WIKIPEDIA
The Free Encyclopedia

Python (programming language)

Python is a high-level, general-purpose programming language. Its design philosophy emphasizes code readability with the use of significant indentation.[34]

Python is dynamically type-checked and garbage-collected. It supports multiple programming paradigms, including structured (particularly procedural), object-oriented and functional programming.

Guido van Rossum began working on Python in the late 1980s as a successor to the ABC programming language. Python 3.0, released in 2008, was a major revision and not completely backward-compatible with earlier versions. Beginning with Python 3.5,[35] capabilities and keywords for typing were added to language, allowing optional static typing.[36] Currently only versions in the 3.x series are supported.

Python consistently ranks as one of the most popular programming languages, and it has gained widespread use in the machine learning community.[37][38][39][40] It is widely taught as an introductory programming language.[41]

Python

Paradigm	Multi-paradigm: object-oriented,[1] procedural (imperative), functional, structured, reflective
Designed by	Guido van Rossum
Developer	Python Software Foundation
First appeared	20 February 1991[2]
Stable release	3.14.0[3] / 7 October 2025
Typing discipline	Duck, dynamic, strong;[4] optional type annotations[a]
OS	Cross-platform including e.g. for mobile/Android[b]
License	Python Software Foundation License
Filename extensions	.py, .pyw, .pyz,[11] .pyi, .pyc, .pyd
Website	python.org (https://www.python.org/)

Major implementations

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

Dialects

Cython, RPython, Starlark[12]

Influenced by

ABC,[13] Ada,[14] ALGOL 68,[15] APL,[16] C,[17] C++,[18] CLU,[19] Dylan,[20] Haskell,[21][16] Icon,[22] Lisp,[23] Modula-3,[15][18] Perl,[24] Standard ML[16]

History

Python was conceived in the late 1980s[42] by Guido van Rossum at Centrum Wiskunde & Informatica (CWI) in the Netherlands.[43] It was designed as a successor to the ABC programming language, which was inspired by SETL,[44] capable of exception handling and interfacing with the Amoeba operating system.[13] Python implementation began in December 1989.[43] Van Rossum first released it in 1991 as Python 0.9.0.[43] Van Rossum assumed sole responsibility for the project, as the lead developer, until 12 July 2018, when he announced his "permanent vacation" from responsibilities as Python's "benevolent dictator for life" (BDFL); this title was bestowed on him by the Python community to reflect his long-term commitment as the project's chief decision-maker.[45] (He has since come out of retirement and is self-titled "BDFL-emeritus".) In January 2019, active Python core developers elected a five-member Steering Council to lead the project.[46][47]

Influenced
Apache Groovy, Boo, Cobra, CoffeeScript,[25] D, F#, GDScript, Go, JavaScript,[26][27] Julia,[28] Mojo,[29] Nim, Ring,[30] Ruby,[31] Swift,[32] V[33]
Python Programming at Wikibooks

The name *Python* derives from the British comedy series Monty Python's Flying Circus.[48] (See § Naming.)

The designer of Python, Guido van Rossum, at PyCon US 2024

Python 2.0 was released on 16 October 2000, featuring many new features such as list comprehensions, cycle-detecting garbage collection, reference counting, and Unicode support.[49] Python 2.7's end-of-life was initially set for 2015, and then postponed to 2020 out of concern that a large body of existing code could not easily be forward-ported to Python 3.[50][51] It no longer receives security patches or updates.[52][53] While Python 2.7 and older versions are officially unsupported, a different unofficial Python implementation, PyPy, continues to support Python 2, i.e., "2.7.18+" (plus 3.10), with the plus signifying (at least some) "backported security updates".[54]

Python 3.0 was released on 3 December 2008, and was a major revision and not completely backward-compatible with earlier versions, with some new semantics and changed syntax. Python 2.7.18, released in 2020, was the last release of Python 2.[55] Several releases in the Python 3.x series have added new syntax to the language, and made a few (considered very minor) backward-incompatible changes.

Since 7 October 2025, Python 3.14.0 is the latest stable release, and Python 3.13.9 released a week later all older 3.x versions also had a security update down to Python 3.9.24, the last in 3.9 series. Python 3.10 is the oldest supported branch.[56] Releases receive two years of full support followed by three years of security support.

Design philosophy and features

Python is a multi-paradigm programming language. Object-oriented programming and structured programming are fully supported, and many of their features support functional programming and aspect-oriented programming – including metaprogramming[57] and metaobjects.[58] Many other paradigms are supported via extensions, including design by contract[59][60] and logic programming.[61] Python is often referred to as a *'glue language'*[62] because it is purposely designed to be able to integrate components written in other languages.

Python uses dynamic typing and a combination of reference counting and a cycle-detecting garbage collector for memory management.[63] It uses 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.[64] The standard library has two modules (`itertools` and `functools`) that implement functional tools borrowed from Haskell and Standard ML.[65]

Python's core philosophy is summarized in the Zen of Python (PEP 20) written by Tim Peters, which includes aphorisms such as these:[66]

- Explicit is better than implicit.
- Simple is better than complex.
- Readability counts.
- Special cases aren't special enough to break the rules.
- Although practicality beats purity.
- Errors should never pass silently.
- Unless explicitly silenced.
- There should be one-- and preferably only one --obvious way to do it.

However, Python have received criticism for violating these principles and adding unnecessary language bloat.[67] Responses to these criticisms note that the Zen of Python is a guideline rather than a rule.[68] The addition of some new features had been controversial: Guido van Rossum resigned as *Benevolent Dictator for Life* after conflict about adding the assignment expression operator in Python 3.8 .[69][70]

Nevertheless, rather than building all functionality into its core, Python was designed to be highly extensible via modules. 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 represented the opposite approach.[42]

Python claims to strive for a simpler, less-cluttered syntax and grammar, while giving developers a choice in their coding methodology. In contrast to Perl's motto "there is more than one way to do it", Python advocates an approach where "there should be one – and preferably only one – obvious way to do it".[66] In practice, however, Python provides many ways to achieve a given goal. There are at least three ways to format a string literal, with no certainty as to which one a programmer should use.[71] Alex Martelli is a Fellow at the Python Software Foundation and Python book author; he wrote that "To describe something as 'clever' is *not* considered a compliment in the Python culture."[72]

Python's developers usually try to avoid premature optimization; they also reject patches to non-critical parts of the CPython reference implementation that would offer marginal increases in speed at the cost of clarity.[73] Execution speed can be improved by moving speed-critical functions to extension modules written in languages such as C, or by using a just-in-time compiler like PyPy. It is also possible to cross-compile to other languages; but this approach either fails to achieve the expected speed-up, since Python is a very dynamic language, or only a restricted subset of Python is compiled (with potential minor semantic changes).[74]

Python is meant to be a fun language to use. This goal is reflected in the name – a tribute to the British comedy group Monty Python[75] – and in playful approaches to some tutorials and reference materials. For instance, some code examples use the terms "spam" and "eggs" (in reference to a Monty Python sketch), rather than the typical terms "foo" and "bar".[76][77]

A common neologism in the Python community is *pythonic*, which has a wide range of meanings related to program style: Pythonic code may use Python idioms well; be natural or show fluency in the language; or conform with Python's minimalist philosophy and emphasis on readability.[78]

Syntax and semantics

Python is meant to be an easily readable language. Its formatting is visually uncluttered and 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 allowed but rarely used. It has fewer syntactic exceptions and special cases than C or Pascal.[79]

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.[80] Thus, the program's visual structure accurately represents its semantic structure.[81] This feature is sometimes termed the off-side rule. Some other languages use indentation this way; but in most, indentation has no semantic meaning. The recommended indent size is four spaces.[82]

Statements and control flow

Python's statements include the following:

- The assignment statement, using a single equals sign =
- 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 boolean condition is true
- The `try` statement, which allows exceptions raised in its attached code block to be caught and handled by `except` clauses (or new syntax `except*` in Python 3.11 for exception groups[83]); the `try` statement also ensures that clean-up code in a `finally` block is always 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, which encloses a code block within a context manager, allowing resource-acquisition-is-initialization (RAII)-like behavior and replacing a common try/finally idiom[84] Examples of a context include acquiring a lock before some code is run, and then releasing the lock; or opening and then closing a file
- The `break` statement, which exits a loop
- The `continue` statement, which skips the rest of the current iteration and continues with the next

- The `del` statement, which removes a variable—deleting the reference from the name to the value, and producing an error if the variable is referred to before it is redefined [c]
- The `pass` statement, serving as a NOP (i.e., no operation), which is syntactically needed to create an empty code block
- The `assert` statement, used in debugging to check for conditions that should apply
- The `yield` statement, which returns a value from a generator function (and also an operator); used to implement coroutines
- The `return` statement, used to return a value from a function
- The `import` and `from` statements, used to import modules whose functions or variables can be used in the current program
- The `match` and `case` statements, analogous to a switch statement construct, which compares an expression against one or more cases as a control-flow measure

The assignment statement (`=`) binds a name as a reference to a separate, dynamically allocated object. Variables may subsequently be rebound at any time to any object. In Python, a variable name is a generic reference holder without a fixed data type; however, it always refers to *some* object with a type. This is called dynamic typing—in contrast to statically-typed languages, where each variable may contain only a value of a certain type.

Python does not support tail call optimization or first-class continuations; according to Van Rossum, the language never will.[85][86] However, better support for coroutine-like functionality is provided by extending Python's generators.[87] Before 2.5, generators were lazy iterators; data was passed unidirectionally out of the generator. From Python 2.5 on, it is possible to pass data back into a generator function; and from version 3.3, data can be passed through multiple stack levels.[88]

Expressions

Python's expressions include the following:

- The `+`, `-`, and `*` operators for mathematical addition, subtraction, and multiplication are similar to other languages, but the behavior of division differs. There are two types of division in Python: floor division (or integer division) `//`, and floating-point division `/`.[89] Python uses the `**` operator for exponentiation.
- Python uses the `+` operator for string concatenation. The language uses the `*` operator for duplicating a string a specified number of times.
- The `@` infix operator is intended to be used by libraries such as NumPy for matrix multiplication.[90][91]
- The syntax `:=`, called the "walrus operator", was introduced in Python 3.8. This operator assigns values to variables as part of a larger expression.[92]
- In Python, `==` compares two objects by value. Python's `is` operator may be used to compare object identities (i.e., comparison by reference), and comparisons may be chained—for example, `a <= b <= c`.
- Python uses `and`, `or`, and `not` as Boolean operators.
- Python has a type of expression called a *list comprehension*, and a more general expression called a *generator expression*.[64]

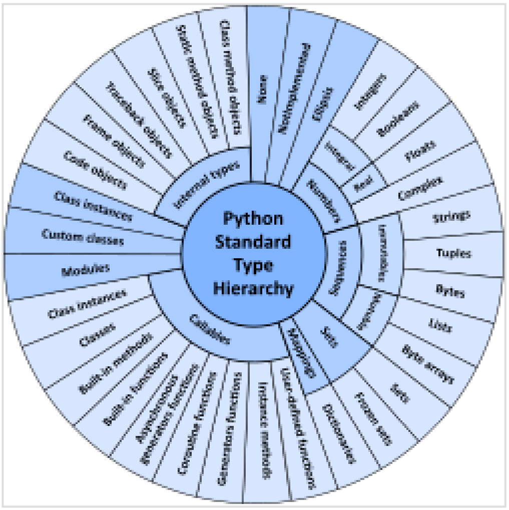
- Anonymous functions are implemented using lambda expressions; however, there may be only one expression in each body.
- Conditional expressions are written as `x if c else y`.[93] (This is different in operand order from the `c ? x : y` operator common to many other languages.)
- Python makes a distinction between lists and tuples. Lists are written as `[1, 2, 3]`, are mutable, and cannot be used as the keys of dictionaries (since dictionary keys must be immutable in Python). Tuples, written as `(1, 2, 3)`, are immutable and thus can be used as the keys of dictionaries, provided that all of the tuple's elements are immutable. The + operator can be used to concatenate two tuples, which does not directly modify their contents, but produces a new tuple containing the elements of both. For example, 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)`; this result 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.[94]
- Python features *sequence unpacking* where multiple expressions, each evaluating to something assignable (e.g., a variable or a writable property) are associated just as in forming tuple literal; as a whole, the results are then put on the left-hand side of the equal sign in an assignment statement. This statement expects an *iterable* object on the right-hand side of the equal sign to produce the same number of values as the writable expressions on the left-hand side; while iterating, the statement assigns each of the values produced on the right to the corresponding expression on the left.[95]
- Python has a "string format" operator % that functions analogously to `printf` format strings in the C language—e.g. `"spam=%s eggs=%d" % ("blah", 2)` evaluates to `"spam=blah eggs=2"`. In Python 2.6+ and 3+, this operator 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": `spam = "blah"; eggs = 2; f'spam={spam} eggs={eggs}'`.[96]
- Strings in Python can be concatenated by "adding" them (using the same operator as for adding integers and floats); e.g., `"spam" + "eggs"` returns `"spameggs"`. If strings contain numbers, they are concatenated as strings rather than as integers, e.g. `"2" + "2"` returns `"22"`.
- Python supports string literals in several ways:
  - Delimited by single or double quotation marks; single and double quotation marks have equivalent functionality (unlike in Unix shells, Perl, and Perl-influenced languages). Both marks use the backslash (`\`) as an escape character. String interpolation became available in Python 3.6 as "formatted string literals".[96]
  - Triple-quoted, i.e., starting and ending with three single or double quotation marks; this may span multiple lines and function like here documents in shells, Perl, and Ruby.
  - Raw string varieties, denoted by prefixing the string literal with `r`. Escape sequences are not interpreted; hence raw strings are useful where literal backslashes are common, such as in regular expressions and Windows-style paths. (Compare "@-quoting" in C#.)
- Python has array index and array slicing expressions in lists, which are written 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 (optional) third slice parameter, called *step* or *stride*, allows elements to be skipped or 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 distinction 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 function is for expressions, while the latter is for statements

A statement cannot be part of an expression; because of this restriction, expressions such as list and `dict` comprehensions (and lambda expressions) cannot contain statements. As a particular case, an assignment statement such as `a = 1` cannot be part of the conditional expression of a conditional statement.

Typing



The standard type hierarchy in Python 3

Python uses duck typing, and it has typed objects but untyped variable names. Type constraints are not checked at definition time; rather, operations on an object may fail at usage time, indicating that the object is not of an appropriate type. Despite being dynamically typed, Python is strongly typed, forbidding operations that are poorly defined (e.g., adding a number and a string) rather than quietly attempting to interpret them.

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

Before version 3.0, Python had two kinds of classes, both using the same syntax: *old-style* and *new-style*.[97] Current Python versions support the semantics of only the new style.

Python supports optional type annotations.[5][98] These annotations are not enforced by the language, but may be used by external tools such as **mypy** to catch errors. Python includes a module `typing` including several type names for type annotations.[99][100] Mypy also supports a Python compiler called mypyc, which leverages type annotations for optimization.[101]

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`
			`3 + 2.7j`
			`5j`

`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}`
			`{}`

`types.EllipsisType`	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 it is generally implemented as a 64-bit IEEE 754 number with 53 bits of precision.[102]	`1.33333`

`frozenset`	immutable	Unordered set, contains no duplicates; can contain mixed types, if hashable	`frozenset({4.0, 'string', True})`
			`frozenset()`

`int`	immutable	Integer of unlimited magnitude[103]	`42`

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

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

`range`	immutable	An *immutable sequence* of numbers, commonly used for	`range(-1, 10)`
			`range(10, -5, -2)`

		iterating a specific number of times in `for` loops[104]	

`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	Tuple, can contain mixed types	`(4.0, 'string', True)`
			`('single element',)`
			`()`

Arithmetic operations

Python includes conventional symbols for arithmetic operators (`+`, `-`, `*`, `/`), the floor-division operator `//`, and the modulo operator `%`. (With the modulo operator, a remainder can be negative, e.g., `4 % -3 == -2`.) Python also offers the `**` symbol for exponentiation, e.g. `5**3 == 125` and `9**0.5 == 3.0`; it also offers the matrix-multiplication operator `@` .[105] These operators work as in traditional mathematics; with the same precedence rules, the infix operators `+` and `-` can also be unary, to represent positive and negative numbers respectively.

Division between integers produces floating-point results. The behavior of division has changed significantly over time:[106]

- The current version of Python (i.e., since 3.0) changed the `/` operator to always represent floating-point division, e.g., `5/2 == 2.5`.
- The floor division `//` operator was introduced, meaning that `7//3 == 2`, `-7//3 == -3`, `7.5//3 == 2.0`, and `-7.5//3 == -3.0`. For Python 2.7, adding the `from __future__ import division` statement allows a module in Python 2.7 to use Python 3.x rules for division (see above).

In Python terms, the `/` operator represents *true division* (or simply *division*), while the `//` operator represents *floor division*. Before version 3.0, the `/` operator represents *classic division*.[106]

Rounding towards negative infinity, though a different method than in most languages, adds consistency to Python. For instance, this rounding implies that the equation `(a + b)//b == a//b + 1` is always true. The rounding also implies that the equation `b*(a//b) + a%b == a` is valid for both positive and negative values of `a`. As expected, the result of `a%b` lies in the half-open interval [0, *b*), where `b` is a positive integer; however, maintaining the validity of the equation requires that the result must lie in the interval (*b*, 0] when `b` is negative.[107]

Python provides a `round` function for rounding a float to the nearest integer. For tie-breaking, Python 3 uses the *round to even* method: `round(1.5)` and `round(2.5)` both produce `2`.[108] Python versions before 3 used the round-away-from-zero method: `round(0.5)` is `1.0`, and `round(-0.5)` is `-1.0`.[109]

Python allows Boolean expressions that contain multiple equality relations to be consistent with general usage in mathematics. For example, the expression `a < b < c` tests whether `a` is less than `b` and `b` is less than `c`.[110] 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`.[111]

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 with several rounding modes.[112] The `Fraction` class in the `fractions` module provides arbitrary precision for rational numbers.[113]

Due to Python's extensive mathematics library and the third-party library NumPy, the language is frequently used for scientific scripting in tasks such as numerical data processing and manipulation.[114][115]

Function syntax

Functions are created in Python by using the `def` keyword. A function is defined similarly to how it is called, by first providing the function name and then the required parameters. Here is an example of a function that prints its inputs:

```python
def printer(input1, input2 = "already there"):
    print(input1)
    print(input2)

printer("hello")

# Example output:
# hello
# already there
```

To assign a default value to a function parameter in case no actual value is provided at run time, variable-definition syntax can be used inside the function header.

Code examples

"Hello, World!" program:

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

Program to calculate the factorial of a non-negative integer:

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

if n < 0:
    raise ValueError('You must enter a non-negative integer')

factorial = 1
for i in range(2, n + 1):
    factorial *= i

print(factorial)
```

Libraries

Python's large standard library[116] is commonly cited as one of its greatest strengths. For Internet-facing applications, many standard formats and protocols such as MIME and HTTP are supported. The language includes modules for creating graphical user interfaces, connecting to relational databases, generating pseudorandom numbers, arithmetic with arbitrary-precision decimals,[112] 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[117]—but most parts 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 must be altered or rewritten for variant implementations.

As of 13 March 2025, the Python Package Index (PyPI), the official repository for third-party Python software, contains over 614,339[118] packages.

Development environments

Most Python implementations (including CPython) include a read–eval–print loop (REPL); this permits the environment to function as a command line interpreter, with which users enter statements sequentially and receive results immediately.[119]

Python is also bundled with an integrated development environment (IDE) called IDLE,[120] which is oriented toward beginners.

Other shells, including IDLE and IPython, add additional capabilities such as improved auto-completion, session-state retention, and syntax highlighting.[120][121]

Standard desktop IDEs include PyCharm, Spyder, and Visual Studio Code; there are also web browser-based IDEs, such as the following environments:

- Jupyter Notebooks, an open-source interactive computing platform;[122]
- PythonAnywhere, a browser-based IDE and hosting environment; and
- Canopy, a commercial IDE from Enthought that emphasizes scientific computing.[123][124]

Implementations

Reference implementation

CPython is the reference implementation of Python. This implementation is written in C, meeting the C11 standard[125] since version 3.11. Older versions use the C89 standard with several select C99 features, but third-party extensions are not limited to older C versions—e.g., they can be implemented using C11 or C++.[126][127] CPython compiles Python programs into an intermediate bytecode,[128] which is then executed by a virtual machine.[129] CPython is distributed with a large standard library written in a combination of C and native Python.

CPython is available for many platforms, including Windows and most modern Unix-like systems, including macOS (and Apple M1 Macs, since Python 3.9.1, using an experimental installer). Starting with Python 3.9, the Python installer intentionally fails to install on Windows 7 and 8;[130][131] Windows XP was supported until Python 3.5, with unofficial support for VMS.[132] Platform portability was one of Python's earliest priorities.[133] During development of Python 1 and 2, even OS/2 and Solaris were supported;[8] since that time, support has been dropped for many platforms.

All current Python versions (since 3.7) support only operating systems that feature multithreading, by now supporting not nearly as many operating systems (dropping many outdated) than in the past.

Other implementations

All alternative implementations have at least slightly different semantics. For example, an alternative may include unordered dictionaries, in contrast to other current Python versions. As another example in the larger Python ecosystem, PyPy does not support the full C Python API. Alternative implementations include the following:

- PyPy is a fast, compliant interpreter of Python 2.7 and 3.10.[134][135] PyPy's just-in-time compiler often improves speed significantly relative to CPython, but PyPy does not support some libraries written in C.[136] PyPy offers support for the RISC-V instruction-set architecture.
- Codon is an implementation with an ahead-of-time (AOT) compiler, which compiles a statically-typed Python-like language whose "syntax and semantics are nearly identical to Python's, there are some notable differences"[137] For example, Codon uses 64-bit machine integers for speed, not arbitrarily as with Python; Codon developers claim that speedups over CPython are usually on the order of ten to a hundred times. Codon compiles to machine code (via LLVM) and supports native multithreading.[138] Codon can also compile to Python extension modules that can be imported and used from Python.
- MicroPython and CircuitPython are Python 3 variants that are optimized for microcontrollers, including the Lego Mindstorms EV3.[139]
- Pyston is a variant of the Python runtime that uses just-in-time compilation to speed up execution of Python programs.[140]
- Cinder is a performance-oriented fork of CPython 3.8 that features a number of optimizations, including bytecode inline caching, eager evaluation of coroutines, a method-at-a-time JIT, and an experimental bytecode compiler.[141]
- The Snek[142][143][144] embedded computing language "is Python-inspired, but it is not Python. It is possible to write Snek programs that run under a full Python system, but most Python programs will not run under Snek."[145] Snek is compatible with 8-bit AVR microcontrollers such as ATmega 328P-based Arduino, as well as larger microcontrollers that are compatible with MicroPython. Snek is an imperative language that (unlike Python) omits object-oriented programming. Snek supports only one numeric data type, which features 32-bit single precision (resembling JavaScript numbers, though smaller).

Unsupported implementations

Stackless Python is a significant fork of CPython that implements microthreads. This implementation uses the call stack differently, thus allowing massively concurrent programs. PyPy also offers a stackless version.[146]

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

- Google began a project named Unladen Swallow in 2009: this project aimed to speed up the Python interpreter five-fold by using LLVM, and improve multithreading capability for scaling to thousands of cores,[147] while typical implementations are limited by the global interpreter lock.
- Psyco is a discontinued just-in-time specializing compiler, which 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. Psyco does not support Python 2.7 or later.
- PyS60 was a Python 2 interpreter for Series 60 mobile phones, which was released by Nokia in 2005. The interpreter implemented many modules from Python's standard library, as well as additional modules for integration with the Symbian operating system. The Nokia N900 also supports Python through the GTK widget library, allowing programs to be written and run on the target device.[148]

Cross-compilers to other languages

There are several compilers/transpilers to high-level object languages; the source language is unrestricted Python, a subset of Python, or a language similar to Python:

- Brython[149] and Transcrypt[150][151] compile Python to JavaScript.
- Cython compiles a superset of Python to C. The resulting code can be used with Python via direct C-level API calls into the Python interpreter.
- PyJL compiles/transpiles a subset of Python to "human-readable, maintainable, and high-performance Julia source code".[74] Despite the developers' performance claims, this is not possible for *arbitrary* Python code; that is, compiling to a faster language or machine code is known to be impossible in the general case. The semantics of Python might potentially be changed, but in many cases speedup is possible with few or no changes in the Python code. The faster Julia source code can then be used from Python or compiled to machine code.
- Nuitka compiles Python into C.[152] This compiler works with Python 3.4 to 3.13 (and 2.6 and 2.7) for Python's main supported platforms (and Windows 7 or even Windows XP) and for Android. The compiler developers claim full support for Python 3.10, partial support for Python 3.11 and 3.12, and experimental support for Python 3.13. Nuitka supports macOS including Apple Silicon-based versions. The compiler is free of cost, though it has commercial add-ons (e.g., for hiding source code).
- Numba is a JIT compiler that is used from Python; the compiler translates a subset of Python and NumPy code into fast machine code. This tool is enabled by adding a decorator to the relevant Python code.
- Pythran compiles a subset of Python 3 to C++ (C++11).[153]
- RPython can be compiled to C, and it is used to build the PyPy interpreter for Python.
- The Python → 11l → C++ transpiler[154] compiles a subset of Python 3 to C++ (C++17).

There are also specialized compilers:

- MyHDL is a Python-based hardware description language (HDL) that converts MyHDL code to Verilog or VHDL code.

Some older projects existed, as well as compilers not designed for use with Python 3.x and related syntax:

- Google's Grumpy transpiles Python 2 to Go.[155][156][157] The latest release was in 2017.

- IronPython allows running Python 2.7 programs with the .NET Common Language Runtime.[158] An alpha version (released in 2021), is available for "Python 3.4, although features and behaviors from later versions may be included."[159]
- Jython compiles Python 2.7 to Java bytecode, allowing the use of Java libraries from a Python program.[160]
- Pyrex (last released in 2010) and Shed Skin (last released in 2013) compile to C and C++ respectively.

Performance

A performance comparison among various Python implementations, using a non-numerical (combinatorial) workload, was presented at EuroSciPy '13.[161] In addition, Python's performance relative to other programming languages is benchmarked by The Computer Language Benchmarks Game.[162]

There are several approaches to optimizing Python performance, given the inherent slowness of an interpreted language. These approaches include the following strategies or tools:

- Just-in-time compilation: Dynamically compiling Python code just before it is executed. This technique is used in libraries such as Numba and PyPy.
- Static compilation: Python code is compiled into machine code sometime before execution. An example of this approach is Cython, which compiles Python into C.
- Concurrency and parallelism: Multiple tasks can be run simultaneously. Python contains modules such as `multiprocessing` to support this form of parallelism. Moreover, this approach helps to overcome limitations of the Global Interpreter Lock (GIL) in CPU tasks.
- Efficient data structures: Performance can also be improved by using data types such as `Set` for membership tests, or `deque` from `collections` for queue operations.
- Performance gains can be observed by utilizing vectorized operations from libraries such as NumPy, which uses highly optimized functions written in C or Fortran for computationally intensive tasks rather than the Python interpreter.[163]

Language Development

Python's development is conducted largely through the *Python Enhancement Proposal* (PEP) process; this process is the primary mechanism for proposing major new features, collecting community input on issues, and documenting Python design decisions.[164] Python coding style is covered in PEP 8.[82] Outstanding PEPs are reviewed and commented on by the Python community and the steering council.[164]

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 were originally discussed in the Roundup bug tracker hosted by the foundation.[165] In 2022, all issues and discussions were migrated to GitHub.[166] Development originally took place on a self-hosted source-code repository running Mercurial, until Python moved to GitHub in January 2017.[167]

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

- *Backward-incompatible versions*, where code is expected to break and must be manually ported. The first part of the version number is incremented. These releases happen

infrequently—version 3.0 was released 8 years after 2.0. According to Guido van Rossum, a version 4.0 will probably never exist.[168]
- *Major or "feature" releases* are largely compatible with the previous version but introduce new features. The second part of the version number is incremented. Starting with Python 3.9, these releases are expected to occur annually.[169][170] Each major version is supported by bug fixes for several years after its release.[171]
- *Bug fix releases*,[172] which introduce no new features, occur approximately every three months; these releases 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.[172]

Many alpha, beta, and release-candidates are also released as previews and for testing before final releases. Although there is a rough schedule for releases, they are often delayed if the code is not ready yet. Python's development team monitors the state of the code by running a large unit test suite during development.[173]

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

Naming

Python's name is inspired by 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;[174] for example, the metasyntactic variables often used in Python literature are *spam* and *eggs*, rather than the traditional *foo* and *bar*.[174][175] The official Python documentation also contains various references to Monty Python routines.[176][177] Python users are sometimes referred to as "Pythonistas".[178]

The affix *Py* is often used when naming Python applications or libraries. Some examples include the following:

- Pygame, a binding of Simple DirectMedia Layer to Python (commonly used to create games);
- PyQt and PyGTK, which bind Qt and GTK to Python respectively;
- PyPy, a Python implementation originally written in Python;
- NumPy, a Python library for numerical processing.
- Jupyter, a notebook interface and associated project for interactive computing

See also

Computer programming portal

Free and open-source software portal

- Google Colab – zero setup online IDE that runs Python
- List of Python programming books
- pip (package manager)

External links

- Python documentation (https://docs.python.org/3/)

Notes

a. since 3.5, but those hints are ignored, except with unofficial tools[5]

b.
- **Tier 1**: 64-bit Linux, macOS; 64- and 32-bit Windows 10+[6]
- **Tier 2**: E.g. 32-bit WebAssembly (WASI)
- **Tier 3**: 64-bit Android,[7] iOS, FreeBSD, and (32-bit) Raspberry Pi OS
  Unofficial (or has been known to work): Other Unix-like/BSD variants) and a few other platforms[8][9][10]

c. `del` in Python does not behave the same way `delete` in languages such as C++ does, where such a word is used to call the destructor and deallocate heap memory.

References

1. "General Python FAQ – Python 3 documentation" (https://docs.python.org/3/faq/general.html#what-is-python). *docs.python.org*. Retrieved 7 July 2024.
2. "Python 0.9.1 part 01/21" (https://www.tuhs.org/Usenet/alt.sources/1991-February/001749.html). alt.sources archives. Archived (https://web.archive.org/web/20210811171015/https://www.tuhs.org/Usenet/alt.sources/1991-February/001749.html) from the original on 11 August 2021. Retrieved 11 August 2021.
3. Hugo van Kemenade (7 October 2025). "Python Insider: Python 3.14.0 (final) is here!" (https://pythoninsider.blogspot.com/2025/10/python-3140-final-is-here.html). Retrieved 7 October 2025.
4. "Why is Python a dynamic language and also a strongly typed language" (https://wiki.python.org/moin/Why%20is%20Python%20a%20dynamic%20language%20and%20also%20a%20strongly%20typed%20language). *Python Wiki*. Archived (https://web.archive.org/web/20210314173706/https://wiki.python.org/moin/Why%20is%20Python%20a%20dynamic%20language%20and%20also%20a%20strongly%20typed%20language) from the original on 14 March 2021. Retrieved 27 January 2021.
5. van Rossum, Guido; Levkivskyi, Ivan. "PEP 483 – The Theory of Type Hints" (https://www.python.org/dev/peps/pep-0483/). *Python Enhancement Proposals (PEPs)*. Archived (https://web.archive.org/web/20200614153558/https://www.python.org/dev/peps/pep-0483/) from the original on 14 June 2020. Retrieved 14 June 2018.
6. von Löwis, Martin; Cannon, Brett. "PEP 11 – CPython platform support" (https://peps.python.org/pep-0011/). *Python Enhancement Proposals (PEPs)*. Retrieved 22 April 2024.
7. "PEP 738 – Adding Android as a supported platform | peps.python.org" (https://peps.python.org/pep-0738/). *Python Enhancement Proposals (PEPs)*. Retrieved 19 May 2024.
8. "Download Python for Other Platforms" (https://www.python.org/download/other/). *Python.org*. Archived (https://web.archive.org/web/20201127015815/https://www.python.org/download/other/) from the original on 27 November 2020. Retrieved 18 August 2023.

9. "test – Regression tests package for Python" (https://docs.python.org/3.7/library/test.html?highlight=android#test.support.is_android). *Python 3.7.17 documentation*. Archived (https://web.archive.org/web/20220517151240/https://docs.python.org/3.7/library/test.html?highlight=android#test.support.is_android) from the original on 17 May 2022. Retrieved 17 May 2022.
10. "platform – Access to underlying platform's identifying data" (https://docs.python.org/3.10/library/platform.html?highlight=android). *Python 3.10.4 documentation*. Archived (https://web.archive.org/web/20220517150826/https://docs.python.org/3/library/platform.html?highlight=android) from the original on 17 May 2022. Retrieved 17 May 2022.
11. Holth, Daniel; Moore, Paul (30 March 2014). "PEP 0441 – Improving Python ZIP Application Support" (https://www.python.org/dev/peps/pep-0441/). *Python Enhancement Proposals (PEPs)*. Archived (https://web.archive.org/web/20181226141117/https://www.python.org/dev/peps/pep-0441/%20) from the original on 26 December 2018. Retrieved 12 November 2015.
12. "Starlark Language" (https://docs.bazel.build/versions/master/skylark/language.html). *bazel.build*. Archived (https://web.archive.org/web/20200615140534/https://docs.bazel.build/versions/master/skylark/language.html) from the original on 15 June 2020. Retrieved 25 May 2019.
13. "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. Archived (https://web.archive.org/web/20121024164224/http://docs.python.org/faq/general.html#why-was-python-created-in-the-first-place) from the original on 24 October 2012. Retrieved 22 March 2007. "I had extensive experience with implementing an interpreted language in the ABC group at CWI, and from working with this group I had learned a lot about language design. This is the origin of many Python features, including the use of indentation for statement grouping and the inclusion of very high-level data types (although the details are all different in Python)."
14. "Ada 83 Reference Manual (raise statement)" (https://archive.adaic.com/standards/83lrm/html/lrm-11-03.html#11.3). *archive.adaic.com*. Archived (https://web.archive.org/web/20191022155758/http://archive.adaic.com/standards/83lrm/html/lrm-11-03.html#11.3) from the original on 22 October 2019. Retrieved 7 January 2020.
15. 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. "I'd spent a summer at DEC's Systems Research Center, which introduced me to Modula-2+; the Modula-3 final report was being written there at about the same time. What I learned there later showed up in Python's exception handling, modules, and the fact that methods explicitly contain 'self' in their parameter list. String slicing came from Algol-68 and Icon."
16. "itertools – Functions creating iterators for efficient looping" (https://docs.python.org/3.7/library/itertools.html). *Python 3.7.17 documentation*. Archived (https://web.archive.org/web/20200614153629/https://docs.python.org/3/library/itertools.html) from the original on 14 June 2020. Retrieved 22 November 2016. "This module implements a number of iterator building blocks inspired by constructs from APL, Haskell, and SML."
17. 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."
18. "Classes" (https://docs.python.org/tutorial/classes.html). *The Python Tutorial*. Python Software Foundation. Archived (https://web.archive.org/web/20121023030209/http://docs.python.org/tutorial/classes.html) from the original on 23 October 2012. Retrieved 20 February 2012. "It is a mixture of the class mechanisms found in C++ and Modula-3"

19. Lundh, Fredrik. "Call By Object" (http://effbot.org/zone/call-by-object.htm). *effbot.org*. Archived (https://web.archive.org/web/20191123043655/http://effbot.org/zone/call-by-object.htm) from the original on 23 November 2019. 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."
20. Simionato, Michele. "The Python 2.3 Method Resolution Order" (https://www.python.org/download/releases/2.3/mro/). Python Software Foundation. Archived (https://web.archive.org/web/20200820231854/https://www.python.org/download/releases/2.3/mro/) from the original on 20 August 2020. Retrieved 29 July 2014. "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"
21. Kuchling, A. M. "Functional Programming HOWTO" (https://docs.python.org/howto/functional.html). *Python v2.7.2 documentation*. Python Software Foundation. Archived (https://web.archive.org/web/20121024163217/http://docs.python.org/howto/functional.html) from the original on 24 October 2012. Retrieved 9 February 2012. "List comprehensions and generator expressions [...] are a concise notation for such operations, borrowed from the functional programming language Haskell."
22. 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. Archived (https://web.archive.org/web/20200605012926/https://www.python.org/dev/peps/pep-0255/) from the original on 5 June 2020. Retrieved 9 February 2012.
23. "More Control Flow Tools" (https://docs.python.org/3.2/tutorial/controlflow.html). *Python 3 documentation*. Python Software Foundation. Archived (https://web.archive.org/web/20160604080843/https://docs.python.org/3.2/tutorial/controlflow.html) from the original on 4 June 2016. Retrieved 24 July 2015. "By popular demand, a few features commonly found in functional programming languages like Lisp have been added to Python. With the lambda keyword, small anonymous functions can be created."
24. "re – Regular expression operations" (https://docs.python.org/3.10/library/re.html). *Python 3.10.6 documentation*. Archived (https://web.archive.org/web/20180718132241/https://docs.python.org/3/library/re.html) from the original on 18 July 2018. Retrieved 6 September 2022. "This module provides regular expression matching operations similar to those found in Perl."
25. "CoffeeScript" (https://coffeescript.org/). *coffeescript.org*. Archived (https://web.archive.org/web/20200612100004/http://coffeescript.org/) from the original on 12 June 2020. Retrieved 3 July 2018.
26. Rauschmayer, Axel (24 February 2013). "Perl and Python influences in JavaScript" (https://www.2ality.com/2013/02/javascript-influences.html). *2ality.com*. Archived (https://web.archive.org/web/20181226141121/http://2ality.com/2013/02/javascript-influences.html%0A) from the original on 26 December 2018. Retrieved 15 May 2015.
27. Rauschmayer, Axel. "Chapter 3: The Nature of JavaScript; Influences" (https://speakingjs.com/es5/ch03.html). *Speaking JavaScript*. O'Reilly. Archived (https://web.archive.org/web/20181226141123/http://speakingjs.com/es5/ch03.html%0A) from the original on 26 December 2018. Retrieved 15 May 2015.
28. Bezanson, Jeff; Karpinski, Stefan; Shah, Viral B.; Edelman, Alan (February 2012). "Why We Created Julia" (https://julialang.org/blog/2012/02/why-we-created-julia). *Julia website*. Archived (https://web.archive.org/web/20200502144010/https://julialang.org/blog/2012/02/why-we-created-julia/) from the original on 2 May 2020. Retrieved 5 June 2014. "We want something as usable for general programming as Python [...]"

29. Krill, Paul (4 May 2023). "Mojo language marries Python and MLIR for AI development" (https://www.infoworld.com/article/3695588/mojo-language-marries-python-and-mlir-for-ai-development.html). *InfoWorld*. Archived (https://web.archive.org/web/20230505064554/https://www.infoworld.com/article/3695588/mojo-language-marries-python-and-mlir-for-ai-development.html) from the original on 5 May 2023. Retrieved 5 May 2023.
30. Ring Team (4 December 2017). "Ring and other languages" (https://web.archive.org/web/20181225175312/http://ring-lang.sourceforge.net/doc1.6/introduction.html#ring-and-other-languages). *ring-lang.net*. ring-lang. Archived from the original (https://ring-lang.sourceforge.net/doc1.6/introduction.html#ring-and-other-languages) on 25 December 2018. Retrieved 4 December 2017.
31. 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.
32. Lattner, Chris (3 June 2014). "Chris Lattner's Homepage" (http://nondot.org/sabre/). Chris Lattner. Archived (https://web.archive.org/web/20181225175312/http://nondot.org/sabre/) from the original on 25 December 2018. 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."
33. "V documentation (Introduction)" (https://github.com/vlang/v/blob/master/doc/docs.md#introduction). *GitHub*. Retrieved 24 December 2024.
34. 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.
35. "PEP 484 – Type Hints" (https://peps.python.org/pep-0484/). *Python Enhancement Proposals*. Retrieved 27 October 2025.
36. "mypy – Optional Static Typing for Python" (https://mypy-lang.org/). *mypy-lang.org*. Retrieved 17 August 2025.
37. "Stack Overflow Developer Survey 2022" (https://survey.stackoverflow.co/2022/). *Stack Overflow*. Archived (https://web.archive.org/web/20220627175307/https://survey.stackoverflow.co/2022/) from the original on 27 June 2022. Retrieved 12 August 2022.
38. "The State of Developer Ecosystem in 2020 Infographic" (https://www.jetbrains.com/lp/devecosystem-2020/). *JetBrains*. Archived (https://web.archive.org/web/20210301062411/https://www.jetbrains.com/lp/devecosystem-2020/) from the original on 1 March 2021. Retrieved 5 March 2021.
39. "TIOBE Index" (https://www.tiobe.com/tiobe-index/). TIOBE. Archived (https://web.archive.org/web/20180225101948/https://www.tiobe.com/tiobe-index/) from the original on 25 February 2018. Retrieved 3 January 2023. "The TIOBE Programming Community index is an indicator of the popularity of programming languages" Updated as required.
40. Healy, John; McInnes, Leland; Weir, Colin (2017). "Bridging the Cyber-Analysis Gap: The Democratization of Data Science". *The Cyber Defense Review*. **2** (1): 109–118. ISSN 2474-2120 (https://search.worldcat.org/issn/2474-2120). JSTOR 26267404 (https://www.jstor.org/stable/26267404). "Python is the lingua franca of data science and machine learning."
41. Sultana, Simon G.; Reed, Philip A. (2017). "Curriculum for an Introductory Computer Science Course: Identifying Recommendations from Academia and Industry". *The Journal of Technology Studies*. **43** (2): 80–92. doi:10.21061/jots.v43i2.a.3 (https://doi.org/10.21061%2Fjots.v43i2.a.3). ISSN 1071-6084 (https://search.worldcat.org/issn/1071-6084). JSTOR 90023144 (https://www.jstor.org/stable/90023144).

42. Venners, Bill (13 January 2003). "The Making of Python" (http://www.artima.com/intv/pythonP.html). *Artima Developer*. Artima. Archived (https://web.archive.org/web/20160901183332/http://www.artima.com/intv/pythonP.html) from the original on 1 September 2016. Retrieved 22 March 2007.
43. 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*. Archived (https://web.archive.org/web/20200605032200/https://python-history.blogspot.com/2009/01/brief-timeline-of-python.html) from the original on 5 June 2020. Retrieved 20 January 2009.
44. 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). Archived (https://web.archive.org/web/20180714064019/https://mail.python.org/pipermail/python-dev/2000-August/008881.html) from the original on 14 July 2018. Retrieved 13 March 2011.
45. 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*. Archived (https://web.archive.org/web/20180713192427/https://www.linuxjournal.com/content/guido-van-rossum-stepping-down-role-pythons-benevolent-dictator-life) from the original on 13 July 2018. Retrieved 13 July 2018.
46. Smith, Nathaniel J.; Durbin, Ee. "PEP 8100 – January 2019 Steering Council election" (https://peps.python.org/pep-8100/). *Python Enhancement Proposals (PEPs)*. Python Software Foundation. Archived (https://web.archive.org/web/20200604235027/https://www.python.org/dev/peps/pep-8100/) from the original on 4 June 2020. Retrieved 4 May 2019.
47. The Python core team and community. "PEP 13 – Python Language Governance" (https://peps.python.org/pep-0013/). *Python Enhancement Proposals (PEPs)*. Archived (https://web.archive.org/web/20210527000035/https://www.python.org/dev/peps/pep-0013/) from the original on 27 May 2021. Retrieved 25 August 2021.
48. Briggs, Jason R.; Lipovača, Miran (2013). *Python for kids: a playful introduction to programming* (https://archive.org/details/pythonforkidspla0000brig). San Francisco, California, USA: No Starch Press. ISBN 978-1-59327-407-8. LCCN 2012044047 (https://lccn.loc.gov/2012044047). OCLC 825076499 (https://search.worldcat.org/oclc/825076499). OL 26119645M (https://openlibrary.org/books/OL26119645M).
49. 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. Archived (https://web.archive.org/web/20121023112045/http://docs.python.org/whatsnew/2.0.html) from the original on 23 October 2012. Retrieved 11 February 2012.
50. Peterson, Benjamin. "PEP 373 – Python 2.7 Release Schedule" (https://legacy.python.org/dev/peps/pep-0373/). *python.org*. Archived (https://web.archive.org/web/20200519075520/https://legacy.python.org/dev/peps/pep-0373/) from the original on 19 May 2020. Retrieved 9 January 2017.
51. Coghlan, Alyssa. "PEP 466 – Network Security Enhancements for Python 2.7.x" (https://peps.python.org/pep-0466/). *Python Enhancement Proposals (PEPs)*. Archived (https://web.archive.org/web/20200604232833/https://www.python.org/dev/peps/pep-0466/) from the original on 4 June 2020. Retrieved 9 January 2017.
52. "Sunsetting Python 2" (https://www.python.org/doc/sunset-python-2/). *Python.org*. Archived (https://web.archive.org/web/20200112080903/https://www.python.org/doc/sunset-python-2/) from the original on 12 January 2020. Retrieved 22 September 2019.
53. Peterson, Benjamin. "PEP 373 – Python 2.7 Release Schedule" (https://peps.python.org/pep-0373/). *Python Enhancement Proposals (PEPs)*. Archived (https://web.archive.org/web/20200113033257/https://www.python.org/dev/peps/pep-0373/) from the original on 13 January 2020. Retrieved 22 September 2019.

54. mattip (25 December 2023). "PyPy v7.3.14 release" (https://www.pypy.org/posts/2023/12/pypy-v7314-release.html). *PyPy*. Archived (https://web.archive.org/web/20240105132820/https://www.pypy.org/posts/2023/12/pypy-v7314-release.html) from the original on 5 January 2024. Retrieved 5 January 2024.
55. Peterson, Benjamin (20 April 2020). "Python 2.7.18, the last release of Python 2" (https://pythoninsider.blogspot.com/2020/04/python-2718-last-release-of-python-2.html). *Python Insider*. Archived (https://web.archive.org/web/20200426204118/https://pythoninsider.blogspot.com/2020/04/python-2718-last-release-of-python-2.html) from the original on 26 April 2020. Retrieved 27 April 2020.
56. "Status of Python versions" (https://devguide.python.org/versions/). *Python Developer's Guide*. Retrieved 8 October 2025.
57. 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.
58. "3.3. Special method names" (https://docs.python.org/3.0/reference/datamodel.html#special-method-names). *The Python Language Reference*. Python Software Foundation. Archived (https://web.archive.org/web/20181215123146/https://docs.python.org/3.0/reference/datamodel.html#special-method-names) from the original on 15 December 2018. Retrieved 27 June 2009.
59. "PyDBC: method preconditions, method postconditions and class invariants for Python" (http://www.nongnu.org/pydbc/). Archived (https://web.archive.org/web/20191123231931/http://www.nongnu.org/pydbc/) from the original on 23 November 2019. Retrieved 24 September 2011.
60. "Contracts for Python" (http://www.wayforward.net/pycontract/). Archived (https://web.archive.org/web/20200615173404/http://www.wayforward.net/pycontract/) from the original on 15 June 2020. Retrieved 24 September 2011.
61. "PyDatalog" (https://sites.google.com/site/pydatalog/). Archived (https://web.archive.org/web/20200613160231/https://sites.google.com/site/pydatalog/) from the original on 13 June 2020. Retrieved 22 July 2012.
62. "Glue it all together with Python" (https://www.python.org/doc/essays/omg-darpa-mcc-position/). *Python.org*. Retrieved 30 September 2024.
63. "Reference counts" (https://docs.python.org/extending/extending.html#reference-counts). Extending and embedding the Python interpreter. *Docs.python.org*. Archived (https://web.archive.org/web/20121018063230/http://docs.python.org/extending/extending.html#reference-counts) from the original on 18 October 2012. Retrieved 5 June 2020. "Since Python makes heavy use of `malloc()` and `free()`}, it needs a strategy to avoid memory leaks as well as the re-use of freed memory. The method chosen is called *reference counting*."
64. Hettinger, Raymond (30 January 2002). "PEP 289 – Generator Expressions" (https://www.python.org/dev/peps/pep-0289/). *Python Enhancement Proposals*. Python Software Foundation. Archived (https://web.archive.org/web/20200614153717/https://www.python.org/dev/peps/pep-0289/) from the original on 14 June 2020. Retrieved 19 February 2012.
65. "6.5 itertools – Functions creating iterators for efficient looping" (https://docs.python.org/3/library/itertools.html). Docs.python.org. Archived (https://web.archive.org/web/20200614153629/https://docs.python.org/3/library/itertools.html) from the original on 14 June 2020. Retrieved 22 November 2016.
66. 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. Archived (https://web.archive.org/web/20181226141127/https://www.python.org/dev/peps/pep-0020/) from the original on 26 December 2018. Retrieved 24 November 2008.

67. Lutz, Mark (January 2022). "Python changes 2014+" (https://learning-python.com/python-changes-2014-plus.html). *Learning Python*. Archived (https://web.archive.org/web/20240315075935/https://learning-python.com/python-changes-2014-plus.html) from the original on 15 March 2024. Retrieved 25 February 2024.
68. "Confusion regarding a rule in 'the Zen of Python' " (https://discuss.python.org/t/confusion-regarding-a-rule-in-the-zen-of-python/15927). Discussions. *Python.org*. Python help. 3 May 2022. Archived (https://web.archive.org/web/20240225221142/https://discuss.python.org/t/confusion-regarding-a-rule-in-the-zen-of-python/15927) from the original on 25 February 2024. Retrieved 25 February 2024.
69. Ambi, Chetan (4 July 2021). "The most controversial Python 'walrus operator' " (https://pythonsimplified.com/the-most-controversial-python-walrus-operator/). *Python simplified (pythonsimplified.com)*. Archived (https://web.archive.org/web/20230827154931/https://pythonsimplified.com/the-most-controversial-python-walrus-operator/) from the original on 27 August 2023. Retrieved 5 February 2024.
70. Grifski, Jeremy (24 May 2020). "The controversy behind the 'walrus operator' in Python" (https://therenegadecoder.com/code/the-controversy-behind-the-walrus-operator-in-python/). *The Renegade Coder (therenegadecoder.com)*. Archived (https://web.archive.org/web/20231228135749/https://therenegadecoder.com/code/the-controversy-behind-the-walrus-operator-in-python/) from the original on 28 December 2023. Retrieved 25 February 2024.
71. Bader, Dan. "Python string formatting best practices" (https://realpython.com/python-string-formatting/). *Real Python (realpython.com)*. Archived (https://web.archive.org/web/20240218083506/https://realpython.com/python-string-formatting/) from the original on 18 February 2024. Retrieved 25 February 2024.
72. 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. Archived (https://web.archive.org/web/20200223171254/http://shop.oreilly.com/product/9780596007973.do) from the original on 23 February 2020. Retrieved 14 November 2015.
73. "Python Culture" (https://web.archive.org/web/20140130021902/http://ebeab.com/2014/01/21/python-culture/). *ebeab*. 21 January 2014. Archived from the original (http://ebeab.com/2014/01/21/python-culture/) on 30 January 2014.
74. "Transpiling Python to Julia using PyJL" (https://web.ist.utl.pt/antonio.menezes.leitao/ADA/documents/publications_docs/2022_TranspilingPythonToJuliaUsingPyJL.pdf) (PDF). Archived (https://web.archive.org/web/20231119071525/https://web.ist.utl.pt/antonio.menezes.leitao/ADA/documents/publications_docs/2022_TranspilingPythonToJuliaUsingPyJL.pdf) (PDF) from the original on 19 November 2023. Retrieved 20 September 2023. "After manually modifying one line of code by specifying the necessary type information, we obtained a speedup of 52.6×, making the translated Julia code 19.5× faster than the original Python code."
75. "Why is it called Python?" (https://docs.python.org/3/faq/general.html#why-is-it-called-python). *General Python FAQ*. Docs.python.org. Archived (https://web.archive.org/web/20121024164224/http://docs.python.org/faq/general.html#why-is-it-called-python) from the original on 24 October 2012. Retrieved 3 January 2023.
76. "15 ways Python is a powerful force on the web" (https://web.archive.org/web/20190511065650/http://insidetech.monster.com/training/articles/8114-15-ways-python-is-a-powerful-force-on-the-web). Archived from the original (https://insidetech.monster.com/training/articles/8114-15-ways-python-is-a-powerful-force-on-the-web) on 11 May 2019. Retrieved 3 July 2018.
77. "`pprint` – data pretty printer – Python 3.11.0 documentation" (https://docs.python.org/3/library/pprint.html). *docs.python.org*. Archived (https://web.archive.org/web/20210122224848/https://docs.python.org/3/library/pprint.html) from the original on 22 January 2021. Retrieved 5 November 2022. "`stuff = ['spam', 'eggs', 'lumberjack', 'knights', 'ni']`"

78. "Code style" (https://docs.python-guide.org/writing/style). The hitchhiker's guide to Python. *docs.python-guide.org*. Archived (https://web.archive.org/web/20210127154341/https://docs.python-guide.org/writing/style/) from the original on 27 January 2021. Retrieved 20 January 2021.
79. "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. Archived (https://web.archive.org/web/20121024164224/http://docs.python.org/faq/general.html#is-python-a-good-language-for-beginning-programmers) from the original on 24 October 2012. Retrieved 21 March 2007.
80. "Myths about indentation in Python" (https://web.archive.org/web/20180218162410/http://www.secnetix.de/~olli/Python/block_indentation.hawk). Secnetix.de. Archived from the original (http://www.secnetix.de/~olli/Python/block_indentation.hawk) on 18 February 2018. Retrieved 19 April 2011.
81. 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.
82. van Rossum, Guido; Warsaw, Barry. "PEP 8 – Style Guide for Python Code" (https://www.python.org/dev/peps/pep-0008/). *Python Enhancement Proposals (PEPs)*. Archived (https://web.archive.org/web/20190417223549/https://www.python.org/dev/peps/pep-0008/) from the original on 17 April 2019. Retrieved 26 March 2019.
83. "8. Errors and Exceptions – Python 3.12.0a0 documentation" (https://docs.python.org/3.11/tutorial/errors.html). *docs.python.org*. Archived (https://web.archive.org/web/20220509145745/https://docs.python.org/3.11/tutorial/errors.html) from the original on 9 May 2022. Retrieved 9 May 2022.
84. "Highlights: Python 2.5" (https://www.python.org/download/releases/2.5/highlights/). *Python.org*. Archived (https://web.archive.org/web/20190804120408/https://www.python.org/download/releases/2.5/highlights/) from the original on 4 August 2019. Retrieved 20 March 2018.
85. van Rossum, Guido (22 April 2009). "Tail Recursion Elimination" (http://neopythonic.blogspot.be/2009/04/tail-recursion-elimination.html). Neopythonic.blogspot.be. Archived (https://web.archive.org/web/20180519225253/http://neopythonic.blogspot.be/2009/04/tail-recursion-elimination.html) from the original on 19 May 2018. Retrieved 3 December 2012.
86. 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. Archived (https://web.archive.org/web/20200117182525/https://www.artima.com/weblogs/viewpost.jsp?thread=147358) from the original on 17 January 2020. Retrieved 21 March 2007.
87. 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. Archived (https://web.archive.org/web/20200529003739/https://www.python.org/dev/peps/pep-0342/) from the original on 29 May 2020. Retrieved 19 February 2012.
88. "PEP 380" (https://www.python.org/dev/peps/pep-0380/). Python.org. Archived (https://web.archive.org/web/20200604233821/https://www.python.org/dev/peps/pep-0380/) from the original on 4 June 2020. Retrieved 3 December 2012.
89. "division" (https://docs.python.org). *python.org*. Archived (https://web.archive.org/web/20060720033244/http://docs.python.org/) from the original on 20 July 2006. Retrieved 30 July 2014.
90. "PEP 0465 – A dedicated infix operator for matrix multiplication" (https://www.python.org/dev/peps/pep-0465/). *python.org*. Archived (https://web.archive.org/web/20200604224255/https://www.python.org/dev/peps/pep-0465/) from the original on 4 June 2020. Retrieved 1 January 2016.

91. "Python 3.5.1 Release and Changelog" (https://www.python.org/downloads/release/python-351/). *python.org*. Archived (https://web.archive.org/web/20200514034938/https://www.python.org/downloads/release/python-351/) from the original on 14 May 2020. Retrieved 1 January 2016.
92. "What's New in Python 3.8" (https://docs.python.org/3.8/whatsnew/3.8.html). Archived (https://web.archive.org/web/20200608124345/https://docs.python.org/3.8/whatsnew/3.8.html) from the original on 8 June 2020. Retrieved 14 October 2019.
93. 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. Archived (https://web.archive.org/web/20160313113147/https://www.python.org/dev/peps/pep-0308/) from the original on 13 March 2016. Retrieved 13 July 2011.
94. "4. Built-in Types – Python 3.6.3rc1 documentation" (https://docs.python.org/3/library/stdtypes.html#tuple). *python.org*. Archived (https://web.archive.org/web/20200614194325/https://docs.python.org/3/library/stdtypes.html#tuple) from the original on 14 June 2020. Retrieved 1 October 2017.
95. "5.3. Tuples and Sequences – Python 3.7.1rc2 documentation" (https://docs.python.org/3/tutorial/datastructures.html#tuples-and-sequences). *python.org*. Archived (https://web.archive.org/web/20200610050047/https://docs.python.org/3/tutorial/datastructures.html#tuples-and-sequences) from the original on 10 June 2020. Retrieved 17 October 2018.
96. "PEP 498 – Literal String Interpolation" (https://www.python.org/dev/peps/pep-0498/). *python.org*. Archived (https://web.archive.org/web/20200615184141/https://www.python.org/dev/peps/pep-0498/) from the original on 15 June 2020. Retrieved 8 March 2017.
97. "The Python Language Reference, section 3.3. New-style and classic classes, for release 2.7.1" (https://web.archive.org/web/20121026063834/http://docs.python.org/reference/datamodel.html#new-style-and-classic-classes). Archived from the original (https://docs.python.org/reference/datamodel.html#new-style-and-classic-classes) on 26 October 2012. Retrieved 12 January 2011.
98. "PEP 484 – Type Hints | peps.python.org" (https://peps.python.org/pep-0484/). *peps.python.org*. Archived (https://web.archive.org/web/20231127205023/https://peps.python.org/pep-0484/) from the original on 27 November 2023. Retrieved 29 November 2023.
99. "typing — Support for type hints" (https://docs.python.org/3/library/typing.html). *Python documentation*. Python Software Foundation. Archived (https://web.archive.org/web/20200221184042/https://docs.python.org/3/library/typing.html) from the original on 21 February 2020. Retrieved 22 December 2023.
100. "mypy – Optional Static Typing for Python" (http://mypy-lang.org/). Archived (https://web.archive.org/web/20200606192012/http://mypy-lang.org/) from the original on 6 June 2020. Retrieved 28 January 2017.
101. "Introduction" (https://mypyc.readthedocs.io/en/latest/introduction.html). *mypyc.readthedocs.io*. Archived (https://web.archive.org/web/20231222000457/https://mypyc.readthedocs.io/en/latest/introduction.html) from the original on 22 December 2023. Retrieved 22 December 2023.
102. "15. Floating Point Arithmetic: Issues and Limitations – Python 3.8.3 documentation" (https://docs.python.org/3.8/tutorial/floatingpoint.html#representation-error). *docs.python.org*. Archived (https://web.archive.org/web/20200606113842/https://docs.python.org/3.8/tutorial/floatingpoint.html#representation-error) from the original on 6 June 2020. Retrieved 6 June 2020. "Almost all machines today (November 2000) use IEEE-754 floating point arithmetic, and almost all platforms map Python floats to IEEE-754 "double precision"."

103. 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. Archived (https://web.archive.org/web/20200528063237/https://www.python.org/dev/peps/pep-0237/) from the original on 28 May 2020. Retrieved 24 September 2011.
104. "Built-in Types" (https://docs.python.org/3/library/stdtypes.html#typesseq-range). Archived (https://web.archive.org/web/20200614194325/https://docs.python.org/3/library/stdtypes.html#typesseq-range) from the original on 14 June 2020. Retrieved 3 October 2019.
105. "PEP 465 – A dedicated infix operator for matrix multiplication" (https://legacy.python.org/dev/peps/pep-0465/). *python.org*. Archived (https://web.archive.org/web/20200529200310/https://legacy.python.org/dev/peps/pep-0465/) from the original on 29 May 2020. Retrieved 3 July 2018.
106. 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. Archived (https://web.archive.org/web/20200528115550/https://www.python.org/dev/peps/pep-0238/) from the original on 28 May 2020. Retrieved 23 October 2013.
107. "Why Python's Integer Division Floors" (https://python-history.blogspot.com/2010/08/why-pythons-integer-division-floors.html). 24 August 2010. Archived (https://web.archive.org/web/20200605151500/https://python-history.blogspot.com/2010/08/why-pythons-integer-division-floors.html) from the original on 5 June 2020. Retrieved 25 August 2010.
108. "round" (https://docs.python.org/py3k/library/functions.html#round), *The Python standard library, release 3.2, §2: Built-in functions*, archived (https://web.archive.org/web/20121025141808/http://docs.python.org/py3k/library/functions.html#round) from the original on 25 October 2012, retrieved 14 August 2011
109. "round" (https://docs.python.org/library/functions.html#round), *The Python standard library, release 2.7, §2: Built-in functions*, archived (https://web.archive.org/web/20121027081602/http://docs.python.org/library/functions.html#round) from the original on 27 October 2012, retrieved 14 August 2011
110. Beazley, David M. (2009). *Python Essential Reference* (https://archive.org/details/pythonessentialr00beaz_036) (4th ed.). Addison-Wesley Professional. p. 66 (https://archive.org/details/pythonessentialr00beaz_036/page/n90). ISBN 978-0-672-32978-4.
111. Kernighan, Brian W.; Ritchie, Dennis M. (1988). *The C Programming Language* (2nd ed.). p. 206 (https://archive.org/details/cprogramminglang00bria/page/206).
112. Batista, Facundo (17 October 2003). "PEP 327 – Decimal Data Type" (https://www.python.org/dev/peps/pep-0327/). *Python Enhancement Proposals*. Python Software Foundation. Archived (https://web.archive.org/web/20200604234830/https://www.python.org/dev/peps/pep-0327/) from the original on 4 June 2020. Retrieved 24 November 2008.
113. "What's New in Python 2.6" (https://docs.python.org/2.6/whatsnew/2.6.html). *Python v2.6.9 documentation*. 29 October 2013. Archived (https://web.archive.org/web/20191223213856/https://docs.python.org/2.6/whatsnew/2.6.html) from the original on 23 December 2019. Retrieved 26 September 2015.
114. "10 Reasons Python Rocks for Research (And a Few Reasons it Doesn't) – Hoyt Koepke" (https://web.archive.org/web/20200531211840/https://www.stat.washington.edu/~hoytak/blog/whypython.html). *University of Washington Department of Statistics*. Archived from the original (https://www.stat.washington.edu/~hoytak/blog/whypython.html) on 31 May 2020. Retrieved 3 February 2019.
115. Shell, Scott (17 June 2014). "An introduction to Python for scientific computing" (https://engineering.ucsb.edu/~shell/che210d/python.pdf) (PDF). Archived (https://web.archive.org/web/20190204014642/https://engineering.ucsb.edu/~shell/che210d/python.pdf) (PDF) from the original on 4 February 2019. Retrieved 3 February 2019.

116. 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. Archived (https://web.archive.org/web/20190402124435/https://www.oracle.com/technetwork/articles/piotrowski-pythoncore-084049.html) from the original on 2 April 2019. Retrieved 12 March 2012.
117. 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. Archived (https://web.archive.org/web/20200614170344/https://www.python.org/dev/peps/pep-0333/) from the original on 14 June 2020. Retrieved 19 February 2012.
118. "PyPI" (https://pypi.org/). *PyPI*. 13 March 2025. Archived (https://web.archive.org/web/20250222013445/https://pypi.org/) from the original on 22 February 2025.
119. "Glossary: interactive" (https://docs.python.org/3/glossary.html#term-interactive). *Python documentation*. v3.13.7. Retrieved 31 August 2025.
120. "IDLE — Python editor and shell" (https://docs.python.org/3/library/idle.html). *Python documentation*. v3.13.7. Retrieved 31 August 2025. "IDLE is Python's Integrated Development and Learning Environment."
121. "IPython Documentation" (https://ipython.readthedocs.io/en/stable/). v9.5.0. 29 August 2025. Archived (https://web.archive.org/web/20250831204721/https://ipython.readthedocs.io/en/stable/) from the original on 31 August 2025. Retrieved 31 August 2025.
122. "Project Jupyter" (https://jupyter.org). *Jupyter.org*. Archived (https://web.archive.org/web/20231012055917/https://jupyter.org/) from the original on 12 October 2023. Retrieved 2 April 2025.
123. Harper, Doug (Spring 2024). "Enthought Canopy" (http://physics.wku.edu/phys316/software/canopy/). *WKU Physics 316*. Western Kentucky University. Archived (https://web.archive.org/web/20240818041226/http://physics.wku.edu/phys316/software/canopy/) from the original on 18 August 2024. Retrieved 31 August 2025.
124. "Enthought Canopy" (https://web.archive.org/web/20170715151703/https://www.enthought.com/products/canopy/). *Enthought*. Archived from the original (https://www.enthought.com/products/canopy/) on 15 July 2017. Retrieved 20 August 2016.
125. "PEP 7 – Style Guide for C Code | peps.python.org" (https://peps.python.org/pep-0007/). *peps.python.org*. Archived (https://web.archive.org/web/20220424202827/https://peps.python.org/pep-0007/) from the original on 24 April 2022. Retrieved 28 April 2022.
126. "4. Building C and C++ Extensions – Python 3.9.2 documentation" (https://docs.python.org/3/extending/building.html). *docs.python.org*. Archived (https://web.archive.org/web/20210303002519/https://docs.python.org/3/extending/building.html) from the original on 3 March 2021. Retrieved 1 March 2021.
127. 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. Archived (https://web.archive.org/web/20200601203908/https://www.python.org/dev/peps/pep-0007/) from the original on 1 June 2020. Retrieved 24 November 2008.
128. "CPython byte code" (https://docs.python.org/3/library/dis.html#python-bytecode-instructions). Docs.python.org. Archived (https://web.archive.org/web/20200605151542/https://docs.python.org/3/library/dis.html#python-bytecode-instructions) from the original on 5 June 2020. Retrieved 16 February 2016.
129. "Python 2.5 internals" (http://www.troeger.eu/teaching/pythonvm08.pdf) (PDF). Archived (https://web.archive.org/web/20120806094951/http://www.troeger.eu/teaching/pythonvm08.pdf) (PDF) from the original on 6 August 2012. Retrieved 19 April 2011.
130. "Changelog – Python 3.9.0 documentation" (https://docs.python.org/release/3.9.0/whatsnew/changelog.html#changelog). *docs.python.org*. Archived (https://web.archive.org/web/20210207001142/https://docs.python.org/release/3.9.0/whatsnew/changelog.html#changelog) from the original on 7 February 2021. Retrieved 8 February 2021.

131. "Download Python" (https://www.python.org/downloads/release/python-391). *Python.org*. Archived (https://web.archive.org/web/20201208045225/https://www.python.org/downloads/release/python-391/) from the original on 8 December 2020. Retrieved 13 December 2020.
132. "history [vmspython]" (https://www.vmspython.org/doku.php?id=history). *www.vmspython.org*. Archived (https://web.archive.org/web/20201202194743/https://www.vmspython.org/doku.php?id=history) from the original on 2 December 2020. Retrieved 4 December 2020.
133. "An Interview with Guido van Rossum" (http://www.oreilly.com/pub/a/oreilly/frank/rossum_1099.html). Oreilly.com. Archived (https://web.archive.org/web/20140716222652/http://oreilly.com/pub/a/oreilly/frank/rossum_1099.html) from the original on 16 July 2014. Retrieved 24 November 2008.
134. "PyPy compatibility" (https://pypy.org/compat.html). Pypy.org. Archived (https://web.archive.org/web/20200606041845/https://www.pypy.org/compat.html) from the original on 6 June 2020. Retrieved 3 December 2012.
135. Team, The PyPy (28 December 2019). "Download and Install" (https://www.pypy.org/download.html). *PyPy*. Archived (https://web.archive.org/web/20220108212951/https://www.pypy.org/download.html) from the original on 8 January 2022. Retrieved 8 January 2022.
136. "speed comparison between CPython and Pypy" (https://speed.pypy.org/). Speed.pypy.org. Archived (https://web.archive.org/web/20210510014902/https://speed.pypy.org/) from the original on 10 May 2021. Retrieved 3 December 2012.
137. "Codon: Differences with Python" (https://docs.exaloop.io/codon/general/differences). Archived (https://web.archive.org/web/20230525002540/https://docs.exaloop.io/codon/general/differences) from the original on 25 May 2023. Retrieved 28 August 2023.
138. Lawson, Loraine (14 March 2023). "MIT-Created Compiler Speeds up Python Code" (https://thenewstack.io/mit-created-compiler-speeds-up-python-code/). *The New Stack*. Archived (https://web.archive.org/web/20230406054200/https://thenewstack.io/mit-created-compiler-speeds-up-python-code/) from the original on 6 April 2023. Retrieved 28 August 2023.
139. "Python-for-EV3" (https://education.lego.com/en-us/support/mindstorms-ev3/python-for-ev3). *LEGO Education*. Archived (https://web.archive.org/web/20200607234814/https://education.lego.com/en-us/support/mindstorms-ev3/python-for-ev3) from the original on 7 June 2020. Retrieved 17 April 2019.
140. Yegulalp, Serdar (29 October 2020). "Pyston returns from the dead to speed Python" (https://www.infoworld.com/article/3587591/pyston-returns-from-the-dead-to-speed-python.html). *InfoWorld*. Archived (https://web.archive.org/web/20210127113233/https://www.infoworld.com/article/3587591/pyston-returns-from-the-dead-to-speed-python.html) from the original on 27 January 2021. Retrieved 26 January 2021.
141. "cinder: Instagram's performance-oriented fork of CPython" (https://github.com/facebookincubator/cinder). *GitHub*. Archived (https://web.archive.org/web/20210504112500/https://github.com/facebookincubator/cinder) from the original on 4 May 2021. Retrieved 4 May 2021.
142. Aroca, Rafael (7 August 2021). "Snek Lang: feels like Python on Arduinos" (https://rafaelaroca.wordpress.com/2021/08/07/snek-lang-feels-like-python-on-arduinos/). *Yet Another Technology Blog*. Archived (https://web.archive.org/web/20240105001031/https://rafaelaroca.wordpress.com/2021/08/07/snek-lang-feels-like-python-on-arduinos/) from the original on 5 January 2024. Retrieved 4 January 2024.
143. Aufranc (CNXSoft), Jean-Luc (16 January 2020). "Snekboard Controls LEGO Power Functions with CircuitPython or Snek Programming Languages (Crowdfunding) – CNX Software" (https://www.cnx-software.com/2020/01/16/snekboard-controls-lego-power-functions-with-circuitpython-or-snek-programming-languages/). *CNX Software – Embedded Systems News*. Archived (https://web.archive.org/web/20240105001031/https://www.cnx-software.com/2020/01/16/snekboard-controls-lego-power-functions-with-circuitpython-or-snek-programming-languages/) from the original on 5 January 2024. Retrieved 4 January 2024.

144. Kennedy (@mkennedy), Michael. "Ready to find out if you're git famous?" (https://pythonbytes.fm/episodes/show/187/ready-to-find-out-if-youre-git-famous). *pythonbytes.fm*. Archived (https://web.archive.org/web/20240105001031/https://pythonbytes.fm/episodes/show/187/ready-to-find-out-if-youre-git-famous) from the original on 5 January 2024. Retrieved 4 January 2024.
145. Packard, Keith (20 December 2022). "The Snek Programming Language: A Python-inspired Embedded Computing Language" (https://sneklang.org/doc/snek.pdf) (PDF). Archived (https://web.archive.org/web/20240104162458/https://sneklang.org/doc/snek.pdf) (PDF) from the original on 4 January 2024. Retrieved 4 January 2024.
146. "Application-level Stackless features – PyPy 2.0.2 documentation" (http://doc.pypy.org/en/latest/stackless.html). Doc.pypy.org. Archived (https://web.archive.org/web/20200604231513/https://doc.pypy.org/en/latest/stackless.html) from the original on 4 June 2020. Retrieved 17 July 2013.
147. "Plans for optimizing Python" (https://code.google.com/p/unladen-swallow/wiki/ProjectPlan). *Google Project Hosting*. 15 December 2009. Archived (https://web.archive.org/web/20160411181848/https://code.google.com/p/unladen-swallow/wiki/ProjectPlan) from the original on 11 April 2016. Retrieved 24 September 2011.
148. "Python on the Nokia N900" (http://www.stochasticgeometry.ie/2010/04/29/python-on-the-nokia-n900/). *Stochastic Geometry*. 29 April 2010. Archived (https://web.archive.org/web/20190620000053/http://www.stochasticgeometry.ie/2010/04/29/python-on-the-nokia-n900/) from the original on 20 June 2019. Retrieved 9 July 2015.
149. "Brython" (https://brython.info/). *brython.info*. Archived (https://web.archive.org/web/20180803065954/http://brython.info/) from the original on 3 August 2018. Retrieved 21 January 2021.
150. "Transcrypt – Python in the browser" (https://www.transcrypt.org). *transcrypt.org*. Archived (https://web.archive.org/web/20180819133303/http://www.transcrypt.org/) from the original on 19 August 2018. Retrieved 22 December 2020.
151. "Transcrypt: Anatomy of a Python to JavaScript Compiler" (https://www.infoq.com/articles/transcrypt-python-javascript-compiler/). *InfoQ*. Archived (https://web.archive.org/web/20201205193339/https://www.infoq.com/articles/transcrypt-python-javascript-compiler/) from the original on 5 December 2020. Retrieved 20 January 2021.
152. "Nuitka Home | Nuitka Home" (http://nuitka.net/). *nuitka.net*. Archived (https://web.archive.org/web/20200530211233/https://nuitka.net/) from the original on 30 May 2020. Retrieved 18 August 2017.
153. Guelton, Serge; Brunet, Pierrick; Amini, Mehdi; Merlini, Adrien; Corbillon, Xavier; Raynaud, Alan (16 March 2015). "Pythran: enabling static optimization of scientific Python programs" (https://doi.org/10.1088%2F1749-4680%2F8%2F1%2F014001). *Computational Science & Discovery*. **8** (1) 014001. IOP Publishing. Bibcode:2015CS&D....8a4001G (https://ui.adsabs.harvard.edu/abs/2015CS&D....8a4001G). doi:10.1088/1749-4680/8/1/014001 (https://doi.org/10.1088%2F1749-4680%2F8%2F1%2F014001). ISSN 1749-4699 (https://search.worldcat.org/issn/1749-4699).
154. "The Python → 11l → C++ transpiler" (https://11l-lang.org/transpiler). Archived (https://web.archive.org/web/20220924233728/https://11l-lang.org/transpiler/) from the original on 24 September 2022. Retrieved 17 July 2022.
155. "google/grumpy" (https://github.com/google/grumpy). 10 April 2020. Archived (https://web.archive.org/web/20200415054919/https://github.com/google/grumpy) from the original on 15 April 2020. Retrieved 25 March 2020 – via GitHub.
156. "Projects" (https://opensource.google/projects/). *opensource.google*. Archived (https://web.archive.org/web/20200424191248/https://opensource.google/projects/) from the original on 24 April 2020. Retrieved 25 March 2020.

157. Francisco, Thomas Claburn in San. "Google's Grumpy code makes Python Go" (https://www.theregister.com/2017/01/05/googles_grumpy_makes_python_go/). *www.theregister.com*. Archived (https://web.archive.org/web/20210307165521/https://www.theregister.com/2017/01/05/googles_grumpy_makes_python_go/) from the original on 7 March 2021. Retrieved 20 January 2021.
158. "IronPython.net /" (https://ironpython.net/). *ironpython.net*. Archived (https://web.archive.org/web/20210417064418/https://ironpython.net/) from the original on 17 April 2021.
159. "GitHub – IronLanguages/ironpython3: Implementation of Python 3.x for .NET Framework that is built on top of the Dynamic Language Runtime" (https://github.com/IronLanguages/ironpython3). *GitHub*. Archived (https://web.archive.org/web/20210928101250/https://github.com/IronLanguages/ironpython3) from the original on 28 September 2021.
160. "Jython FAQ" (https://www.jython.org/jython-old-sites/archive/22/userfaq.html). *www.jython.org*. Archived (https://web.archive.org/web/20210422055726/https://www.jython.org/jython-old-sites/archive/22/userfaq.html) from the original on 22 April 2021. Retrieved 22 April 2021.
161. 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).
162. "The Computer Language Benchmarks Game" (https://benchmarksgame-team.pages.debian.net/benchmarksgame/fastest/python.html). Archived (https://web.archive.org/web/20200614210246/https://benchmarksgame-team.pages.debian.net/benchmarksgame/fastest/python.html) from the original on 14 June 2020. Retrieved 30 April 2020.
163. Python, Real. "Look Ma, No for Loops: Array Programming With NumPy – Real Python" (https://realpython.com/numpy-array-programming/). *realpython.com*. Retrieved 15 October 2025.
164. 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. Archived (https://web.archive.org/web/20200606042011/https://www.python.org/dev/peps/pep-0001/) from the original on 6 June 2020. Retrieved 19 April 2011.
165. 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.
166. Edge, Jake (23 February 2022). "Moving Python's bugs to GitHub [LWN.net]" (https://lwn.net/Articles/885854/). Archived (https://web.archive.org/web/20221002183818/https://lwn.net/Articles/885854/) from the original on 2 October 2022. Retrieved 2 October 2022.
167. "Python Developer's Guide – Python Developer's Guide" (https://devguide.python.org/). *devguide.python.org*. Archived (https://web.archive.org/web/20201109032501/https://devguide.python.org/) from the original on 9 November 2020. Retrieved 17 December 2019.
168. Hughes, Owen (24 May 2021). "Programming languages: Why Python 4.0 might never arrive, according to its creator" (https://www.techrepublic.com/article/programming-languages-why-python-4-0-will-probably-never-arrive-according-to-its-creator/). *TechRepublic*. Archived (https://web.archive.org/web/20220714201302/https://www.techrepublic.com/article/programming-languages-why-python-4-0-will-probably-never-arrive-according-to-its-creator/) from the original on 14 July 2022. Retrieved 16 May 2022.
169. "PEP 602 – Annual Release Cycle for Python" (https://www.python.org/dev/peps/pep-0602/). *Python.org*. Archived (https://web.archive.org/web/20200614202755/https://www.python.org/dev/peps/pep-0602/) from the original on 14 June 2020. Retrieved 6 November 2019.

170. Edge, Jake (23 October 2019). "Changing the Python release cadence [LWN.net]" (https://lwn.net/Articles/802777/). *lwn.net*. Archived (https://web.archive.org/web/20191106170153/https://lwn.net/Articles/802777/) from the original on 6 November 2019. Retrieved 6 November 2019.
171. Norwitz, Neal (8 April 2002). "[Python-Dev] Release Schedules (was Stability & change)" (https://mail.python.org/pipermail/python-dev/2002-April/022739.html). Archived (https://web.archive.org/web/20181215122750/https://mail.python.org/pipermail/python-dev/2002-April/022739.html) from the original on 15 December 2018. Retrieved 27 June 2009.
172. 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. Archived (https://web.archive.org/web/20200605001318/https://www.python.org/dev/peps/pep-0006/) from the original on 5 June 2020. Retrieved 27 June 2009.
173. "Python Buildbot" (https://www.python.org/dev/buildbot/). *Python Developer's Guide*. Python Software Foundation. Archived (https://web.archive.org/web/20200605001322/https://www.python.org/dev/buildbot/) from the original on 5 June 2020. Retrieved 24 September 2011.
174. "Whetting Your Appetite" (https://docs.python.org/tutorial/appetite.html). *The Python Tutorial*. Python Software Foundation. Archived (https://web.archive.org/web/20121026063559/http://docs.python.org/tutorial/appetite.html) from the original on 26 October 2012. Retrieved 20 February 2012.
175. "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. Archived (https://web.archive.org/web/20190620000050/https://stackoverflow.com/questions/5033906/in-python-should-i-use-else-after-a-return-in-an-if-block) from the original on 20 June 2019. Retrieved 6 May 2011.
176. Lutz 2013, p. 17.
177. Fehily, Chris (2002). *Python* (https://books.google.com/books?id=carqdIdfVlYC&pg=PR15). Peachpit Press. p. xv. ISBN 978-0-201-74884-0. Archived (https://web.archive.org/web/20170717044040/https://books.google.com/books?id=carqdIdfVlYC&pg=PR15) from the original on 17 July 2017. Retrieved 9 May 2017.
178. Lubanovic, Bill (2014). *Introducing Python* (http://archive.org/details/introducingpytho0000luba). Sebastopol, CA : O'Reilly Media. p. 305. ISBN 978-1-4493-5936-2. Retrieved 31 July 2023.

Sources

- "Python for Artificial Intelligence" (https://web.archive.org/web/20121101045354/http://wiki.python.org/moin/PythonForArtificialIntelligence). Python Wiki. 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" (https://web.archive.org/web/20120326105810/http://www.ainewsletter.com/newsletters/aix_0508.htm#python_ai_ai). *AI Expert Newsletter*. Amzi!. Archived from the original (http://www.ainewsletter.com/newsletters/aix_0508.htm#python_ai_ai) on 26 March 2012. 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 (July 2024). *Think Python: How to Think Like a Computer Scientist* (https://allendowney.github.io/ThinkPython/) (3rd ed.). O'Reilly Media. ISBN 978-1-0981-5543-8.
- Lutz, Mark (2013). *Learning Python* (5th ed.). O'Reilly Media. ISBN 978-0-596-15806-4.
- Summerfield, Mark (2009). *Programming in Python 3* (2nd ed.). Addison-Wesley Professional. ISBN 978-0-321-68056-3.
- Ramalho, Luciano (May 2022). *Fluent Python* (https://www.thoughtworks.com/insights/books/fluent-python-2nd-edition). O'Reilly Media. ISBN 978-1-4920-5632-4.

External links

- Official website (https://www.python.org/)
- The Python Tutorial (https://docs.python.org/3/tutorial/)