or libraries include Pygame, a binding of SDL to Python (commonly used to create games); PyQt and PyGTK, which bind Qt and GTK to Python respectively; and PyPy, a Python implementation originally written in Python.

API documentation generators

Python API documentation generators include:

- Sphinx
- Epydoc
- HeaderDoc
- pydoc

Uses

Since 2003, Python has consistently ranked in the top ten most popular programming languages in the TIOBE Programming Community Index where, as of February 2020, it is the third most popular language (behind Java, and C).[141] It was selected Programming Language of the Year in 2007, 2010, and 2018.[142]

An empirical study found that scripting languages, such as Python, are more productive than conventional languages, such as C and Java, for programming problems involving string manipulation and search in a dictionary, and determined that memory consumption was often "better than Java and not much worse than C or C++".[143]

Large organizations that use Python include Wikipedia, Google,[144] Yahoo!,[145] CERN,[146] NASA,[147] Facebook,[148] Amazon, Instagram,[149] Spotify[150] and some smaller entities like ILM[151] and ITA.[152] The social news networking site Reddit is written entirely in Python.[153]

Python can serve as a scripting language for web applications, e.g., via mod_wsgi for the Apache web server.[154] With Web Server Gateway Interface, a standard API has evolved to facilitate these applications. Web frameworks like Django, Pylons, Pyramid, TurboGears, web2py, Tornado, Flask, Bottle and Zope support developers in the design and maintenance of complex applications. Pyjs and IronPython can be used to develop the client-side of Ajax-based applications. SQLAlchemy can be used as data mapper to a relational database. Twisted is a framework to program communications between computers, and is used (for example) by Dropbox.

Libraries such as NumPy, SciPy and Matplotlib allow the effective use of Python in scientific computing,[155][156] with specialized libraries such as Biopython and Astropy providing domain-specific functionality. SageMath is a mathematical software with a notebook interface programmable in Python: its library covers many aspects of mathematics, including algebra, combinatorics, numerical mathematics, number theory, and calculus.

Python has been successfully embedded in many software products as a scripting language, including in finite element method software such as Abaqus, 3D parametric modeler like FreeCAD, 3D animation packages such as 3ds Max, Blender, Cinema 4D, Lightwave, Houdini, Maya, modo, MotionBuilder, Softimage, the visual effects compositor Nuke, 2D imaging programs like GIMP,[157] Inkscape, Scribus and Paint Shop Pro,[158] and musical notation programs like scorewriter and capella. GNU Debugger uses

Python as a pretty printer to show complex structures such as C++ containers. Esri promotes Python as the best choice for writing scripts in ArcGIS.[159] It has also been used in several video games,[160][161] and has been adopted as first of the three available programming languages in Google App Engine, the other two being Java and Go.[162]

Python is commonly used in artificial intelligence projects with the help of libraries like TensorFlow, Keras, Pytorch and Scikit-learn.[163][164][165][166] As a scripting language with modular architecture, simple syntax and rich text processing tools, Python is often used for natural language processing.[167]

Many operating systems include Python as a standard component. It ships with most Linux distributions[168], AmigaOS 4, FreeBSD (as a package), NetBSD, OpenBSD (as a package) and macOS and can be used from the command line (terminal). Many Linux distributions use installers written in Python: Ubuntu uses the Ubiquity installer, while Red Hat Linux and Fedora use the Anaconda installer. Gentoo Linux uses Python in its package management system, Portage.

Python is used extensively in the information security industry, including in exploit development.[169][170]

Most of the Sugar software for the One Laptop per Child XO, now developed at Sugar Labs, is written in Python.[171] The Raspberry Pi single-board computer project has adopted Python as its main user-programming language.

Due to Python's user-friendly conventions and easy-to-understand language, it is commonly used as an intro language into computing sciences with students. This allows students to easily learn computing theories and concepts and then apply them to other programming languages.

LibreOffice includes Python, and intends to replace Java with Python. Its Python Scripting Provider is a core feature[172] since Version 4.0 from 7 February 2013.

Languages influenced by Python

Python's design and philosophy have influenced many other programming languages:

- Boo uses indentation, a similar syntax, and a similar object model.[173]
- Cobra uses indentation and a similar syntax, and its "Acknowledgements" document lists Python first among languages that influenced it.[174] However, Cobra directly supports design-by-contract, unit tests, and optional static typing.[175]
- CoffeeScript, a programming language that cross-compiles to JavaScript, has Python-inspired syntax.
- ECMAScript borrowed iterators and generators from Python.[176]
- Go is designed for the "speed of working in a dynamic language like Python"[177] and shares the same syntax for slicing arrays.
- Groovy was motivated by the desire to bring the Python design philosophy to Java.[178]
- Julia was designed "with true macros [.. and to be] as usable for general programming as Python [and] should be as fast as C".[26] Calling to or from Julia is possible; to with PyCall.jl and a Python package pyjulia allows calling, in the other direction, from Python.
- Kotlin is a functional programming language with an interactive shell similar to Python. However, Kotlin is statically typed with access to standard Java libraries.[179]
- Nim uses indentation and a similar syntax, however it is statically typed, and offers powerful macros.

- Ruby's creator, Yukihiro Matsumoto, has said: "I wanted a scripting language that was more powerful than Perl, and more object-oriented than Python. That's why I decided to design my own language."[180]
- Swift, a programming language developed by Apple, has some Python-inspired syntax.[181]
- GDScript, dynamically typed programming language used to create video-games. It is extremely similar to Python with a few minor differences.

Python's development practices have also been emulated by other languages. For example, the practice of requiring a document describing the rationale for, and issues surrounding, a change to the language (in Python, a PEP) is also used in Tcl[182] and Erlang.[183]

See also

- Python syntax and semantics
- pip (package manager)
- IPython

References

1. Guttag, John V. (12 August 2016). *Introduction to Computation and Programming Using Python: With Application to Understanding Data*. MIT Press. ISBN 978-0-262-52962-4.
2. "Python Release Python 3.8.2" (https://www.python.org/downloads/release/python-382/). *Python.org*.
3. "Python Release Python 3.9.0a5" (https://www.python.org/downloads/release/python-390a5/). *Python.org*.
4. Benjamin (6 April 2020). "Python Insider: Python 2.7.18 release candidate 1 available" (https://pythoninsider.blogspot.com/2020/04/python-2718-release-candidate-1.html). *Python Insider*. Retrieved 14 April 2020. "Python 2.7.18 will be the last release of the Python 2.7 series, and thus Python 2."
5. "Petition: abandon plans to ship a 2.7.18 in April" (https://discuss.python.org/t/petition-abandon-plans-to-ship-a-2-7-18-in-april/2946/4). *Discussions on Python.org*. 3 January 2020. Retrieved 14 April 2020. "Any changes that might have been made since 2.7.17 shipped haven’t yet been released, but as a final service to the community, python-dev will bundle those fixes (and only those fixes) and release a 2.7.18. We plan on doing that in April"
6. "PEP 483 -- The Theory of Type Hints" (https://www.python.org/dev/peps/pep-0483/). *Python.org*.
7. File extension .pyo was removed in Python 3.5. See PEP 0488 (https://www.python.org/dev/peps/pep-0488/)
8. Holth, Moore (30 March 2014). "PEP 0441 -- Improving Python ZIP Application Support" (https://www.python.org/dev/peps/pep-0441/). Retrieved 12 November 2015.
9. "Starlark Language" (https://docs.bazel.build/versions/master/skylark/language.html). Retrieved 25 May 2019.
10. "Why was Python created in the first place?" (https://docs.python.org/faq/general.html#why-was-python-created-in-the-first-place). *General Python FAQ*. Python Software Foundation. Retrieved 22 March 2007.
11. "Ada 83 Reference Manual (raise statement)" (http://archive.adaic.com/standards/83lrm/html/lrm-11-03.html#11.3).

12. Kuchling, Andrew M. (22 December 2006). "Interview with Guido van Rossum (July 1998)" (https://web.archive.org/web/20070501105422/http://www.amk.ca/python/writing/gvr-interview). *amk.ca*. Archived from the original (http://www.amk.ca/python/writing/gvr-interview) on 1 May 2007. Retrieved 12 March 2012.
13. "itertools — Functions creating iterators for efficient looping — Python 3.7.1 documentation" (https://docs.python.org/3/library/itertools.html). *docs.python.org*.
14. van Rossum, Guido (1993). "An Introduction to Python for UNIX/C Programmers". *Proceedings of the NLUUG Najaarsconferentie (Dutch UNIX Users Group)*. CiteSeerX 10.1.1.38.2023 (https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.38.2023). "even though the design of C is far from ideal, its influence on Python is considerable."
15. "Classes" (https://docs.python.org/tutorial/classes.html). *The Python Tutorial*. Python Software Foundation. Retrieved 20 February 2012. "It is a mixture of the class mechanisms found in C++ and Modula-3"
16. Lundh, Fredrik. "Call By Object" (http://effbot.org/zone/call-by-object.htm). *effbot.org*. Retrieved 21 November 2017. "replace "CLU" with "Python", "record" with "instance", and "procedure" with "function or method", and you get a pretty accurate description of Python's object model."
17. Simionato, Michele. "The Python 2.3 Method Resolution Order" (https://www.python.org/download/releases/2.3/mro/). Python Software Foundation. "The C3 method itself has nothing to do with Python, since it was invented by people working on Dylan and it is described in a paper intended for lispers"
18. Kuchling, A. M. "Functional Programming HOWTO" (https://docs.python.org/howto/functional.html). *Python v2.7.2 documentation*. Python Software Foundation. Retrieved 9 February 2012.
19. Schemenauer, Neil; Peters, Tim; Hetland, Magnus Lie (18 May 2001). "PEP 255 – Simple Generators" (https://www.python.org/dev/peps/pep-0255/). *Python Enhancement Proposals*. Python Software Foundation. Retrieved 9 February 2012.
20. Smith, Kevin D.; Jewett, Jim J.; Montanaro, Skip; Baxter, Anthony (2 September 2004). "PEP 318 – Decorators for Functions and Methods" (https://www.python.org/dev/peps/pep-0318/). *Python Enhancement Proposals*. Python Software Foundation. Retrieved 24 February 2012.
21. "More Control Flow Tools" (https://docs.python.org/3.2/tutorial/controlflow.html). *Python 3 documentation*. Python Software Foundation. Retrieved 24 July 2015.
22. "CoffeeScript" (https://coffeescript.org/). *coffeescript.org*.
23. "The Genie Programming Language Tutorial" (https://wiki.gnome.org/action/show/Projects/Genie). Retrieved 28 February 2020.
24. "Perl and Python influences in JavaScript" (http://www.2ality.com/2013/02/javascript-influences.html). *www.2ality.com*. 24 February 2013. Retrieved 15 May 2015.
25. Rauschmayer, Axel. "Chapter 3: The Nature of JavaScript; Influences" (http://speakingjs.com/es5/ch03.html). *O'Reilly, Speaking JavaScript*. Retrieved 15 May 2015.
26. "Why We Created Julia" (https://julialang.org/blog/2012/02/why-we-created-julia). *Julia website*. February 2012. Retrieved 5 June 2014.
27. Ring Team (4 December 2017). "Ring and other languages" (http://ring-lang.sourceforge.net/doc1.6/introduction.html#ring-and-other-languages). *ring-lang.net*. ring-lang.
28. Bini, Ola (2007). *Practical JRuby on Rails Web 2.0 Projects: bringing Ruby on Rails to the Java platform* (https://archive.org/details/practicaljrubyon0000bini/page/3). Berkeley: APress. p. 3 (https://archive.org/details/practicaljrubyon0000bini/page/3). ISBN 978-1-59059-881-8.
29. Lattner, Chris (3 June 2014). "Chris Lattner's Homepage" (http://nondot.org/sabre/). Chris Lattner. Retrieved 3 June 2014. "The Swift language is the product of tireless effort from a team of language experts, documentation gurus, compiler optimization ninjas, and an incredibly important internal dogfooding group who provided feedback to help refine and battle-test ideas. Of course, it also greatly benefited from the experiences hard-won by many other languages in the field, drawing ideas from Objective-C, Rust, Haskell, Ruby, Python, C#, CLU, and far too many others to list."

30. Kuhlman, Dave. "A Python Book: Beginning Python, Advanced Python, and Python Exercises" (https://web.archive.org/web/20120623165941/http://cutter.rexx.com/~dkuhlman/python_book_01.html). Section 1.1. Archived from the original (https://www.davekuhlman.org/python_book_01.pdf) (PDF) on 23 June 2012.
31. "About Python" (https://www.python.org/about). Python Software Foundation. Retrieved 24 April 2012., second section "Fans of Python use the phrase "batteries included" to describe the standard library, which covers everything from asynchronous processing to zip files."
32. "Sunsetting Python 2" (https://www.python.org/doc/sunset-python-2/). *Python.org*. Retrieved 22 September 2019.
33. "PEP 373 -- Python 2.7 Release Schedule" (https://www.python.org/dev/peps/pep-0373/). *Python.org*. Retrieved 22 September 2019.
34. "Python Developer's Guide — Python Developer's Guide" (https://devguide.python.org/#status-of-python-branches). *devguide.python.org*. Retrieved 17 December 2019.
35. "History and License" (https://docs.python.org/3/license.html). Retrieved 5 December 2016. "All Python releases are Open Source"
36. Venners, Bill (13 January 2003). "The Making of Python" (http://www.artima.com/intv/pythonP.html). *Artima Developer*. Artima. Retrieved 22 March 2007.
37. van Rossum, Guido (29 August 2000). "SETL (was: Lukewarm about range literals)" (https://mail.python.org/pipermail/python-dev/2000-August/008881.html). *Python-Dev* (Mailing list). Retrieved 13 March 2011.
38. van Rossum, Guido (20 January 2009). "A Brief Timeline of Python" (https://python-history.blogspot.com/2009/01/brief-timeline-of-python.html). *The History of Python*. Retrieved 20 January 2009.
39. Fairchild, Carlie (12 July 2018). "Guido van Rossum Stepping Down from Role as Python's Benevolent Dictator For Life" (https://www.linuxjournal.com/content/guido-van-rossum-stepping-down-role-pythons-benevolent-dictator-life). *Linux Journal*. Retrieved 13 July 2018.
40. "Guido van Rossum Stepping Down from Role as Python's Benevolent Dictator For Life | Linux Journal" (https://www.linuxjournal.com/content/guido-van-rossum-stepping-down-role-pythons-benevolent-dictator-life). *www.linuxjournal.com*.
41. "Python boss Guido van Rossum steps down after 30 years" (https://www.theinquirer.net/inquirer/news/3035842/python-boss-guido-van-rossum-steps-down-after-30-years). *The Inquirer*.
42. "PEP 8100" (https://www.python.org/dev/peps/pep-8100/). *python*. Python Software Foundation. Retrieved 4 May 2019.
43. "PEP 8100" (https://www.python.org/dev/peps/pep-8100/). Python Software Foundation. Retrieved 4 May 2019.
44. Kuchling, A. M.; Zadka, Moshe (16 October 2000). "What's New in Python 2.0" (https://docs.python.org/whatsnew/2.0.html). Python Software Foundation. Retrieved 11 February 2012.
45. "Python 3.0 Release" (https://www.python.org/download/releases/3.0/). Python Software Foundation. Retrieved 8 July 2009.
46. van Rossum, Guido (5 April 2006). "PEP 3000 – Python 3000" (https://www.python.org/dev/peps/pep-3000/). *Python Enhancement Proposals*. Python Software Foundation. Retrieved 27 June 2009.
47. "Automated Python 2 to 3 code translation — Python Documentation" (https://docs.python.org/3/library/2to3.html). Retrieved 11 February 2018.
48. "PEP 373 -- Python 2.7 Release Schedule" (https://legacy.python.org/dev/peps/pep-0373/). *python.org*. Retrieved 9 January 2017.
49. "PEP 466 -- Network Security Enhancements for Python 2.7.x" (https://www.python.org/dev/peps/pep-0466/). *python.org*. Retrieved 9 January 2017.

50. The Cain Gang Ltd. "Python Metaclasses: Who? Why? When?" (https://web.archive.org/web/20090530030205/http://www.python.org/community/pycon/dc2004/papers/24/metaclasses-pycon.pdf) (PDF). Archived from the original (https://www.python.org/community/pycon/dc2004/papers/24/metaclasses-pycon.pdf) (PDF) on 30 May 2009. Retrieved 27 June 2009.
51. "3.3. Special method names" (https://docs.python.org/3.0/reference/datamodel.html#special-method-names). *The Python Language Reference*. Python Software Foundation. Retrieved 27 June 2009.
52. "PyDBC: method preconditions, method postconditions and class invariants for Python" (http://www.nongnu.org/pydbc/). Retrieved 24 September 2011.
53. "Contracts for Python" (http://www.wayforward.net/pycontract/). Retrieved 24 September 2011.
54. "PyDatalog" (https://sites.google.com/site/pydatalog/). Retrieved 22 July 2012.
55. Hettinger, Raymond (30 January 2002). "PEP 289 – Generator Expressions" (https://www.python.org/dev/peps/pep-0289/). *Python Enhancement Proposals*. Python Software Foundation. Retrieved 19 February 2012.
56. "6.5 itertools – Functions creating iterators for efficient looping" (https://docs.python.org/3/library/itertools.html). Docs.python.org. Retrieved 22 November 2016.
57. Peters, Tim (19 August 2004). "PEP 20 – The Zen of Python" (https://www.python.org/dev/peps/pep-0020/). *Python Enhancement Proposals*. Python Software Foundation. Retrieved 24 November 2008.
58. Martelli, Alex; Ravenscroft, Anna; Ascher, David (2005). *Python Cookbook, 2nd Edition* (http://shop.oreilly.com/product/9780596007973.do). O'Reilly Media. p. 230. ISBN 978-0-596-00797-3.
59. "Ebeab.com" (http://ebeab.com/2014/01/21/python-culture/). *ebeab.com*.
60. "General Python FAQ" (https://docs.python.org/2/faq/general.html#why-is-it-called-python). *Python v2.7.3 documentation*. Docs.python.org. Retrieved 3 December 2012.
61. "15 Ways Python Is a Powerful Force on the Web" (https://insidetech.monster.com/training/articles/8114-15-ways-python-is-a-powerful-force-on-the-web).
62. "8.18. pprint — Data pretty printer — Python 2.7.18rc1 documentation" (https://docs.python.org/2/library/pprint.html). *docs.python.org*.
63. Goodger, David. "Code Like a Pythonista: Idiomatic Python" (http://python.net/~goodger/projects/pycon/2007/idiomatic/handout.html).
64. "How to think like a Pythonista" (http://python.net/crew/mwh/hacks/objectthink.html).
65. "Is Python a good language for beginning programmers?" (https://docs.python.org/faq/general.html#is-python-a-good-language-for-beginning-programmers). *General Python FAQ*. Python Software Foundation. Retrieved 21 March 2007.
66. "Myths about indentation in Python" (http://www.secnetix.de/~olli/Python/block_indentation.hawk). Secnetix.de. Retrieved 19 April 2011.
67. "Python 2.5 Release" (https://www.python.org/download/releases/2.5/). *Python.org*.
68. "Highlights: Python 2.5" (https://www.python.org/download/releases/2.5/highlights/). *Python.org*.
69. van Rossum, Guido (22 April 2009). "Tail Recursion Elimination" (http://neopythonic.blogspot.be/2009/04/tail-recursion-elimination.html). Neopythonic.blogspot.be. Retrieved 3 December 2012.
70. van Rossum, Guido (9 February 2006). "Language Design Is Not Just Solving Puzzles" (http://www.artima.com/weblogs/viewpost.jsp?thread=147358). *Artima forums*. Artima. Retrieved 21 March 2007.
71. van Rossum, Guido; Eby, Phillip J. (10 May 2005). "PEP 342 – Coroutines via Enhanced Generators" (https://www.python.org/dev/peps/pep-0342/). *Python Enhancement Proposals*. Python Software Foundation. Retrieved 19 February 2012.

72. "PEP 380" (https://www.python.org/dev/peps/pep-0380/). Python.org. Retrieved 3 December 2012.
73. "division" (https://docs.python.org). *python.org*.
74. "PEP 0465 -- A dedicated infix operator for matrix multiplication" (https://www.python.org/dev/peps/pep-0465/). *python.org*. Retrieved 1 January 2016.
75. "Python 3.5.1 Release and Changelog" (https://www.python.org/downloads/release/python-351/). *python.org*. Retrieved 1 January 2016.
76. "What's New In Python 3.8" (https://docs.python.org/3.8/whatsnew/3.8.html). Retrieved 14 October 2019.
77. "Chapter 15. Expressions - 15.21.1. Numerical Equality Operators == and !=" (https://docs.oracle.com/javase/specs/jls/se8/html/jls-15.html#jls-15.21.1). Oracle Corporation. Retrieved 28 August 2016.
78. "Chapter 15. Expressions - 15.21.3. Reference Equality Operators == and !=" (https://docs.oracle.com/javase/specs/jls/se8/html/jls-15.html#jls-15.21.3). Oracle Corporation. Retrieved 28 August 2016.
79. van Rossum, Guido; Hettinger, Raymond (7 February 2003). "PEP 308 – Conditional Expressions" (https://www.python.org/dev/peps/pep-0308/). *Python Enhancement Proposals*. Python Software Foundation. Retrieved 13 July 2011.
80. "4. Built-in Types — Python 3.6.3rc1 documentation" (https://docs.python.org/3/library/stdtypes.html#tuple). *python.org*. Retrieved 1 October 2017.
81. "5.3. Tuples and Sequences — Python 3.7.1rc2 documentation" (https://docs.python.org/3/tutorial/datastructures.html#tuples-and-sequences). *python.org*. Retrieved 17 October 2018.
82. "PEP 498 -- Literal String Interpolation" (https://www.python.org/dev/peps/pep-0498/). *python.org*. Retrieved 8 March 2017.
83. "Why must 'self' be used explicitly in method definitions and calls?" (https://docs.python.org/faq/design.html#why-must-self-be-used-explicitly-in-method-definitions-and-calls). *Design and History FAQ*. Python Software Foundation. Retrieved 19 February 2012.
84. "The Python Language Reference, section 3.3. New-style and classic classes, for release 2.7.1" (https://docs.python.org/reference/datamodel.html#new-style-and-classic-classes). Retrieved 12 January 2011.
85. "Type hinting for Python" (https://lwn.net/Articles/627418/). LWN.net. 24 December 2014. Retrieved 5 May 2015.
86. "mypy - Optional Static Typing for Python" (http://mypy-lang.org/). Retrieved 28 January 2017.
87. Zadka, Moshe; van Rossum, Guido (11 March 2001). "PEP 237 – Unifying Long Integers and Integers" (https://www.python.org/dev/peps/pep-0237/). *Python Enhancement Proposals*. Python Software Foundation. Retrieved 24 September 2011.
88. "Built-in Types" (https://docs.python.org/3/library/stdtypes.html#typesseq-range). Retrieved 3 October 2019.
89. "PEP 465 -- A dedicated infix operator for matrix multiplication" (https://legacy.python.org/dev/peps/pep-0465/). *python.org*.
90. "The tilde operator in Python - Stackoverflow" (https://stackoverflow.com/questions/8305199/the-tilde-operator-in-python). *stackoverflow.com*.
91. "BitwiseOperators - Python Wiki" (https://wiki.python.org/moin/BitwiseOperators). *wiki.python.org*.
92. Zadka, Moshe; van Rossum, Guido (11 March 2001). "PEP 238 – Changing the Division Operator" (https://www.python.org/dev/peps/pep-0238/). *Python Enhancement Proposals*. Python Software Foundation. Retrieved 23 October 2013.
93. "Why Python's Integer Division Floors" (https://python-history.blogspot.com/2010/08/why-pythons-integer-division-floors.html). Retrieved 25 August 2010.

94. "round" (https://docs.python.org/py3k/library/functions.html#round), *The Python standard library, release 3.2, §2: Built-in functions*, retrieved 14 August 2011
95. "round" (https://docs.python.org/library/functions.html#round), *The Python standard library, release 2.7, §2: Built-in functions*, retrieved 14 August 2011
96. Beazley, David M. (2009). *Python Essential Reference* (4th ed.). p. 66.
97. Kernighan, Brian W.; Ritchie, Dennis M. (1988). *The C Programming Language* (2nd ed.). p. 206 (https://archive.org/details/cprogramminglang00bria/page/206).
98. Batista, Facundo. "PEP 0327 -- Decimal Data Type" (https://www.python.org/dev/peps/pep-0327/). *Python.org*. Retrieved 26 September 2015.
99. "What's New in Python 2.6 — Python v2.6.9 documentation" (https://docs.python.org/2.6/whatsnew/2.6.html). *docs.python.org*. Retrieved 26 September 2015.
00. "10 Reasons Python Rocks for Research (And a Few Reasons it Doesn't) — Hoyt Koepke" (https://www.stat.washington.edu/~hoytak/blog/whypython.html). *www.stat.washington.edu*. Retrieved 3 February 2019.
01. Shell, Scott (17 June 2014). "An introduction to Python for scientific computing" (https://engineering.ucsb.edu/~shell/che210d/python.pdf) (PDF). Retrieved 3 February 2019.
02. Piotrowski, Przemyslaw (July 2006). "Build a Rapid Web Development Environment for Python Server Pages and Oracle" (http://www.oracle.com/technetwork/articles/piotrowski-pythoncore-084049.html). *Oracle Technology Network*. Oracle. Retrieved 12 March 2012.
03. Batista, Facundo (17 October 2003). "PEP 327 – Decimal Data Type" (https://www.python.org/dev/peps/pep-0327/). *Python Enhancement Proposals*. Python Software Foundation. Retrieved 24 November 2008.
04. Eby, Phillip J. (7 December 2003). "PEP 333 – Python Web Server Gateway Interface v1.0" (https://www.python.org/dev/peps/pep-0333/). *Python Enhancement Proposals*. Python Software Foundation. Retrieved 19 February 2012.
05. Debill, Erik. "Module Counts" (http://www.modulecounts.com/). *ModuleCounts*. Retrieved 5 November 2019.
06. "20+ Python Web Scraping Examples (Beautiful Soup & Selenium) - Like Geeks" (https://likegeeks.com/python-web-scraping/). *likegeeks.com*. 5 December 2017. Retrieved 12 March 2018.
07. Enthought, Canopy. "Canopy" (https://www.enthought.com/products/canopy/). *www.enthought.com*. Retrieved 20 August 2016.
08. van Rossum, Guido (5 June 2001). "PEP 7 – Style Guide for C Code" (https://www.python.org/dev/peps/pep-0007/). *Python Enhancement Proposals*. Python Software Foundation. Retrieved 24 November 2008.
09. "CPython byte code" (https://docs.python.org/3/library/dis.html#python-bytecode-instructions). Docs.python.org. Retrieved 16 February 2016.
10. "Python 2.5 internals" (http://www.troeger.eu/teaching/pythonvm08.pdf) (PDF). Retrieved 19 April 2011.
11. "An Interview with Guido van Rossum" (http://www.oreilly.com/pub/a/oreilly/frank/rossum_1099.html). Oreilly.com. Retrieved 24 November 2008.
12. "PyPy compatibility" (https://pypy.org/compat.html). Pypy.org. Retrieved 3 December 2012.
13. "speed comparison between CPython and Pypy" (http://speed.pypy.org/). Speed.pypy.org. Retrieved 3 December 2012.
14. Shaw, Anthony (30 March 2018). "Which is the fastest version of Python?" (https://hackernoon.com/which-is-the-fastest-version-of-python-2ae7c61a6b2b). Hacker Noon. Retrieved 20 December 2019.
15. "Application-level Stackless features — PyPy 2.0.2 documentation" (http://doc.pypy.org/en/latest/stackless.html). Doc.pypy.org. Retrieved 17 July 2013.
16. "Python-for-EV3" (https://education.lego.com/en-us/support/mindstorms-ev3/python-for-ev3). *LEGO Education*. Retrieved 17 April 2019.

17. "Plans for optimizing Python" (https://code.google.com/p/unladen-swallow/wiki/ProjectPlan). *Google Project Hosting*. 15 December 2009. Retrieved 24 September 2011.
18. "Python on the Nokia N900" (http://www.stochasticgeometry.ie/2010/04/29/python-on-the-nokia-n900/). *Stochastic Geometry*. 29 April 2010.
19. Borderies, Olivier (24 January 2019). "Pythran: Python at C++ speed !" (https://medium.com/@olivier.borderies/pythran-python-at-c-speed-518f26af60e8). *Medium*.
20. "Pythran — Pythran 0.9.5 documentation" (https://pythran.readthedocs.io/en/latest/). *pythran.readthedocs.io*.
21. "google/grumpy" (https://github.com/google/grumpy). 10 April 2020 – via GitHub.
22. "Projects" (https://opensource.google/projects/). *opensource.google*.
23. "Nuitka Home | Nuitka Home" (http://nuitka.net/). *nuitka.net*. Retrieved 18 August 2017.
24. Murri, Riccardo (2013). *Performance of Python runtimes on a non-numeric scientific code*. European Conference on Python in Science (EuroSciPy). arXiv:1404.6388 (https://arxiv.org/abs/1404.6388). Bibcode:2014arXiv1404.6388M (https://ui.adsabs.harvard.edu/abs/2014arXiv1404.6388M).
25. Warsaw, Barry; Hylton, Jeremy; Goodger, David (13 June 2000). "PEP 1 – PEP Purpose and Guidelines" (https://www.python.org/dev/peps/pep-0001/). *Python Enhancement Proposals*. Python Software Foundation. Retrieved 19 April 2011.
26. "PEP 8 -- Style Guide for Python Code" (https://www.python.org/dev/peps/pep-0008/). *Python.org*.
27. Cannon, Brett. "Guido, Some Guys, and a Mailing List: How Python is Developed" (https://web.archive.org/web/20090601134342/http://www.python.org/dev/intro/). *python.org*. Python Software Foundation. Archived from the original (https://www.python.org/dev/intro/) on 1 June 2009. Retrieved 27 June 2009.
28. "Python Developer's Guide" (https://docs.python.org/devguide/).
29. Norwitz, Neal (8 April 2002). "[Python-Dev] Release Schedules (was Stability & change)" (https://mail.python.org/pipermail/python-dev/2002-April/022739.html). Retrieved 27 June 2009.
30. Aahz; Baxter, Anthony (15 March 2001). "PEP 6 – Bug Fix Releases" (https://www.python.org/dev/peps/pep-0006/). *Python Enhancement Proposals*. Python Software Foundation. Retrieved 27 June 2009.
31. "What's New In Python 3.9" (https://docs.python.org/3.9/whatsnew/3.9.html). *Python*. Retrieved 28 November 2019.
32. "PEP 602 -- Annual Release Cycle for Python" (https://www.python.org/dev/peps/pep-0602/). *Python.org*. Retrieved 6 November 2019.
33. "Changing the Python release cadence [LWN.net]" (https://lwn.net/Articles/802777/). *lwn.net*. Retrieved 6 November 2019.
34. "PEP 596 -- Python 3.9 Release Schedule" (https://www.python.org/dev/peps/pep-0596/). *Python.org*. Retrieved 6 November 2019.
35. "Python Buildbot" (https://www.python.org/dev/buildbot/). *Python Developer's Guide*. Python Software Foundation. Retrieved 24 September 2011.
36. DeBill, Erik. "Module Counts" (http://www.modulecounts.com/#). *www.modulecounts.com*. Retrieved 29 November 2019.
37. "Whetting Your Appetite" (https://docs.python.org/tutorial/appetite.html). *The Python Tutorial*. Python Software Foundation. Retrieved 20 February 2012.
38. "In Python, should I use else after a return in an if block?" (https://stackoverflow.com/questions/5033906/in-python-should-i-use-else-after-a-return-in-an-if-block). *Stack Overflow*. Stack Exchange. 17 February 2011. Retrieved 6 May 2011.
39. Lutz, Mark (2009). *Learning Python: Powerful Object-Oriented Programming* (https://books.google.com/books?id=1HxWGezDZcgC&pg=PA17). O'Reilly Media, Inc. p. 17. ISBN 9781449379322.

40. Fehily, Chris (2002). *Python* (https://books.google.com/books?id=carqdIdfVlYC&pg=PR15). Peachpit Press. p. xv. ISBN 9780201748840.
41. "TIOBE Index" (http://www.tiobe.com/tiobe-index/). TIOBE - The Software Quality Company. Retrieved 7 March 2017.
42. TIOBE Software Index (2015). "TIOBE Programming Community Index Python" (http://www.tiobe.com/index.php/paperinfo/tpci/Python.html). Retrieved 10 September 2015.
43. Prechelt, Lutz (14 March 2000). "An empirical comparison of C, C++, Java, Perl, Python, Rexx, and Tcl" (http://page.mi.fu-berlin.de/prechelt/Biblio/jccpprt_computer2000.pdf) (PDF). Retrieved 30 August 2013.
44. "Quotes about Python" (https://www.python.org/about/quotes/). Python Software Foundation. Retrieved 8 January 2012.
45. "Organizations Using Python" (https://wiki.python.org/moin/OrganizationsUsingPython). Python Software Foundation. Retrieved 15 January 2009.
46. "Python : the holy grail of programming" (http://cdsweb.cern.ch/journal/CERNBulletin/2006/31/News%20Articles/974627?ln=en). *CERN Bulletin*. CERN Publications (31/2006). 31 July 2006. Retrieved 11 February 2012.
47. Shafer, Daniel G. (17 January 2003). "Python Streamlines Space Shuttle Mission Design" (https://www.python.org/about/success/usa/). Python Software Foundation. Retrieved 24 November 2008.
48. "Tornado: Facebook's Real-Time Web Framework for Python - Facebook for Developers" (https://developers.facebook.com/blog/post/301). *Facebook for Developers*. Retrieved 19 June 2018.
49. "What Powers Instagram: Hundreds of Instances, Dozens of Technologies" (https://instagram-engineering.com/what-powers-instagram-hundreds-of-instances-dozens-of-technologies-adf2e22da2ad). Instagram Engineering. Retrieved 27 May 2019.
50. "How we use Python at Spotify" (https://labs.spotify.com/2013/03/20/how-we-use-python-at-spotify/). *Spotify Labs*. 20 March 2013. Retrieved 25 July 2018.
51. Fortenberry, Tim (17 January 2003). "Industrial Light & Magic Runs on Python" (https://www.python.org/about/success/ilm/). Python Software Foundation. Retrieved 11 February 2012.
52. Taft, Darryl K. (5 March 2007). "Python Slithers into Systems" (http://www.eweek.com/c/a/Application-Development/Python-Slithers-into-Systems/). *eWeek.com*. Ziff Davis Holdings. Retrieved 24 September 2011.
53. *GitHub - reddit-archive/reddit: historical code from reddit.com.* (https://github.com/reddit-archive/reddit), The Reddit Archives, 19 March 2019, retrieved 20 March 2019
54. "Usage statistics and market share of Python for websites" (http://w3techs.com/technologies/details/pl-python/all/all). 2012. Retrieved 18 December 2012.
55. Oliphant, Travis (2007). "Python for Scientific Computing" (https://www.h2desk.com/blog/python-scientific-computing/). *Computing in Science and Engineering*. **9** (3): 10–20. Bibcode:2007CSE.....9c..10O (https://ui.adsabs.harvard.edu/abs/2007CSE.....9c..10O). CiteSeerX 10.1.1.474.6460 (https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.474.6460). doi:10.1109/MCSE.2007.58 (https://doi.org/10.1109%2FMCSE.2007.58).
56. Millman, K. Jarrod; Aivazis, Michael (2011). "Python for Scientists and Engineers" (http://www.computer.org/csdl/mags/cs/2011/02/mcs2011020009.html). *Computing in Science and Engineering*. **13** (2): 9–12. Bibcode:2011CSE....13b...9M (https://ui.adsabs.harvard.edu/abs/2011CSE....13b...9M). doi:10.1109/MCSE.2011.36 (https://doi.org/10.1109%2FMCSE.2011.36).
57. "Installers for GIMP for Windows - Frequently Asked Questions" (https://web.archive.org/web/20130717070814/http://gimp-win.sourceforge.net/faq.html). 26 July 2013. Archived from the original (http://gimp-win.sourceforge.net/faq.html) on 17 July 2013. Retrieved 26 July 2013.
58. "jasc psp9components" (https://web.archive.org/web/20080319061519/http://www.jasc.com/support/customercare/articles/psp9components.asp). Archived from the original (http://www.jasc.com/support/customercare/articles/psp9components.asp) on 19 March 2008.

59. "About getting started with writing geoprocessing scripts" (http://webhelp.esri.com/arcgisdesktop/9.2/index.cfm?TopicName=About_getting_started_with_writing_geoprocessing_scripts). *ArcGIS Desktop Help 9.2*. Environmental Systems Research Institute. 17 November 2006. Retrieved 11 February 2012.
60. CCP porkbelly (24 August 2010). "Stackless Python 2.7" (https://community.eveonline.com/news/dev-blogs/stackless-python-2.7/). *EVE Community Dev Blogs*. CCP Games. "As you may know, EVE has at its core the programming language known as Stackless Python."
61. Caudill, Barry (20 September 2005). "Modding Sid Meier's Civilization IV" (https://web.archive.org/web/20101202164144/http://www.2kgames.com/civ4/blog_03.htm). *Sid Meier's Civilization IV Developer Blog*. Firaxis Games. Archived from the original (http://www.2kgames.com/civ4/blog_03.htm) on 2 December 2010. "we created three levels of tools ... The next level offers Python and XML support, letting modders with more experience manipulate the game world and everything in it."
62. "Python Language Guide (v1.0)" (https://web.archive.org/web/20100715145616/http://code.google.com/apis/documents/docs/1.0/developers_guide_python.html). *Google Documents List Data API v1.0*. Archived from the original (https://code.google.com/apis/documents/docs/1.0/developers_guide_python.html) on 15 July 2010.
63. Dean, Jeff; Monga, Rajat; et al. (9 November 2015). "TensorFlow: Large-scale machine learning on heterogeneous systems" (http://download.tensorflow.org/paper/whitepaper2015.pdf) (PDF). *TensorFlow.org*. Google Research. Retrieved 10 November 2015.
64. Piatetsky, Gregory. "Python eats away at R: Top Software for Analytics, Data Science, Machine Learning in 2018: Trends and Analysis" (https://www.kdnuggets.com/2018/05/poll-tools-analytics-data-science-machine-learning-results.html/2). *KDnuggets*. KDnuggets. Retrieved 30 May 2018.
65. "Who is using scikit-learn? — scikit-learn 0.20.1 documentation" (https://scikit-learn.org/stable/testimonials/testimonials.html). *scikit-learn.org*.
66. Jouppi, Norm. "Google supercharges machine learning tasks with TPU custom chip" (https://cloudplatform.googleblog.com/2016/05/Google-supercharges-machine-learning-tasks-with-custom-chip.html). *Google Cloud Platform Blog*. Retrieved 19 May 2016.
67. "Natural Language Toolkit — NLTK 3.5b1 documentation" (http://www.nltk.org/). *www.nltk.org*.
68. "Python Setup and Usage" (https://docs.python.org/3/using/unix.html). Python Software Foundation. Retrieved 10 January 2020.
69. "Immunity: Knowing You're Secure" (https://web.archive.org/web/20090216134332/http://immunitysec.com/products-immdbg.shtml). Archived from the original (http://www.immunitysec.com/products-immdbg.shtml) on 16 February 2009.
70. "Core Security" (https://www.coresecurity.com/). *Core Security*.
71. "What is Sugar?" (http://sugarlabs.org/go/Sugar). Sugar Labs. Retrieved 11 February 2012.
72. "4.0 New Features and Fixes" (http://www.libreoffice.org/download/4-0-new-features-and-fixes/). *LibreOffice.org*. The Document Foundation. 2013. Retrieved 25 February 2013.
73. "Gotchas for Python Users" (https://web.archive.org/web/20081211062108/http://boo.codehaus.org/Gotchas+for+Python+Users). *boo.codehaus.org*. Codehaus Foundation. Archived from the original (http://boo.codehaus.org/Gotchas+for+Python+Users) on 11 December 2008. Retrieved 24 November 2008.
74. Esterbrook, Charles. "Acknowledgements" (http://cobra-language.com/docs/acknowledgements/). *cobra-language.com*. Cobra Language. Retrieved 7 April 2010.
75. Esterbrook, Charles. "Comparison to Python" (http://cobra-language.com/docs/python/). *cobra-language.com*. Cobra Language. Retrieved 7 April 2010.
76. "Proposals: iterators and generators [ES4 Wiki]" (https://web.archive.org/web/20071020082650/http://wiki.ecmascript.org/doku.php?id=proposals:iterators_and_generators). wiki.ecmascript.org. Archived from the original (http://wiki.ecmascript.org/doku.php?id=proposals:iterators_and_generators) on 20 October 2007. Retrieved 24 November 2008.

77. Kincaid, Jason (10 November 2009). "Google's Go: A New Programming Language That's Python Meets C++" (https://techcrunch.com/2009/11/10/google-go-language/). *TechCrunch*. Retrieved 29 January 2010.
78. Strachan, James (29 August 2003). "Groovy – the birth of a new dynamic language for the Java platform" (http://radio.weblogs.com/0112098/2003/08/29.html).
79. "Working with the Command Line Compiler - Kotlin Programming Language" (https://kotlinlang.org/docs/tutorials/command-line.html). *Kotlin*. Retrieved 12 March 2018.
80. "An Interview with the Creator of Ruby" (http://www.linuxdevcenter.com/pub/a/linux/2001/11/29/ruby.html). Linuxdevcenter.com. Retrieved 3 December 2012.
81. Lattner, Chris (3 June 2014). "Chris Lattner's Homepage" (http://nondot.org/sabre). Chris Lattner. Retrieved 3 June 2014. "I started work on the Swift Programming Language in July of 2010. I implemented much of the basic language structure, with only a few people knowing of its existence. A few other (amazing) people started contributing in earnest late in 2011, and it became a major focus for the Apple Developer Tools group in July 2013 [...] drawing ideas from Objective-C, Rust, Haskell, Ruby, Python, C#, CLU, and far too many others to list."
82. Kupries, Andreas; Fellows, Donal K. (14 September 2000). "TIP #3: TIP Format" (http://www.tcl.tk/cgi-bin/tct/tip/3.html). *tcl.tk*. Tcl Developer Xchange. Retrieved 24 November 2008.
83. Gustafsson, Per; Niskanen, Raimo (29 January 2007). "EEP 1: EEP Purpose and Guidelines" (http://www.erlang.org/eeps/eep-0001.html). erlang.org. Retrieved 19 April 2011.

Sources

- "Python for Artificial Intelligence" (https://web.archive.org/web/20121101045354/http://wiki.python.org/moin/PythonForArtificialIntelligence). Wiki.python.org. 19 July 2012. Archived from the original (https://wiki.python.org/moin/PythonForArtificialIntelligence) on 1 November 2012. Retrieved 3 December 2012.
- Paine, Jocelyn, ed. (August 2005). "AI in Python" (http://www.ainewsletter.com/newsletters/aix_0508.htm#python_ai_ai). *AI Expert Newsletter*. Amzi!. Retrieved 11 February 2012.
- "PyAIML 0.8.5 : Python Package Index" (https://pypi.python.org/pypi/PyAIML). Pypi.python.org. Retrieved 17 July 2013.
- Russell, Stuart J. & Norvig, Peter (2009). *Artificial Intelligence: A Modern Approach* (3rd ed.). Upper Saddle River, NJ: Prentice Hall. ISBN 978-0-13-604259-4.

Further reading

- Downey, Allen B. (May 2012). *Think Python: How to Think Like a Computer Scientist* (Version 1.6.6 ed.). ISBN 978-0-521-72596-5.
- Hamilton, Naomi (5 August 2008). "The A-Z of Programming Languages: Python" (https://web.archive.org/web/20081229095320/http://www.computerworld.com.au/index.php/id%3B66665771). *Computerworld*. Archived from the original (http://www.computerworld.com.au/index.php/id;66665771) on 29 December 2008. Retrieved 31 March 2010.
- Lutz, Mark (2013). *Learning Python* (5th ed.). O'Reilly Media. ISBN 978-0-596-15806-4.
- Pilgrim, Mark (2004). *Dive Into Python* (https://archive.org/details/diveintopython0000pilg). Apress. ISBN 978-1-59059-356-1.
- Pilgrim, Mark (2009). *Dive Into Python 3*. Apress. ISBN 978-1-4302-2415-0.
- Summerfield, Mark (2009). *Programming in Python 3* (2nd ed.). Addison-Wesley Professional. ISBN 978-0-321-68056-3.

External links

- Official website (https://www.python.org/)
- Python (programming language) (https://curlie.org/Computers/Programming/Languages/Python) at Curlie