Python (programming language)

From Wikipedia, the free encyclopedia

Python is a widely used high-level programming language for general-purpose programming, created by Guido van Rossum and first released in 1991. An interpreted language, Python has a design philosophy which emphasizes code readability (notably using whitespace indentation to delimit code blocks rather than curly brackets or keywords), and a syntax which allows programmers to express concepts in fewer lines of code than might be used in languages such as C++ or Java.[22][23] The language provides constructs intended to enable writing clear programs on both a small and large scale.[24]

Python features a dynamic type system and automatic memory management and supports multiple programming paradigms, including object-oriented, imperative, functional programming, and procedural styles. It has a large and comprehensive standard library.[25]

Python interpreters are available for many operating systems, allowing Python code to run on a wide variety of systems. CPython, the reference implementation of Python, is open source software[26] and has a community-based development model, as do nearly all of its variant implementations. CPython is managed by the non-profit Python Software Foundation.

Contents

Python

Paradigm	multi-paradigm: object-oriented, imperative, functional, procedural, reflective
Designed by	Guido van Rossum
Developer	Python Software Foundation
First appeared	20 February 1991[1]
Stable release	3.6.1 / 21 March 2017[2] 2.7.13 / 17 December 2016[3]
Typing discipline	duck, dynamic, strong
OS	Cross-platform
License	Python Software Foundation License
Filename extensions	.py, .pyc, .pyd, .pyo (prior to 3.5),[4] .pyw, .pyz (since 3.5)[5]
Website	www.python.org
Major implementations	
CPython, IronPython, Jython, MicroPython, Numba, PyPy	
Dialects	
Cython, RPython, Stackless Python	
Influenced by	
ABC,[6] ALGOL 68,[7] C,[8] C++,[9] Dylan,[10] Haskell,[11] Icon,[12] Java,[13] Lisp,[14] Modula-3,[9] Perl	
Influenced	
Boo, Cobra, CoffeeScript,[15] D, F#, Falcon, Genie,[16] Go, Groovy, JavaScript,[17][18] Julia,[19] Nim, Ruby,[20] Swift[21]	

Python Programming at Wikibooks

History

Python was conceived in the late 1980s,[27] and its implementation began in December 1989[28] by Guido van Rossum at Centrum Wiskunde & Informatica (CWI) in the Netherlands as a successor to the ABC language (itself inspired by SETL)[29] capable of exception handling and interfacing with the operating system Amoeba.[6] Van Rossum is Python's principal author, and his continuing central role in deciding the direction of Python is reflected in the title given to him by the Python community, *benevolent dictator for life* (BDFL).


Guido van Rossum, the creator of Python

About the origin of Python, Van Rossum wrote in 1996:[30]

> Over six years ago, in December 1989, I was looking for a "hobby" programming project that would keep me occupied during the week around Christmas. My office ... would be closed, but I had a home computer, and not much else on my hands. I decided to write an interpreter for the new scripting language I had been thinking about lately: a descendant of ABC that would appeal to Unix/C hackers. I chose Python as a working title for the project, being in a slightly irreverent mood (and a big fan of *Monty Python's Flying Circus*).

Python 2.0 was released on 16 October 2000 and had many major new features, including a cycle-detecting garbage collector and support for Unicode. With this release the development process was changed and became more transparent and community-backed.[31]

Python 3.0 (which early in its development was commonly referred to as Python 3000 or py3k), a major, backwards-incompatible release, was released on 3 December 2008[32] after a long period of testing. Many of its major features have been backported to the backwards-compatible Python 2.6.x[33] and 2.7.x version series.

The End Of Life date (EOL, sunset date) for Python 2.7 was initially set at 2015, then postponed to 2020 out of concern that a large body of existing code cannot easily be forward-ported to Python 3.[34][35] In January 2017, Google announced work on a Python 2.7 to Go transcompiler, which The Register speculated was in response to Python 2.7's planned end-of-life[36] but Google cited performance under concurrent workloads as their only motivation.[37]

Features and philosophy

Python is a multi-paradigm programming language: object-oriented programming and structured programming are fully supported, and many language features support functional programming and aspect-oriented programming (including by metaprogramming[38] and metaobjects (magic methods)).[39] Many other paradigms are supported via extensions, including design by contract[40][41] and logic programming.[42]

Python uses dynamic typing and a mix of reference counting and a cycle-detecting garbage collector for memory management. An important feature of Python is dynamic name resolution (late binding), which binds method and variable names during program execution.

The design of Python offers some support for functional programming in the Lisp tradition. The language has map(), reduce() and filter() functions; list comprehensions, dictionaries, and sets; and generator expressions.[43] The standard library has two modules (itertools and functools) that implement functional tools borrowed from Haskell and Standard ML.[44]

The core philosophy of the language is summarized by the document *The Zen of Python* (*PEP 20*), which includes aphorisms such as:[45]

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

Rather than requiring all desired functionality to be built into the language's core, Python was designed to be highly extensible. Python can also be embedded in existing applications that need a programmable interface. This design of a small core language with a large standard library and an easily extensible interpreter was intended by Van Rossum from the start because of his frustrations with ABC, which espoused the opposite mindset.[27]

While offering choice in coding methodology, the Python philosophy rejects exuberant syntax, such as in Perl, in favor of a sparser, less-cluttered grammar. As Alex Martelli put it: "To describe something as clever is *not* considered a compliment in the Python culture."[46] Python's philosophy rejects the Perl "there is more than one way to do it" approach to language design in favor of "there should be one—and preferably only one—obvious way to do it".[45]

Python's developers strive to avoid premature optimization, and moreover, reject patches to non-critical parts of CPython that would offer a marginal increase in speed at the cost of clarity.[47] When speed is important, a Python programmer can move time-critical functions to extension modules written in languages such as C, or try using PyPy, a just-in-time compiler. Cython is also available, which translates a Python script into C and makes direct C-level API calls into the Python interpreter.

An important goal of Python's developers is making it fun to use. This is reflected in the origin of the name, which comes from Monty Python,[48] and in an occasionally playful approach to tutorials and reference materials, such as using examples that refer to spam and eggs instead of the standard foo and bar.[49][50]

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

Users and admirers of Python, especially those considered knowledgeable or experienced, are often referred to as *Pythonists*, *Pythonistas*, and *Pythoneers*.[51][52]

Syntax and semantics

Python is intended to be a highly readable language. It is designed to have an uncluttered visual layout, often using English keywords where other languages use punctuation. Python doesn't have semicolons and curly brackets "{}" which is different compared to most of the programming language. Further, Python has fewer

syntactic exceptions and special cases than C or Pascal.[53]

Indentation

Python uses whitespace indentation to delimit blocks – rather than curly braces or keywords. An increase in indentation comes after certain statements; a decrease in indentation signifies the end of the current block.[54] This feature is also sometimes termed the off-side rule.

Statements and control flow

Python's statements include (among others):

- The assignment statement (token '=', the equals sign). This operates differently than in traditional imperative programming languages, and this fundamental mechanism (including the nature of Python's version of *variables*) illuminates many other features of the language. Assignment in C, e.g., `x = 2`, translates to "typed variable name x receives a copy of numeric value 2". The (right-hand) value is copied into an allocated storage location for which the (left-hand) variable name is the symbolic address. The memory allocated to the variable is large enough (potentially quite large) for the declared type. In the simplest case of Python assignment, using the same example, `x = 2`, translates to "(generic) name x receives a reference to a separate, dynamically allocated object of numeric (int) type of value 2." This is termed *binding* the name to the object. Since the name's storage location doesn't *contain* the indicated value, it is improper to call it a *variable*. Names may be subsequently rebound at any time to objects of greatly varying types, including strings, procedures, complex objects with data and methods, etc. Successive assignments of a common value to multiple names, e.g., `x = 2; y = 2; z = 2` result in allocating storage to (at most) three names and one numeric object, to which all three names are bound. Since a name is a generic reference holder it is unreasonable to associate a fixed data type with it. However at a given time a name will be bound to *some* object, which **will** have a type; thus there is dynamic typing.
- The `if` statement, which conditionally executes a block of code, along with `else` and `elif` (a contraction of else-if).
- The `for` statement, which iterates over an iterable object, capturing each element to a local variable for use by the attached block.
- The `while` statement, which executes a block of code as long as its condition is true.
- The `try` statement, which allows exceptions raised in its attached code block to be caught and handled by `except` clauses; it also ensures that clean-up code in a `finally` block will always be run regardless of how the block exits.
- The `class` statement, which executes a block of code and attaches its local namespace to a class, for use in object-oriented programming.
- The `def` statement, which defines a function or method.
- The `with` statement (from Python 2.5), which encloses a code block within a context manager (for example, acquiring a lock before the block of code is run and releasing the lock afterwards, or opening a file and then closing it), allowing Resource Acquisition Is Initialization (RAII)-like behavior.
- The `pass` statement, which serves as a NOP. It is syntactically needed to create an empty code block.
- The `assert` statement, used during debugging to check for conditions that ought to apply.
- The `yield` statement, which returns a value from a generator function. From Python 2.5, `yield` is also an operator. This form is used to implement coroutines.
- The `import` statement, which is used to import modules whose functions or variables can be used in the current program.
- The `print` statement was changed to the `print()` function in Python 3.[55]

Python does not support tail call optimization or first-class continuations, and, according to Guido van Rossum, it never will.[56][57] However, better support for coroutine-like functionality is provided in 2.5, by extending Python's generators.[58] Before 2.5, generators were lazy iterators; information was passed unidirectionally out of the generator. As of Python 2.5, it is possible to pass information back into a generator function, and as of Python 3.3, the information can be passed through multiple stack levels.[59]

Expressions

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

- Addition, subtraction, and multiplication are the same, but the behavior of division differs. Python also added the `**` operator for exponentiation.
- As of Python 3.5, it supports matrix multiplication directly with the `@` operator, versus C and Java, which implement these as library functions. Earlier versions of Python also used methods instead of an infix operator.[60][61]
- In Python, `==` compares by value, versus Java, which compares numerics by value[62] and objects by reference.[63] (Value comparisons in Java on objects can be performed with the `equals()` method.) Python's `is` operator may be used to compare object identities (comparison by reference). In Python, comparisons may be chained, for example `a <= b <= c`.
- Python uses the words `and`, `or`, `not` for its boolean operators rather than the symbolic `&&`, `||`, `!` used in Java and C.
- Python has a type of expression termed a *list comprehension*. Python 2.4 extended list comprehensions into a more general expression termed a *generator expression*.[43]
- Anonymous functions are implemented using lambda expressions; however, these are limited in that the body can only be one expression.
- Conditional expressions in Python are written as `x if c else y`[64] (different in order of operands from the `c ? x : y` operator common to many other languages).
- Python makes a distinction between lists and tuples. Lists are written as `[1, 2, 3]`, are mutable, and cannot be used as the keys of dictionaries (dictionary keys must be immutable in Python). Tuples are written as `(1, 2, 3)`, are immutable and thus can be used as the keys of dictionaries, provided all elements of the tuple are immutable. The parentheses around the tuple are optional in some contexts. Tuples can appear on the left side of an equal sign; hence a statement like `x, y = y, x` can be used to swap two variables.
- Python has a "string format" operator `%`. This functions analogous to `printf` format strings in C, e.g. `"spam=%s eggs=%d" % ("blah", 2)` evaluates to `"spam=blah eggs=2"`. In Python 3 and 2.6+, this was supplemented by the `format()` method of the `str` class, e.g. `"spam={0} eggs={1}".format("blah", 2)`, Python 3.6 added "f-strings": `f'spam={"blah"} eggs={2}'`. [65]
- Python has various kinds of string literals:
  - Strings delimited by single or double quote marks. Unlike in Unix shells, Perl and Perl-influenced languages, single quote marks and double quote marks function identically. Both kinds of string use the backslash (`\`) as an escape character. String interpolation became available in Python 3.6 as "formatted string literals".[65]
  - Triple-quoted strings, which begin and end with a series of three single or double quote marks. They may span multiple lines and function like here documents in shells, Perl and Ruby.
  - Raw string varieties, denoted by prefixing the string literal with an `r`. Escape sequences are not interpreted; hence raw strings are useful where literal backslashes are common, such as regular expressions and Windows-style paths. Compare "@-quoting" in C#.
- Python has array index and array slicing expressions on lists, denoted as `a[key]`, `a[start:stop]` or `a[start:stop:step]`. Indexes are zero-based, and negative indexes are relative to the end. Slices take elements from the *start* index up to, but not including, the *stop* index. The third slice parameter, called *step* or *stride*, allows elements to be skipped and reversed. Slice indexes may be omitted, for example `a[:]` returns a copy of the entire list. Each element of a slice is a shallow copy.

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

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

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

Methods

Methods on objects are functions attached to the object's class; the syntax `instance.method(argument)` is, for normal methods and functions, syntactic sugar for `Class.method(instance, argument)`. Python methods have an explicit `self` parameter to access instance data, in contrast to the implicit `self` (or `this`) in some other object-oriented programming languages (e.g., C++, Java, Objective-C, or Ruby).[66]

Typing

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

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

Before version 3.0, Python had two kinds of classes: *old-style* and *new-style*.[67] The syntax of both styles is the same, the difference being whether the class `object` is inherited from, directly or indirectly (all new-style classes inherit from `object` and are instances of `type`). In versions of Python 2 from Python 2.2 onwards, both kinds of classes can be used. Old-style classes were eliminated in Python 3.0.

The long term plan is to support gradual typing[68] and as of Python 3.5, the syntax of the language allows specifying static types but they are not checked in the default implementation, CPython. An experimental optional static type checker named *mypy* supports compile-time type checking.[69]

Summary of Python 3's built-in types

Type	Mutable	Description	Syntax example

`str`	Immutable	A character string: sequence of Unicode codepoints	`'Wikipedia'`
			`"Wikipedia"`
			`"""Spanning multiple lines"""`

`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])`

`list`	Mutable	List, can contain mixed types	`[4.0, 'string', True]`
`tuple`	Immutable	Can contain mixed types	`(4.0, 'string', True)`
`set`	Mutable	Unordered set, contains no duplicates; can contain mixed types if hashable	`{4.0, 'string', True}`
`frozenset`	Immutable	Unordered set, contains no duplicates; can contain mixed types if hashable	`frozenset([4.0, 'string', True])`
`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}`
`int`	Immutable	Integer of unlimited magnitude[70]	`42`
`float`	Immutable	Floating point number, system-defined precision	`3.1415927`
`complex`	Immutable	Complex number with real and imaginary parts	`3+2.7j`
`bool`	Immutable	Boolean value	`True` `False`
`ellipsis`		An ellipsis placeholder to be used as an index in NumPy arrays	`...`

Mathematics

Python has the usual C arithmetic operators (`+`, `-`, `*`, `/`, `%`). It also has `**` for exponentiation, e.g. `5**3 == 125` and `9**0.5 == 3.0`, and a new matrix multiply `@` operator is included in version 3.5.[71] Additionally, it has a unary operator (`~`), which essentially inverts all the bytes of its one argument. For integers, this means `~x=-x-1`.[72] Other operators include bitwise shift operators `x << y`, which shifts `x` to the left `y` places, the same as `x*(2**y)`, and `x >> y`, which shifts `x` to the right `y` places, the same as `x/(2**y)`.[73]

The behavior of division has changed significantly over time:[74]

- Python 2.1 and earlier use the C division behavior. The `/` operator is integer division if both operands are integers, and floating-point division otherwise. Integer division rounds towards 0, e.g. `7 / 3 == 2` and `-7 / 3 == -2`.
- Python 2.2 changes integer division to round towards negative infinity, e.g. `7 / 3 == 2` and `-7 / 3 == -3`. The floor division `//` operator is introduced. So `7 // 3 == 2`, `-7 // 3 == -3`, `7.5 // 3 == 2.0`

and `-7.5 // 3 == -3.0`. Adding `from __future__ import division` causes a module to use Python 3.0 rules for division (see next).
- Python 3.0 changes `/` to be always floating-point division. In Python terms, the pre-3.0 `/` is *classic division*, the version-3.0 `/` is *real division*, and `//` is *floor division*.

Rounding towards negative infinity, though different from most languages, adds consistency. For instance, it means that the equation `(a+b) // b == a // b + 1` is always true. It also means that the equation `b * (a // b) + a % b == a` is valid for both positive and negative values of `a`. However, maintaining the validity of this equation means that while the result of `a % b` is, as expected, in the half-open interval [0, *b*), where `b` is a positive integer, it has to lie in the interval (*b*, 0] when `b` is negative.[75]

Python provides a `round` function for rounding a float to the nearest integer. For tie-breaking, versions before 3 use round-away-from-zero: `round(0.5)` is 1.0, `round(-0.5)` is −1.0.[76] Python 3 uses round to even: `round(1.5)` is 2, `round(2.5)` is 2.[77]

Python allows boolean expressions with multiple equality relations in a manner that is consistent with general use in mathematics. For example, the expression `a < b < c` tests whether `a` is less than `b` and `b` is less than `c`. 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`.[78]

Python has extensive built-in support for arbitrary precision arithmetic. Integers are transparently switched from the machine-supported maximum fixed-precision (usually 32 or 64 bits), belonging to the python type `int`, to arbitrary precision, belonging to the python type `long`, where needed. The latter have an "L" suffix in their textual representation.[79] (In Python 3, the distinction between the `int` and `long` types was eliminated; this behavior is now entirely contained by the `int` class.) The `Decimal` type/class in module `decimal` (since version 2.4) provides decimal floating point numbers to arbitrary precision and several rounding modes.[80] The `Fraction` type in module `fractions` (since version 2.6) provides arbitrary precision for rational numbers.[81]

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

Libraries

Python has a large standard library, commonly cited as one of Python's greatest strengths,[82] providing tools suited to many tasks. This is deliberate and has been described as a "batteries included"[25] Python philosophy. For Internet-facing applications, many standard formats and protocols (such as MIME and HTTP) are supported. Modules for creating graphical user interfaces, connecting to relational databases, generating pseudorandom numbers, arithmetic with arbitrary precision decimals,[83] manipulating regular expressions, and doing unit testing are also included.

Some parts of the standard library are covered by specifications (for example, the Web Server Gateway Interface (WSGI) implementation `wsgiref` follows PEP 333[84]), but most modules are not. They are specified by their code, internal documentation, and test suites (if supplied). However, because most of the standard library is cross-platform Python code, only a few modules need altering or rewriting for variant implementations.

As of May, 2017, the Python Package Index, the official repository containing third-party software for Python, contains over 107,000[85] packages offering a wide range of functionality, including:

- graphical user interfaces, web frameworks, multimedia, databases, networking and communications
- test frameworks, automation and web scraping, documentation tools, system administration

- scientific computing, text processing, image processing

Development environments

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

Other shells add abilities beyond those in the basic interpreter, including IDLE and IPython. While generally following the visual style of the Python shell, they implement features like auto-completion, session state retention, and syntax highlighting.

In addition to standard desktop integrated development environments (Python IDEs), there are also web browser-based IDEs, SageMath (intended for developing science and math-related Python programs), and a browser-based IDE and hosting environment, PythonAnywhere. Additionally, the Canopy IDE is also an option for writing Python programs.[86]

Implementations

Reference implementation

The main Python implementation, named CPython, is written in C meeting the C89 standard.[87] It compiles Python programs into intermediate bytecode,[88] which is executed by the virtual machine.[89] CPython is distributed with a large standard library written in a mixture of C and Python. It is available in versions for many platforms, including Windows and most modern Unix-like systems. CPython was intended from almost its very conception to be cross-platform.[90]

Other implementations

PyPy is a fast, compliant[91] interpreter of Python 2.7 and 3.5. Its just-in-time compiler brings a significant speed improvement over CPython.[92] A version taking advantage of multi-core processors using software transactional memory is being created.[93]

Stackless Python is a significant fork of CPython that implements microthreads; it does not use the C memory stack, thus allowing massively concurrent programs. PyPy also has a stackless version.[94]

MicroPython is a Python 3 variant that is optimised to run on microcontrollers.

No longer supported implementations

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

- Google began a project named Unladen Swallow in 2009 with the aim of speeding up the Python interpreter fivefold by using the LLVM, and of improving its multithreading ability to scale to thousands of cores.[95]
- Psyco is a just-in-time specialising compiler that integrates with CPython and transforms bytecode to machine code at runtime. The emitted code is specialised for certain data types and is faster than standard Python code.

In 2005, Nokia released a Python interpreter for the Series 60 mobile phones named PyS60. It includes many of the modules from the CPython implementations and some additional modules to integrate with the Symbian operating system. This project has been kept up to date to run on all variants of the S60 platform and there are

several third party modules available. The Nokia N900 also supports Python with GTK widget libraries, with the feature that programs can be both written and run on the target device.[96]

Cross-compilers to other langua ges

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

- Jython compiles into Java byte code, which can then be executed by every Java virtual machine implementation. This also enables the use of Java class library functions from the Python program.
- IronPython follows a similar approach in order to run Python programs on the .NET Common Language Runtime.
- The RPython language can be compiled to C, Java bytecode, or Common Intermediate Language, and is used to build the PyPy interpreter of Python.
- Pyjamas compiles Python to JavaScript.
- Cython compiles Python to C and C++.
- Pythran compiles Python to C++.
- Somewhat dated Pyrex (latest release in 2010) and Shed Skin (latest release in 2013) compile to C and C++ respectively.
- Google's Grumpy compiles Python to Go.

Performance

A performance comparison of various Python implementations on a non-numerical (combinatorial) workload was presented at EuroSciPy '13.[97]

Development

Python's development is conducted largely through the *Python Enhancement Proposal* (PEP) process. The PEP process is the primary mechanism for proposing major new features, for collecting community input on an issue, and for documenting the design decisions that have gone into Python.[98] Outstanding PEPs are reviewed and commented upon by the Python community and by Van Rossum, the Python project's benevolent dictator for life.[98]

Enhancement of the language goes along with development of the CPython reference implementation. The mailing list python-dev is the primary forum for discussion about the language's development; specific issues are discussed in the Roundup bug tracker maintained at python.org.[99] Development took place on a self-hosted source code repository running Mercurial, until Python moved to GitHub in January 2017.[100]

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

- Backwards-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—for example, version 3.0 was released 8 years after 2.0.
- Major or "feature" releases, which are largely compatible but introduce new features. The second part of the version number is incremented. These releases are scheduled to occur roughly every 18 months, and each major version is supported by bugfixes for several years after its release.[101]
- Bugfix releases, which introduce no new features but fix bugs. The third and final part of the version number is incremented. These releases are made whenever a sufficient number of bugs have been fixed upstream since the last release, or roughly every 3 months. Security vulnerabilities are also patched in bugfix releases.[102]

Many alpha, beta, and release-candidates are also released as previews and for testing before final releases. Although there is a rough schedule for each release, this is often pushed back if the code is not ready. The development team monitors the state of the code by running the large unit test suite during development, and using the BuildBot continuous integration system.[103]

The community of Python developers has also contributed over 86,000[104] software modules (as of 20 August 2016) to the Python Package Index (PyPI), the official repository of third-party libraries for Python.

The major academic conference on Python is named PyCon. There are special mentoring programmes like the Pyladies.

Naming

Python's name is derived from the television series *Monty Python's Flying Circus*,[105] and it is common to use Monty Python references in example code.[106] For example, the metasyntactic variables often used in Python literature are *spam* and *eggs*, instead of the traditional *foo* and *bar*.[106][107] Also, the official Python documentation and many code examples often contain various obscure Monty Python references.[108][109]

The prefix *Py-* is used to show that something is related to Python. Examples of the use of this prefix in names of Python applications or libraries include Pygame, a binding of SDL to Python (commonly used to create games); PyS60, an implementation for the Symbian S60 operating system; PyQt and PyGTK, which bind Qt and GTK to Python respectively; and PyPy, a Python implementation originally written in Python.

Uses

Since 2003, Python has consistently ranked in the top ten most popular programming languages as measured by the TIOBE Programming Community Index. As of March 2017, it is the fifth most popular language.[110] It was ranked as Programming Language of the Year for the year 2007 and 2010.[111] It is the third most popular language whose grammatical syntax is not predominantly based on C, e.g. C++, Objective-C (note, C# and Java only have partial syntactic similarity to C, such as the use of curly braces, and are closer in similarity to each other than C).

An empirical study found scripting languages (such as Python) more productive than conventional languages (such as C and Java) for a programming problem involving string manipulation and search in a dictionary. Memory consumption was often "better than Java and not much worse than C or C++".[112]

Large organizations that make use of Python include Wikipedia, Google,[113] Yahoo!,[114] CERN,[115] NASA,[116] and some smaller entities like ILM,[117] and ITA.[118] The social news networking site Reddit is written entirely in Python.

Python can serve as a scripting language for web applications, e.g., via mod_wsgi for the Apache web server.[119] With Web Server Gateway Interface, a standard API has evolved to facilitate these applications. Web frameworks like Django, Pylons, Pyramid, TurboGears, web2py, Tornado, Flask, Bottle and Zope support developers in the design and maintenance of complex applications. Pyjamas and IronPython can be used to develop the client-side of Ajax-based applications. SQLAlchemy can be used as data mapper to a relational database. Twisted is a framework to program communications between computers, and is used (for example) by Dropbox.

Libraries like NumPy, SciPy and Matplotlib allow the effective use of Python in scientific computing,[120][121] with specialized libraries such as Biopython and Astropy providing domain-specific functionality. SageMath is a mathematical software with a "notebook" programmable in Python: its library covers many aspects of

mathematics, including algebra, combinatorics, numerical mathematics, number theory, and calculus. The Python language re-implemented in Java platform is used for numeric and statistical calculations with 2D/3D visualization by the DMelt project.[122][123]

Python has been successfully embedded in many software products as a scripting language, including in finite element method software such as Abaqus, 3D parametric modeler like FreeCAD, 3D animation packages such as 3ds Max, Blender, Cinema 4D, Lightwave, Houdini, Maya, modo, MotionBuilder, Softimage, the visual effects compositor Nuke, 2D imaging programs like GIMP,[124] Inkscape, Scribus and Paint Shop Pro,[125] and musical notation programs like scorewriter and capella. GNU Debugger uses Python as a pretty printer to show complex structures such as C++ containers. Esri promotes Python as the best choice for writing scripts in ArcGIS.[126] It has also been used in several video games,[127][128] and has been adopted as first of the three available programming languages in Google App Engine, the other two being Java and Go.[129] Python is also used in algorithmic trading and quantitative finance.[130] Python can also be implemented in APIs of online brokerages that run on other languages by using wrappers.[131]

Python has been used in artificial intelligence tasks.[132][133][134][135] As a scripting language with module architecture, simple syntax and rich text processing tools, Python is often used for natural language processing tasks.[136]

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

Python has also seen extensive use in the information security industry, including in exploit development.[137][138]

Most of the Sugar software for the One Laptop per Child XO, now developed at Sugar Labs, is written in Python.[139]

The Raspberry Pi single-board computer project has adopted Python as its main user-programming language.

LibreOffice includes Python and intends to replace Java with Python. Python Scripting Provider is a core feature[140] since Version 4.0 from 7 February 2013.

Languages influenced by Python

Python's design and philosophy have influenced several programming languages, including:

- Boo uses indentation, a similar syntax, and a similar object model. However, Boo uses static typing (and optional duck typing) and is closely integrated with the .NET Framework.[141]
- Cobra uses indentation and a similar syntax. Cobra's "Acknowledgements" document lists Python first among languages that influenced it.[142] However, Cobra directly supports design-by-contract, unit tests, and optional static typing.[143]
- ECMAScript borrowed iterators, generators, and list comprehensions from Python.[144]
- Go is described as incorporating the "development speed of working in a dynamic language like Python".[145]
- Groovy was motivated by the desire to bring the Python design philosophy to Java.[146]
- Julia was designed "with true macros [.. and to be] as usable for general programming as Python [and] should be as fast as C".[19] Calling to or from Julia is possible; to with PyCall.jl and a Python package pyjulia allows calling, in the other direction, from Python.
- OCaml has an optional syntax, named twt (The Whitespace Thing), inspired by Python and Haskell.[147]

- Ruby's creator, Yukihiro Matsumoto, has said: "I wanted a scripting language that was more powerful than Perl, and more object-oriented than Python. That's why I decided to design my own language."[148]
- CoffeeScript is a programming language that cross-compiles to JavaScript; it has Python-inspired syntax.
- Swift is a programming language invented by Apple; it has some Python-inspired syntax.[149]

Python's development practices have also been emulated by other languages. The practice of requiring a document describing the rationale for, and issues surrounding, a change to the language (in Python's case, a PEP) is also used in Tcl[150] and Erlang[151] because of Python's influence.

Python has been awarded a TIOBE Programming Language of the Year award twice (in 2007 and 2010), which is given to the language with the greatest growth in popularity over the course of a year, as measured by the TIOBE index.[152]

See also

- Comparison of integrated development environments for Python
- Comparison of programming languages
- List of programming languages
- Off-side rule

References

1. "The History of Python: A Brief Timeline of Python" (http://python-history.blogspot.com/2009/01/brief-timeline-of-python.html). *Blogger*. 2009-01-20. Retrieved 2016-03-20.
2. Deily, Ned (2017-03-21). "Python 3.6.1 is now available" (http://blog.python.org/2017/03/python-361-is-now-available.html). *Python Insider*. The Python Core Developers. Retrieved 2017-03-22.
3. Peterson, Benjamin (2016-12-17). "Python 2.7.13 released" (http://blog.python.org/2016/12/python-2713-released.html). *Python Insider*. The Python Core Developers. Retrieved 2016-12-17.
4. File extension .pyo was removed in Python 3.5. See PEP 0488 (https://www.python.org/dev/peps/pep-0488/)
5. Holth, Moore (30 March 2014). "PEP 0441 -- Improving Python ZIP Application Support" (https://www.python.org/dev/peps/pep-0441/). Retrieved 12 Nov 2015.
6. "Why was Python created in the first place?" (https://docs.python.org/faq/general.html#why-was-python-created-in-the-first-place). *General Python FAQ*. Python Software Foundation. Retrieved 22 March 2007.
7. Kuchling, Andrew M. (22 December 2006). "Interview with Guido van Rossum (July 1998)" (http://www.amk.ca/python/writing/gvr-interview). *amk.ca*. Retrieved 12 March 2012.
8. van Rossum, Guido (1993). "An Introduction to Python for UNIX/C Programmers" (http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.38.2023). *Proceedings of the NLUUG najaarsconferentie (Dutch UNIX users group)*. "even though the design of C is far from ideal, its influence on Python is considerable."
9. "Classes" (https://docs.python.org/tutorial/classes.html) *The Python Tutorial*. Python Software Foundation. Retrieved 20 February 2012. "It is a mixture of the class mechanisms found in C++ and Modula-3
10. Simionato, Michele. "The Python 2.3 Method Resolution Order" (https://www.python.org/download/releases/2.3/mro/). Python Software Foundation. "The C3 method itself has nothing to do with Python, since it was invented by people working on Dylan and it is described in a paper intended for lispers."
11. Kuchling, A. M. "Functional Programming HOWTO" (https://docs.python.org/howto/functional.html). *Python v2.7.2 documentation*. Python Software Foundation. Retrieved 9 February 2012.
12. Schemenauer, Neil; Peters, Tim; Hetland, Magnus Lie (18 May 2001). "PEP 255 – Simple Generators" (https://www.python.org/dev/peps/pep-0255/). *Python Enhancement Proposals*. Python Software Foundation. Retrieved 9 February 2012.
13. Smith, Kevin D.; Jewett, Jim J.; Montanaro, Skip; Baxter, Anthony (2 September 2004). "PEP 318 – Decorators for Functions and Methods" (https://www.python.org/dev/peps/pep-0318/). *Python Enhancement Proposals*. Python Software Foundation. Retrieved 24 February 2012.
14. "More Control Flow Tools" (https://docs.python.org/3.2/tutorial/controlflow.html). *Python 3 documentation*. Python Software Foundation. Retrieved 24 July 2015.
15. "CoffeeScript borrows chained comparisons from Python" (http://coffeescript.org/).
16. "Genie Language - A brief guide" (https://wiki.gnome.org/action/show/Projects/Genie). Retrieved 2015-12-28.
17. "Perl and Python influences in JavaScript" (http://www.2ality.com/2013/02/javascript-influences.html). *www.2ality.com*. 24 February 2013. Retrieved 15 May 2015.

18. Rauschmayer, Axel. "Chapter 3: The Nature of JavaScript; Influences" (http://speakingjs.com/es5/ch03.html). *O'Reilly, Speaking JavaScript*. Retrieved 15 May 2015.
19. "Why We Created Julia" (http://julialang.org/blog/2012/02/why-we-created-julia). *Julia website*. February 2012. Retrieved 5 June 2014.
20. Bini, Ola (2007). *Practical JRuby on Rails Web 2.0 Projects: bringing Ruby on Rails to the Java platform*. Berkeley: APress. p. 3. ISBN 978-1-59059-881-8.
21. Lattner, Chris (3 June 2014). "Chris Lattner's Homepage" (http://nondot.org/sabre/). Chris Lattner. Retrieved 3 June 2014. "The Swift language is the product of tireless effort from a team of language experts, documentation gurus, compiler optimization ninjas, and an incredibly important internal dogfooding group who provided feedback to help refine and battle-test ideas. Of course, it also greatly benefited from the experiences hard-won by many other languag in the field, drawing ideas from Objective-C, Rust, Haskell, Ruby, Python, C#, CLU, and far too many others to list."
22. Summerfield, Mark. *Rapid GUI Programming with Python and Qt* "Python is a very expressive language, which means that we can usually write far fewer lines of Python code than would be required for an equivalent application written i say, C++ or Java"
23. McConnell, Steve (30 November 2009). *Code Complete, p. 100* (https://books.google.com/books?id=3JfE7TGUwvgC&pg=PT100). ISBN 9780735636972.
24. Kuhlman, Dave. "A Python Book: Beginning Python, Advanced Python, and Python Exercises" (http://cutter.rexx.com/~dkuhlman/python_book_01.html).
25. "About Python" (https://www.python.org/about). Python Software Foundation. Retrieved 24 April 2012., second section "Fans of Python use the phrase "batteries included" to describe the standard library, which covers everything from asynchronous processing to zip files."
26. "History and License" (https://docs.python.org/3/license.html). Retrieved 5 December 2016. "All Python releases are Open Source"
27. Venners, Bill (13 January 2003). "The Making of Python" (http://www.artima.com/intv/python.html). *Artima Developer*. Artima. Retrieved 22 March 2007.
28. van Rossum, Guido (20 January 2009). "A Brief Timeline of Python" (http://python-history.blogspot.com/2009/01/brief-timeline-of-python.html) *The History of Python*. Google. Retrieved 20 January 2009.
29. van Rossum, Guido (29 August 2000). "SETL (was: Lukewarm about range literals)" (http://mail.python.org/pipermail/python-dev/2000-August/008881.html). *Python-Dev* (Mailing list). Retrieved 13 March 2011.
30. van Rossum, Guido (1996). "Foreword for "Programming Python" (1st ed.)" (https://www.python.org/doc/essays/forword/). Retrieved 10 July 2014.
31. Kuchling, A. M.; Zadka, Moshe (16 October 2000). "What's New in Python 2.0" (https://docs.python.org/whatsnew/2.0.html). Python Software Foundation. Retrieved 11 February 2012.
32. "Python 3.0 Release" (https://www.python.org/download/releases/3.0/). Python Software Foundation. Retrieved 8 July 2009.
33. van Rossum, Guido (5 April 2006). "PEP 3000 – Python 3000" (https://www.python.org/dev/peps/pep-3000/). *Python Enhancement Proposals*. Python Software Foundation. Retrieved 27 June 2009.
34. "PEP 373 -- Python 2.7 Release Schedule" (http://legacy.python.org/dev/peps/pep-0373/). *python.org*. Retrieved 9 January 2017.
35. "PEP 466 -- Network Security Enhancements for Python 2.7.x" (https://www.python.org/dev/peps/pep-0466/). *python.org*. Retrieved 9 January 2017.
36. Claburn, Thomas (5 January 2017). "Google's Grumpy code makes Python Go" (http://www.theregister.co.uk/2017/01/05/googles_grumpy_makes_python_go/). Retrieved 9 January 2017.
37. "Google Open Source Blog: Grumpy: Go running Python!" (https://opensource.googleblog.com/2017/01/grumpy-go-running-python.html). 4 January 2017. Retrieved 7 March 2017.
38. The Cain Gang Ltd. "Python Metaclasses: Who? Why? When?" (http://www.webcitation.org/5lubkaJRc). Archived from the original (https://www.python.org/community/pycon/dc2004/papers/24/metaclasses-pycon.pdf) (PDF) on 10 December 2009. Retrieved 27 June 2009.
39. "3.3. Special method names" (https://docs.python.org/3.0/reference/datamodel.html#special-method-names). *The Python Language Reference*. Python Software Foundation. Retrieved 27 June 2009.
40. "PyDBC: method preconditions, method postconditions and class invariants for Python" (http://www.nongnu.org/pydbc/). Retrieved 24 September 2011.
41. "Contracts for Python" (http://www.wayforward.net/pycontract/). Retrieved 24 September 2011.
42. "PyDatalog" (https://sites.google.com/site/pydatalog/). Retrieved 22 July 2012.
43. Hettinger, Raymond (30 January 2002). "PEP 289 – Generator Expressions" (https://www.python.org/dev/peps/pep-0289/). *Python Enhancement Proposals*. Python Software Foundation. Retrieved 19 February 2012.
44. "6.5 itertools – Functions creating iterators for efficient looping" (https://docs.python.org/3/library/itertools.html). Docs.python.org. Retrieved 22 November 2016.
45. Peters, Tim (19 August 2004). "PEP 20 – The Zen of Python" (https://www.python.org/dev/peps/pep-0020/). *Python Enhancement Proposals*. Python Software Foundation. Retrieved 24 November 2008.

46. 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.
47. "Python Culture" (http://ebeab.com/2014/01/21/python-culture/)
48. "General Python FAQ - Why is it called Python?" (https://docs.python.org/2/faq/general.html#why-is-it-called-python)
49. "15 Ways Python Is a Powerful Force on the Web" (http://insidetech.monster.com/training/articles/8114-15-ways-python-is-a-powerful-force-on-the-web)
50. "pprint - Data pretty printer - Python Documentation" (https://docs.python.org/2/library/pprint.html)
51. Goodger, David. "Code Like a Pythonista: Idiomatic Python" (http://python.net/~goodger/projects/pycon/2007/idiomatic/handout.html).
52. "How to think like a Pythonista" (http://python.net/crew/mwh/hacks/objectthink.html)
53. "Is Python a good language for beginning programmers?" (https://docs.python.org/faq/general.html#is-python-a-good-language-for-beginning-programmers) *General Python FAQ*. Python Software Foundation. Retrieved 21 March 2007.
54. "Myths about indentation in Python" (http://www.secnetix.de/~olli/Python/block_indentation.hawk) Secnetix.de. Retrieved 19 April 2011.
55. Sweigart, Al (2010). "Appendix A: Differences Between Python 2 and 3". *Invent Your Own Computer Games with Python* (http://inventwithpython.com/appendixa.html) (2 ed.). ISBN 978-0-9821060-1-3. Retrieved 20 February 2014.
56. van Rossum, Guido (22 April 2009). "Tail Recursion Elimination" (http://neopythonic.blogspot.be/2009/04/tail-recursion-elimination.html). Neopythonic.blogspot.be. Retrieved 3 December 2012.
57. van Rossum, Guido (9 February 2006). "Language Design Is Not Just Solving Puzzles" (http://www.artima.com/weblogs/viewpost.jsp?thread=147358). *Artima forums*. Artima. Retrieved 21 March 2007.
58. van Rossum, Guido; Eby, Phillip J. (10 May 2005). "PEP 342 – Coroutines via Enhanced Generators" (https://www.python.org/dev/peps/pep-0342/) *Python Enhancement Proposals*. Python Software Foundation. Retrieved 19 February 2012.
59. "PEP 380" (https://www.python.org/dev/peps/pep-0380/) Python.org. Retrieved 3 December 2012.
60. "PEP 0465 -- A dedicated infix operator for matrix multiplication" (https://www.python.org/dev/peps/pep-0465/) *python.org*. Retrieved 1 January 2016.
61. "Python 3.5.1 Release and Changelog" (https://www.python.org/downloads/release/python-351/). *python.org*. Retrieved 1 January 2016.
62. "Chapter 15. Expressions - 15.21.1. Numerical Equality Operators == and !=" (http://docs.oracle.com/javase/specs/jls/se8/html/jls-15.html#jls-15.21.1). Oracle Corporation. Retrieved 28 August 2016.
63. "Chapter 15. Expressions - 15.21.3. Reference Equality Operators == and !=" (http://docs.oracle.com/javase/specs/jls/se8/html/jls-15.html#jls-15.21.3). Oracle Corporation. Retrieved 28 August 2016.
64. van Rossum, Guido; Hettinger, Raymond (7 February 2003). "PEP 308 – Conditional Expressions" (https://www.python.org/dev/peps/pep-0308/) *Python Enhancement Proposals*. Python Software Foundation. Retrieved 13 July 2011.
65. "PEP 498 -- Literal String Interpolation" (https://www.python.org/dev/peps/pep-0498/) *python.org*. Retrieved 8 March 2017.
66. "Why must 'self' be used explicitly in method definitions and calls?" (https://docs.python.org/faq/design.html#why-must-self-be-used-explicitly-in-method-definitions-and-calls) *Design and History FAQ*. Python Software Foundation. Retrieved 19 February 2012.
67. "The Python Language Reference, section 3.3. New-style and classic classes, for release 2.7.1" (https://docs.python.org/reference/datamodel.html#new-style-and-classic-classes) Retrieved 12 January 2011.
68. "Type hinting for Python" (https://lwn.net/Articles/627418/). LWN.net. 24 December 2014. Retrieved 5 May 2015.
69. "mypy - Optional Static Typing for Python" (http://mypy-lang.org/). Retrieved 28 January 2017.
70. Zadka, Moshe; van Rossum, Guido (1 March 2001). "PEP 237 – Unifying Long Integers and Integers" (https://www.python.org/dev/peps/pep-0237/) *Python Enhancement Proposals*. Python Software Foundation. Retrieved 24 September 2011.
71. "PEP 465 -- A dedicated infix operator for matrix multiplication" (http://legacy.python.org/dev/peps/pep-0465/) *python.org*.
72. "The tilde operator in Python - Stackoverflow" (https://stackoverflow.com/questions/8305199/the-tilde-operator-in-python). *stackoverflow.com*.
73. "BitwiseOperators - Python Wiki" (https://wiki.python.org/moin/BitwiseOperators). *wiki.python.org*.
74. Zadka, Moshe; van Rossum, Guido (1 March 2001). "PEP 238 – Changing the Division Operator" (https://www.python.org/dev/peps/pep-0238/) *Python Enhancement Proposals*. Python Software Foundation. Retrieved 23 October 2013.
75. "Why Python's Integer Division Floors" (http://python-history.blogspot.com/2010/08/why-pythons-integer-division-floors.html). Retrieved 25 August 2010.
76. "round" (https://docs.python.org/library/functions.html#round). *The Python standard library, release 2.7, §2: Built-in functions*, retrieved 14 August 2011
77. "round" (https://docs.python.org/py3k/library/functions.html#round). *The Python standard library, release 3.2, §2: Built-in functions*, retrieved 14 August 2011
78. Python Essential Reference, David M. Beazley
79. "Built-in Type" (https://docs.python.org/2.7/library/stdtypes.html). *docs.python.org*.

80. Batista, Facundo. "PEP 0327 -- Decimal Data Type" (https://www.python.org/dev/peps/pep-0327/) *Python.org*. Retrieved 2015-09-26.
81. "What's New in Python 2.6 — Python v2.6.9 documentation" (https://docs.python.org/2.6/whatsnew/2.6.html) *docs.python.org*. Retrieved 2015-09-26.
82. Piotrowski, Przemyslaw (July 2006). "Build a Rapid Web Development Environment for Python Server Pages and Oracle" (http://www.oracle.com/technetwork/articles/piotrowski-pythoncore-084049.html) *Oracle Technology Network*. Oracle. Retrieved 12 March 2012.
83. Batista, Facundo (17 October 2003). "PEP 327 – Decimal Data Type" (https://www.python.org/dev/peps/pep-0327/) *Python Enhancement Proposals*. Python Software Foundation. Retrieved 24 November 2008.
84. Eby, Phillip J. (7 December 2003). "PEP 333 – Python Web Server Gateway Interface v1.0" (https://www.python.org/dev/peps/pep-0333/) *Python Enhancement Proposals*. Python Software Foundation. Retrieved 19 February 2012.
85. Debill, Erik. "Module Counts" (http://www.modulecounts.com/) *ModuleCounts*. Retrieved 10 May 2017.
86. Enthought, Canopy. "Canopy" (https://www.enthought.com/products/canopy/) *www.enthought.com*. Retrieved 20 August 2016.
87. van Rossum, Guido (5 June 2001). "PEP 7 – Style Guide for C Code" (https://www.python.org/dev/peps/pep-0007/) *Python Enhancement Proposals*. Python Software Foundation. Retrieved 24 November 2008.
88. "CPython byte code" (https://docs.python.org/3/library/dis.html#python-bytecode-instructions) Docs.python.org. Retrieved 16 February 2016.
89. "Python 2.5 internals" (http://www.troeger.eu/teaching/pythonvm08.pdf) (PDF). Retrieved 19 April 2011.
90. "An Interview with Guido van Rossum" (http://www.oreilly.com/pub/a/oreilly/frank/rossum_1099.html) Oreilly.com. Retrieved 24 November 2008.
91. "PyPy compatibility" (http://pypy.org/compat.html). Pypy.org. Retrieved 3 December 2012.
92. "speed comparison between CPython and Pypy" (http://speed.pypy.org/). Speed.pypy.org. Retrieved 3 December 2012.
93. "STM with threads" (http://morepypy.blogspot.be/2012/06/stm-with-threads.html) Morepypy.blogspot.be. 10 June 2012. Retrieved 3 December 2012.
94. "Application-level Stackless features — PyPy 2.0.2 documentation" (http://doc.pypy.org/en/latest/stackless.html) Doc.pypy.org. Retrieved 17 July 2013.
95. "Plans for optimizing Python" (https://code.google.com/p/unladen-swallow/wiki/ProjectPlan) *Google Project Hosting*. Google. 15 December 2009. Retrieved 24 September 2011.
96. "Python on the Nokia N900" (http://www.stochasticgeometry.ie/2010/04/29/python-on-the-nokia-n900/) *Stochastic Geometry*.
97. 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).
98. Warsaw, Barry; Hylton, Jeremy; Goodger, David (13 June 2000). "PEP 1 – PEP Purpose and Guidelines" (https://www.python.org/dev/peps/pep-0001/) *Python Enhancement Proposals*. Python Software Foundation. Retrieved 19 April 2011.
99. 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.
100. "Python Developer's Guide" (https://docs.python.org/devguide/).
101. Norwitz, Neal (8 April 2002). "[Python-Dev] Release Schedules (was Stability & change)" (https://mail.python.org/pipermail/python-dev/2002-April/022739.html) Retrieved 27 June 2009.
102. Aahz; Baxter, Anthony (15 March 2001). "PEP 6 – Bug Fix Releases" (https://www.python.org/dev/peps/pep-0006/) *Python Enhancement Proposals*. Python Software Foundation. Retrieved 27 June 2009.
103. "Python Buildbot" (https://www.python.org/dev/buildbot/). *Python Developer's Guide*. Python Software Foundation. Retrieved 24 September 2011.
104. DeBill, Erik. "Module Counts" (http://www.modulecounts.com/#) *www.modulecounts.com*. Retrieved 20 August 2016.
105. "General Python FAQ" (https://docs.python.org/2/faq/general.html#why-is-it-called-python) *Python v2.7.3 documentation*. Docs.python.org. Retrieved 3 December 2012.
106. "Whetting Your Appetite" (https://docs.python.org/tutorial/appetite.html) *The Python Tutorial*. Python Software Foundation. Retrieved 20 February 2012.
107. "In Python, should I use else after a return in an if block?" (https://stackoverflow.com/questions/5033906/in-python-should-i-use-else-after-a-return-in-an-if-block) *Stack Overflow*. Stack Exchange. 17 February 2011. Retrieved 6 May 2011.
108. Lutz, Mark (2009). *Learning Python: Powerful Object-Oriented Programming* (https://books.google.com/books?id=1HxWGezDZcgC&pg=PA17). O'Reilly Media, Inc. p. 17. ISBN 9781449379322.
109. Fehily, Chris (2002). *Python* (https://books.google.com/books?id=carqdIdfVlYC&pg=PR15) Peachpit Press. p. xv. ISBN 9780201748840.
110. "TIOBE Index" (http://www.tiobe.com/tiobe-index/) TIOBE - The Software Quality Company. Retrieved 7 March 2017.
111. TIOBE Software Index (2015). "TIOBE Programming Community Index Python" (http://www.tiobe.com/index.php/paperinfo/tpci/Python.html) Retrieved 10 September 2015.

112. Prechelt, Lutz (14 March 2000). "An empirical comparison of C, C++, Java, Perl, Python, Rexx, and Tcl" (http://page.mi.fu-berlin.de/prechelt/Biblio/jccpprt_computer2000.pdf) (PDF). Retrieved 30 August 2013.
113. "Quotes about Python" (https://www.python.org/about/quotes/). Python Software Foundation. Retrieved 8 January 2012.
114. "Organizations Using Python" (https://wiki.python.org/moin/OrganizationsUsingPython). Python Software Foundation. Retrieved 15 January 2009.
115. "Python : the holy grail of programming" (http://cdsweb.cern.ch/journal/CERNBulletin/2006/31/News%20Articles/974227?ln=en). *CERN Bulletin*. CERN Publications (31/2006). 31 July 2006. Retrieved 11 February 2012.
116. Shafer, Daniel G. (17 January 2003). "Python Streamlines Space Shuttle Mission Design" (https://www.python.org/about/success/usa/). Python Software Foundation. Retrieved 24 November 2008.
117. Fortenberry, Tim (17 January 2003). "Industrial Light & Magic Runs on Python" (https://www.python.org/about/success/ilm/). Python Software Foundation. Retrieved 11 February 2012.
118. Taft, Darryl K. (5 March 2007). "Python Slithers into Systems" (http://www.eweek.com/c/a/Application-Development/Python-Slithers-into-Systems/). *eWeek.com*. Ziff Davis Holdings. Retrieved 24 September 2011.
119. "Usage statistics and market share of Python for websites" (http://w3techs.com/technologies/details/pl-python/all/all) 2012. Retrieved 18 December 2012.
120. Oliphant, Travis (2007). "Python for Scientific Computing" (https://www.h2desk.com/blog/python-scientific-computing/). *Computing in Science and Engineering*
121. Millman, K. Jarrod; Aivazis, Michael (201). "Python for Scientists and Engineers" (http://www.computer.org/csdl/mags/cs/2011/02/mcs2011020009.html). *Computing in Science and Engineering* **13** (2): 9–12.
122. Chekanov, S. (April 2016). *Numeric Computation and Statistical Data Analysis on the Java Platform* (http://www.springer.com/la/book/9783319285290). London: Springer. p. 670. ISBN 978-3-319-28531-3.
123. Chekanov, S. (2010). *Scientific Data Analysis using Jython Scripting and Java* (http://www.springer.com/la/book/9783319285290). London: Springer. p. 600. ISBN 978-3-319-28531-3.
124. "Installers for GIMP for Windows - Frequently Asked Questions" (http://gimp-win.sourceforge.net/faq.html). 26 July 2013. Retrieved 26 July 2013.
125. "jasc psp9components" (https://web.archive.org/web/20080319061519/http://www.jasc.com/support/customercare/articles/psp9components.asp) Archived from the original (http://www.jasc.com/support/customercare/articles/psp9components.asp) on 19 March 2008.
126. "About getting started with writing geoprocessing scripts" (http://webhelp.esri.com/arcgisdesktop/9.2/index.cfm?TopicName=About_getting_started_with_writing_geoprocessing_scripts). *ArcGIS Desktop Help 9.2* Environmental Systems Research Institute. 17 November 2006. Retrieved 11 February 2012.
127. CCP porkbelly (24 August 2010). "Stackless Python 2.7" (http://community.eveonline.com/news/dev-blogs/stackless-python-2.7/). *EVE Community Dev Blogs*. CCP Games. "As you may know, EVE has at its core the programming languag known as Stackless Python."
128. Caudill, Barry (20 September 2005). "Modding Sid Meier's Civilization IV" (http://www.webcitation.org/5ru5VItfv). *Sid Meier's Civilization IV Developer Blog*. Firaxis Games. Archived from the original (http://www.2kgames.com/civ4/blog_03.htm) on 10 August 2010. "we created three levels of tools ... The next level offers Python and XML support, letting modders with more experience manipulate the game world and everything in it.
129. "Python Language Guide (v1.0)" (http://www.webcitation.org/5ru5FHxfV). *Google Documents List Data API v1.0*. Google. Archived from the original (https://code.google.com/apis/documents/docs/1.0/developers_guide_python.html) on 10 August 2010.
130. "Python - Best Programming Language for Algorithmic Trading Systems" (http://www.quantinsti.com/blog/python-best-programming-language-algorithmic-trading/). 2016-03-09. Retrieved 2016-10-03.
131. "Trading with Interactive Brokers using Python: An IBPy Tutorial" (http://www.quantinsti.com/blog/ibpy-tutorial-implement-python-interactive-brokers-api/). 2016-09-19. Retrieved 2016-10-03.
132. "Python for Artificial Intelligence" (https://web.archive.org/web/20121101045354/http://wiki.python.org/moin/PythonForArtificialIntelligence). Wiki.python.org. 19 July 2012. Archived from the original (https://wiki.python.org/moin/PythonForArtificialIntelligence) on 1 November 2012. Retrieved 3 December 2012.
133. Paine, Jocelyn, ed. (August 2005). "AI in Python" (http://www.ainewsletter.com/newsletters/aix_0508.htm#python_ai_ai). *AI Expert Newsletter*. Amzi!. Retrieved 11 February 2012.
134. "PyAIML 0.8.5 : Python Package Index" (https://pypi.python.org/pypi/PyAIML). Pypi.python.org. Retrieved 17 July 2013.
135. Russell, Stuart J. & Norvig, Peter (2009). *Artificial Intelligence: A Modern Approach* (http://aima.cs.berkeley.edu/) (3rd ed.). Upper Saddle River, NJ: Prentice Hall. p. 1062. ISBN 978-0-13-604259-4. Retrieved 11 February 2012.
136. "Natural Language Toolkit" (http://www.nltk.org).
137. "Immunity: Knowing You're Secure" (http://www.immunitysec.com/products-immdbg.shtml).
138. "Corelabs site" (http://oss.coresecurity.com/).
139. "What is Sugar?" (http://sugarlabs.org/go/Sugar). Sugar Labs. Retrieved 11 February 2012.
140. "4.0 New Features and Fixes" (http://www.libreoffice.org/download/4-0-new-features-and-fixes/). *LibreOffice.org*. The Document Foundation 2013. Retrieved 25 February 2013.

141. "Gotchas for Python Users"(http://boo.codehaus.org/Gotchas+for+Python+Users). *boo.codehaus.org*. Codehaus Foundation. Retrieved 24 November 2008.
142. Esterbrook, Charles."Acknowledgements"(http://cobra-language.com/docs/acknowledgements/). *cobra-language.com* Cobra Language. Retrieved 7 April 2010.
143. Esterbrook, Charles."Comparison to Python"(http://cobra-language.com/docs/python/). *cobra-language.com* Cobra Language. Retrieved 7 April 2010.
144. "Proposals: iterators and generators [ES4 Wiki]" (http://wiki.ecmascript.org/doku.php?id=proposals:iterators_and_gene rators). wiki.ecmascript.org. Retrieved 24 November 2008.
145. Kincaid, Jason (10 November 2009)."Google's Go: A New Programming Language That's Python Meets C++"(http:// www.techcrunch.com/2009/11/10/google-go-language/). TechCrunch. Retrieved 29 January 2010.
146. Strachan, James (29 August 2003)."Groovy – the birth of a new dynamic language for the Java platform"(http://radio.w eblogs.com/0112098/2003/08/29.html)
147. Lin, Mike. "The Whitespace Thing for OCaml"(http://people.csail.mit.edu/mikelin/ocaml+twt/)Massachusetts Institute of Technology. Retrieved 12 April 2009.
148. "An Interview with the Creator of Ruby"(http://www.linuxdevcenter.com/pub/a/linux/2001/11/29/ruby.html). Linuxdevcenter.com. Retrieved 3 December 2012.
149. Lattner, Chris (3 June 2014). "Chris Lattner's Homepage"(http://nondot.org/sabre). Chris Lattner. Retrieved 3 June 2014. "I started work on the Swift Programming Language in July of 2010. I implemented much of the basic language structure, with only a few people knowing of its existence. A few other (amazing) people started contributing in earne late in 2011, and it became a major focus for the Apple Developer Tools group in July 2013[...] drawing ideas from Objective-C, Rust, Haskell, Ruby, Python, C#, CLU, and far too many others to list".
150. Kupries, Andreas; Fellows, Donal K. (14 September 2000)."TIP #3: TIP Format"(http://www.tcl.tk/cgi-bin/tct/tip/3.ht ml). *tcl.tk*. Tcl Developer Xchange. Retrieved 24 November 2008.
151. Gustafsson, Per; Niskanen, Raimo (29 January 2007)."EEP 1: EEP Purpose and Guidelines"(http://www.erlang.org/eep s/eep-0001.html). erlang.org. Retrieved 19 April 2011.
152. "TIOBE Programming Community Index for March 2012"(http://www.tiobe.com/index.php/content/paperinfo/tpci/). TIOBE Software. March 2012. Retrieved 25 March 2012.

Further reading

- Downey, Allen B (May 2012). *Think Python: How to Think Like a Computer Scientist* (Version 1.6.6 ed.). ISBN 978-0-521-72596-5.
- Hamilton, Naomi (5 August 2008). "The A-Z of Programming Languages: Python". *Computerworld*. Retrieved 31 March 2010.
- Lutz, Mark (2013). *Learning Python* (5th ed.). O'Reilly Media. ISBN 978-0-596-15806-4.
- Pilgrim, Mark (2004). *Dive Into Python*. Apress. ISBN 978-1-59059-356-1.
- Pilgrim, Mark (2009). *Dive Into Python 3*. Apress. ISBN 978-1-4302-2415-0.
- Summerfield, Mark (2009). *Programming in Python 3* (2nd ed.). Addison-Wesley Professional. ISBN 978-0-321-68056-3.

External links

- Official website
- Python (programming language) newsgroup on Usenet (alternative free web access using Google Groups)
- The History of Python (blog by Guido van Rossum)
- Python development list
- Python at DMOZ

Categories: Class-based programming languages | Computational notebook | Computer science in the Netherlands | Cross-platform free software | Dutch inventions | Dynamically typed programming languages | Educational programming languages | High-level programming languages | Information technology in the Netherlands

Object-oriented programming languages | Programming languages created in 1991 | Python (programming language) | Scripting languages | Text-oriented programming languages

---