Python (programming language)

From Wikipedia, the free encyclopedia

Python is a general-purpose, interpreted high-level programming language[12] whose design philosophy emphasizes code readability. Its syntax is said to be clear[13][14] and expressive.[15] Python has a large and comprehensive standard library.[16]

Python supports multiple programming paradigms, primarily but not limited to object-oriented, imperative and, to a lesser extent, functional programming styles. It features a fully dynamic type system and automatic memory management, similar to that of Scheme, Ruby, Perl, and Tcl. Like other dynamic languages, Python is often used as a scripting language, but is also used in a wide range of non-scripting contexts. Using third-party tools, Python code can be packaged into standalone executable programs. Python interpreters are available for many operating systems.

CPython, the reference implementation of Python, is free and open source software and has a community-based development model, as do nearly all of its alternative implementations. CPython is managed by the non-profit Python Software Foundation.

Contents

Python

Paradigm(s)	multi-paradigm: object-oriented, imperative, functional, procedural, reflective
Appeared in	1991
Designed by	Guido van Rossum
Developer	Python Software Foundation
Stable release	3.3.0 / 29 September 2012 2.7.3 / 11 April 2012
Preview release	3.3.0rc3 / 24 September 2012[1]
Typing discipline	duck, dynamic, strong
Major implementations	**CPython**, IronPython, Jython, Python for S60, PyPy
Dialects	Cython, RPython, Stackless Python
Influenced by	ABC,[2] ALGOL 68,[3] C,[4] C++,[5] Dylan,[6] Haskell,[7] Icon,[8] Java,[9] Lisp,[10] Modula-3,[5] Perl
Influenced	Boo, Cobra, D, Falcon, Groovy, JavaScript, F#, Ruby[11]
OS	Cross-platform
License	Python Software Foundation License
Usual filename extensions	.py, .pyw, .pyc, .pyo, .pyd

Website	python.org (http://www.python.org/)

Python Programming at Wikibooks

History

Main article: History of Python

Guido van Rossum, the creator of Python

Python was conceived in the late 1980s[17] and its implementation was started in December 1989[18] by Guido van Rossum at CWI in the Netherlands as a successor to the ABC language (itself inspired by SETL)[19] capable of exception handling and interfacing with the Amoeba operating system.[2] 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).

Python 2.0 was released on 16 October 2000, with many major new features including a full garbage collector and support for Unicode. However, the most important change was to the development process itself, with a shift to a more transparent and community-backed process.[20] Python 3.0 (also called Python 3000 or py3k), a major, backwards-incompatible release, was released on 3 December 2008[21] after a long period of testing. Many of its major features have been backported to the backwards-compatible Python 2.6 and 2.7.[22] Python has been awarded a TIOBE Programming Language of the Year award twice (2007, 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.[23]

Programming philosophy

Python is a multi-paradigm programming language. Rather than forcing programmers to adopt a particular style of programming, it permits several styles: object-oriented programming and structured programming are fully supported, and there are a number of language features which support functional programming and aspect-oriented programming (including by metaprogramming[24] and by magic methods).[25] Many other paradigms are supported using extensions, including design by contract[26][27] and logic programming.[28]

Python uses dynamic typing and a combination 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.

Rather than requiring all desired functionality to be built into the language's core, Python was designed to be highly extensible. New built-in modules can be easily written in C, C++ or Cython. Python can also be used as an extension language for existing modules and 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 very start because of his frustrations with ABC (which espoused the opposite mindset).[17]

The design of Python offers only limited support for functional programming in the Lisp tradition. However, Python's design philosophy exhibits significant similarities to those of minimalistic Lisp-family languages, such as Scheme.[*citation needed*] The language has map(), reduce() and filter() functions, and the list comprehensions added in Python 2.0 have since been extended with comprehensions for dictionaries and sets, as well as generator expressions. The standard library has two modules (itertools and functools) that implement functional tools borrowed from Haskell and Standard ML.[29]

While offering choice in coding methodology, the Python philosophy rejects exuberant syntax, such as in Perl, in favor of a sparser, less-cluttered grammar. Python's developers expressly promote a particular "culture" or ideology based on what they want the language to be, favoring language forms they see as "beautiful", "explicit" and "simple". As Alex Martelli put it in his *Python Cookbook* (2nd ed., p. 230): "To describe something as clever is NOT considered a compliment in the Python culture." 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".[30]

Python's developers strive to avoid premature optimization, and moreover, reject patches to non-critical parts of CPython which would offer a marginal increase in speed at the cost of clarity.[31] When speed is important, Python programmers tend to try using a JIT compiler such as Psyco or using an alternative language implementation such as PyPy. When pure Python code is not fast enough, time-critical functions can be rewritten in "closer to the metal" languages such as C, or by translating (a dialect of) Python code to C code using tools like Cython.[32]

The core philosophy of the language is summarized by the document "PEP 20 (The Zen of Python)".[30]

Name and neologisms

An important goal of the Python developers is making Python fun to use. This is reflected in the origin of the name (derived from the television series *Monty Python's Flying Circus*), in the common practice of using Monty Python references in example code, and in an occasionally playful approach to tutorials and reference materials.[33][34] For example, the metasyntactic variables often used in Python literature are *spam* and *eggs*,[33][35] instead of the traditional *foo* and *bar*.

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. Likewise, to say of an interface or language feature that it is pythonic is to say that it works well with Python idioms, that its use meshes well with the rest of the language.

In contrast, a mark of *unpythonic* code is that it attempts to write C++ (or Lisp, Perl, or Java) code in Python —that is, provides a rough transcription rather than an idiomatic translation of forms from another language. The concept of pythonicity is tightly bound to Python's minimalist philosophy of readability and avoiding the

"there's more than one way to do it" approach. Unreadable code or incomprehensible idioms are unpythonic.

Users and admirers of Python—most especially those considered knowledgeable or experienced—are often referred to as *Pythonists*, *Pythonistas*, and *Pythoneers*.[36]

The prefix *Py* can be 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, respectively, to Python; and PyPy, a Python implementation written in Python. The prefix is also used outside of naming software packages: the major Python conference is named PyCon.

Usage

Main article: List of Python software

Python is often used as a scripting language for web applications, e.g. via mod_wsgi for the Apache web server. With Web Server Gateway Interface, a standard API has been developed to facilitate these applications. Web application frameworks like Django, Pylons, Pyramid, TurboGears, web2py, Tornado, Flask and Zope support developers in the design and maintenance of complex applications. Libraries like NumPy, SciPy and Matplotlib allow Python to be used effectively in scientific computing.

Python has been successfully embedded in a number of software products as a scripting language, including in finite element method software such as Abaqus, 3D animation packages such as Blender, Cinema 4D, Lightwave, Houdini, Maya, modo, MotionBuilder, Softimage, the visual effects compositor Nuke, and 2D imaging programs like GIMP, Inkscape, Scribus and Paint Shop Pro.[37] GNU GDB uses Python as a pretty printer to show complex structures such as C++ containers. ESRI is now promoting Python as the best choice for writing scripts in ArcGIS.[38] It has even been used in several video games,[39][40] and has been adopted as first of the three available programming languages in Google App Engine, the other two being Java and Go.[41]

Thanks to being a scripting language with module architecture, syntax simplicity and rich text processing tools, Python is often used for Natural language processing tasks.[42] Python has also been used in Artificial Intelligence tasks.[43][44][45]

For many operating systems, Python is a standard component; it ships with most Linux distributions, FreeBSD, NetBSD, OpenBSD and with OS X and can be used from the terminal. A number of 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 and the standard tool to access it, emerge. Pardus uses it for administration and during system boot.[46]

Python has also seen extensive use in the information security industry, including exploit development.[47][48]

Among the users of Python are YouTube[49], the original BitTorrent client,[50] and Spotify.[51] Large organizations that make use of Python include Google,[49] Yahoo!,[52] CERN,[53] NASA,[54] ILM,[55] and ITA.[56] Most of the Sugar software for the One Laptop per Child XO, now developed at Sugar Labs, is written in Python.[57]

In addition to standard desktop Python IDEs, there are also browser-based IDEs, such as NCLab and Sage, intended for developing science and math-related Python programs.

As of April 2012, Python ranks at position 8 in the TIOBE Programming Community Index.[58]

Syntax and semantics

Main article: Python syntax and semantics

Python is intended to be a highly readable language. It is designed to have an uncluttered visual layout, frequently using English keywords where other languages use punctuation. Python requires less boilerplate than traditional manifestly typed structured languages such as C or Pascal, and has a smaller number of syntactic exceptions and special cases than either of these.[59] For a detailed description of the differences between 2.x and 3.x versions, see History of Python.

```
def add5(x):
    return x+5

def dotwrite(ast):
    nodename = getNodename()
    label=symbol.sym_name.get(int(ast[0]),ast[0])
    print '    %s [label="%s' % (nodename,label),
    if isinstance(ast[1], str):
        if ast[1].strip():
            print '= %s"];' % ast[1]
        else:
            print '"]'
    else:
        print '"];'
        children = []
        for n, child in enumerate(ast[1:]):
            children.append(dotwrite(child))
        print '    %s -> {' % nodename,
        for name in children:
            print '%s' % name,
```

Syntax-highlighted Python 2.x code.

The simplicity of Python is demonstrated by its version of the classic "Hello world" program:

```
print("Hello world")
```

Indentation

Python uses whitespace indentation, rather than curly braces or keywords, to delimit blocks; a feature also termed the off-side rule. An increase in indentation comes after certain statements; a decrease in indentation signifies the end of the current block.[60]

Statements and control flow

Python's statements include (among others):

- 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).
- 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 – see below.)
- The `import` statement, which is used to import modules whose functions or variables can be used in the current program.

Each statement has its own semantics: for example, the `def` statement does not execute its block immediately, unlike most other statements.

Python does not support first-class continuations, and according to Guido van Rossum it never will.[61] However, better support for coroutine-like functionality is provided in 2.5, by extending Python's generators.[62] Prior to 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.

Expressions

Python expressions are similar to languages such as C and Java.

- In Python 2, the `/` operator on integers does integer division; it truncates the result to an integer. Floating-point division on integers can be achieved by converting one of the integers to a float (e.g. `float(x) / y`). In Python 3, the result of `/` is always a floating-point value. This behaviour can be enabled in Python 2.2+ using `from __future__ import division`. In both Python 2.2+ and Python 3, `//` can be used to do integer division.
- In Python, `==` compares by value, in contrast to Java, where it compares by reference. (Value comparisons in Java use the `equals()` method.) Python's `is` operator may be used to compare object identities (comparison by reference). 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 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*.[63]
- Anonymous functions are implemented using lambda expressions; however, these are limited in that the body can only be a single expression.
- Conditional expressions in Python are written as `x if c else y`[64] (different in order of operands from the `?:` 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 2 has a "string format" operator `%`. This functions analogous to `printf` format strings in C, e.g. `"foo=%s bar=%d" % ("blah", 2)` evaluates to `"foo=blah bar=2"`. In Python 3, this was

supplemented by the `format()` method of the `str` class, e.g. `"foo={0} bar={1}".format("blah", 2)`.

- Python has various kinds of string literals:
  - Strings delimited by single or double quotation marks. Unlike in Unix shells, Perl and Perl-influenced languages, single quotation marks and double quotation marks function similarly. Both kinds of string use the backslash (`\`) as an escape character and there is no implicit string interpolation such as `"$foo"`.
  - Triple-quoted strings, which begin and end with a series of three single or double quotation 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`. No escape sequences are 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 index and slice 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 some duplication of functionality, e.g.

- 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 and 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 valid 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 in some other object-oriented programming languages (for example, Java, C++ or Ruby).[65]

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 themselves are instances of the metaclass `type` (itself an instance of itself), allowing metaprogramming and reflection.

Prior to version 3.0, Python had two kinds of classes: "old-style" and "new-style".[66] Old-style classes were eliminated in Python 3.0, making all classes new-style. In versions between 2.2 and 3.0, both kinds of classes could be used. 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`).

Here is a summary of Python 3's built-in types:

Type	Description	Syntax example

`str`	An immutable sequence of Unicode codepoints.	`'Wikipedia'`
		`"Wikipedia"`
		`"""Spanning multiple lines"""`

`bytearray`	A mutable sequence of bytes.	`bytearray(b'Some ASCII')` `bytearray(b"Some ASCII")`
`bytes`	An immutable sequence of bytes.	`b'Some ASCII'` `b"Some ASCII"`
`list`	Mutable, can contain mixed types.	`[4.0, 'string', True]`
`tuple`	Immutable, can contain mixed types.	`(4.0, 'string', True)`
`set`, `frozenset`	Unordered, contains no duplicates. A `frozenset` is immutable.	`set([4.0, 'string', True])` `frozenset([4.0, 'string', True])`
`dict`	A mutable group of key and value pairs.	`{'key1': 1.0, 3: False}`
`int`	An immutable integer of unlimited magnitude.[67]	`42`
`float`	An immutable floating point number (system-defined precision).	`3.1415927`
`complex`	An immutable complex number with real and imaginary parts.	`3+2.7j`
`bool`	An immutable truth value.	`True` `False`

Mathematics

In contrast with some programming languages, integer division is defined to round towards minus infinity. Therefore `7 // 3` is 2, but `(-7) // 3` is −3. This is uniform and consistent: for instance, it means that the equation `(a+b) // b == a // b + 1` is always true, whereas in languages such as C, `(-6+7) / 7 == -6`

`/ 7`. 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.[68]

Python provides a `round` function for rounding floats to integers. Versions before 3 use round-away-from-zero: `round(0.5)` is 1.0, `round(-0.5)` is −1.0.[69] Python 3 uses round-to-even: `round(1.5)` is 2.0, `round(2.5)` is 2.0.[70] The `Decimal` type/class in module `decimal` (since version 2.4) provides exact numerical representation and several rounding modes.

Python allows boolean expressions with multiple equality relations in a manner that is 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`. 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`.[71]

Implementations

See also: List of Python software#Python implementations

CPython

Main article: CPython

The mainstream Python implementation, named *CPython*, is written in C meeting the C89 standard.[72] CPython compiles Python programs into intermediate bytecode,[73] which are then executed by the virtual machine.[74] It is distributed with a large standard library written in a mixture of C and Python. CPython ships in versions for many platforms, including Microsoft Windows and most modern Unix-like systems. CPython was intended from almost its very conception to be cross-platform; its use and development on esoteric platforms such as Amoeba, alongside more conventional ones like Unix and Mac OS, has greatly helped in this regard.[75]

Stackless Python is a significant fork of CPython that implements microthreads; it does not use the C memory stack. It can be expected to run on approximately the same platforms that CPython runs on.

Google started a project called Unladen Swallow in 2009 with the aims of increasing the speed of the Python interpreter by 5 times by using the LLVM and improving its multithreading ability to scale to thousands of cores.[76] Later the project lost Google's backing and its main developers. As of 1 February 2012, the project hasn't achieved its goal; the modified interpreter is only about 2 times faster.[*citation needed*]

Alternative implementations

Jython compiles the Python program 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. PyPy is a fast self-hosting implementation of Python, written in Python, that can output several

types of bytecode, object code and intermediate languages. There also exist 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. PyPy is of this type, compiling RPython to several languages; other examples include Pyjamas compiling to JavaScript; Shed Skin compiling to C++; and Cython and Pyrex compiling to C.

In 2005 Nokia released a Python interpreter for the Series 60 mobile phones called PyS60. It includes many of the modules from the CPython implementations and some additional modules for integration 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 device itself. There is also a Python interpreter for Windows CE devices (including Pocket PC). It is called PythonCE.[77] There are additional tools available for easy application and GUI development.

The PyMite virtual machine began in 2000 and made its first public appearance at PyCon 2003.[78] PyMite was folded into Python-on-a-Chip (http://pythononachip.org/) in 2009.[79] Python-on-a-Chip (p14p) is a project to develop a reduced Python virtual machine (codenamed PyMite) that runs a significant subset of the Python language on microcontrollers without an OS in as little as 4KB of RAM.[80]

Around 2004,[*citation needed*] the Pyastra (http://pyastra.sourceforge.net/) project created a specialized translator and assembler that targets resource-constrained microcontrollers.

ChinesePython (中蟒) is a Python programming language using a Chinese-language lexicon. Besides reserved words and variable names, most data type operations can be coded in Chinese as well.[*citation needed*]

Python is available on Android as an option as part of the Android Scripting Environment.[81] or via the Python-for-android project, which produce native apk for android.[82]

Python is available on iOS through the Kivy-ios project, allowing to build cross-platform OpenGL ES 2.0 Python applications.[83]

Interpretational semantics

Most Python implementations (including CPython) can function as a command line interpreter, for which the user enters statements sequentially and receives the results immediately. In short, Python acts as a shell. While the semantics of the other modes of execution (bytecode compilation, or compilation to native code) preserve the sequential semantics, they offer a speed boost at the cost of interactivity, so they are usually only used outside of a command-line interaction (e.g., when importing a module).

Other shells add capabilities 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, retention of session state, and syntax highlighting.

Some implementations can compile not only to bytecode, but can turn Python code into machine code. So far, this has only been done for restricted subsets of Python. PyPy takes this approach, naming its restricted compilable version of Python *RPython*.

Psyco is a specialising just in time compiler that integrates with CPython and transforms bytecode to machine

code at runtime. The produced code is specialised for certain data types and is faster than standard Python code. Psyco is compatible with all Python code, not only a subset.[84]

Development

Python's development is conducted largely through the Python Enhancement Proposal (PEP) process. PEPs are standardized design documents providing general information related to Python, including proposals, descriptions, design rationales, and explanations for language features.[85] Outstanding PEPs are reviewed and commented upon by Van Rossum, the Python project's Benevolent Dictator for Life (leader / language architect).[86] CPython's developers also communicate over a mailing list, python-dev, which is the primary forum for discussion about the language's development; specific issues are discussed in the Roundup bug tracker maintained at python.org.[87] Development takes place at the self-hosted `hg.python.org`.

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.[88]
- 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.[89]

A number of alpha, beta, and release-candidates are also released as previews and for testing before the final release is made. Although there is a rough schedule for each release, this is often pushed back if the code is not ready. The development team monitor the state of the code by running the large unit test suite during development, and using the BuildBot continuous integration system.[90]

Standard library

Python has a large standard library, commonly cited as one of Python's greatest strengths,[91] providing pre-written tools suited to many tasks. This is deliberate and has been described as a "batteries included"[92] Python philosophy. The modules of the standard library can be augmented with custom modules written in either C or Python. Boost C++ Libraries includes a library, Boost.Python, to enable interoperability between C++ and Python. Because of the wide variety of tools provided by the standard library, combined with the ability to use a lower-level language such as C and C++, which is already capable of interfacing between other libraries, Python can be a powerful glue language between languages and tools.

The standard library is particularly well tailored to writing Internet-facing applications, with a large number of standard formats and protocols (such as MIME and HTTP) already supported. Modules for creating graphical user interfaces, connecting to relational databases, arithmetic with arbitrary precision decimals, manipulating

regular expressions, and doing unit testing are also included.[93]

Some parts of the standard library are covered by specifications (for example, the WSGI implementation `wsgiref` follows PEP 333[94]), but the majority of the modules are not. They are specified by their code, internal documentation, and test suite (if supplied). However, because most of the standard library is cross-platform Python code, there are only a few modules that must be altered or completely rewritten by alternative implementations.

The standard library is not essential to run Python or embed Python within an application. Blender 2.49 for instance omits most of the standard library.

For software testing, the standard library provides the `unittest` and `doctest` modules.

Influence on other languages

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

- Pyrex and its derivative Cython are code translators that are targeted at writing fast C extensions for the CPython interpreter. The language is mostly Python with syntax extensions for C and C++ features. Both languages produce compilable C code as output.
- Boo uses indentation, a similar syntax, and a similar object model. However, Boo uses static typing and is closely integrated with the .NET Framework.[95]
- Cobra uses indentation and a similar syntax. Cobra's "Acknowledgements" document lists Python first among languages that influenced it.[96] However, Cobra directly supports design-by-contract, unit tests and optional static typing.[97]
- ECMAScript borrowed iterators, generators, and list comprehensions from Python.[98]
- Go is described as incorporating the "development speed of working in a dynamic language like Python".[99]
- Groovy was motivated by the desire to bring the Python design philosophy to Java.[100]
- Karel the Robot is a popular educational programming language created in 1981 at the Stanford University by R.E. Pattis. The original syntax was close to Pascal, but a new edition in NCLab has a syntax very similar to Python.
- OCaml has an optional syntax, called twt (The Whitespace Thing), inspired by Python and Haskell.[101]
- 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."[102]
- Alore programming language, a new language with optional typing with Python syntax.[103]

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[104] and Erlang[105] because of Python's influence.

See also

- Comparison of Python integrated development environments
- Comparison of command shells
- Comparison of programming languages
- List of programming languages
- Pyladies

References

1. ^ "Python 3.3.0 Release" (http://www.python.org/download/releases/3.3.0/) . Python Software Foundation. http://www.python.org/download/releases/3.3.0/. Retrieved 24 September 2012.
2. ^ ***a*** ***b*** "Why was Python created in the first place?" (http://docs.python.org/faq/general.html#why-was-python-created-in-the-first-place) . *General Python FAQ*. Python Software Foundation. http://docs.python.org/faq/general.html#why-was-python-created-in-the-first-place. Retrieved 22 March 2007.
3. ^ Kuchling, Andrew M. (22 December 2006). "Interview with Guido van Rossum (July 1998)" (http://www.amk.ca/python/writing/gvr-interview) . *amk.ca*. http://www.amk.ca/python/writing/gvr-interview. Retrieved 12 March 2012.
4. ^ 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)*. http://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."
5. ^ ***a*** ***b*** "Classes" (http://docs.python.org/tutorial/classes.html) . *The Python Tutorial*. Python Software Foundation. http://docs.python.org/tutorial/classes.html. Retrieved 20 February 2012. "It is a mixture of the class mechanisms found in C++ and Modula-3"
6. ^ Simionato, Michele. "The Python 2.3 Method Resolution Order" (http://www.python.org/download/releases/2.3/mro/) . Python Software Foundation. http://www.python.org/download/releases/2.3/mro/. "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"
7. ^ Kuchling, A. M.. "Functional Programming HOWTO" (http://docs.python.org/howto/functional.html) . *Python v2.7.2 documentation*. Python Software Foundation. http://docs.python.org/howto/functional.html. Retrieved 9 February 2012.
8. ^ Schemenauer, Neil; Peters, Tim; Hetland, Magnus Lie (18 May 2001). "PEP 255 – Simple Generators" (http://www.python.org/dev/peps/pep-0255/) . *Python Enhancement Proposals*. Python Software Foundation. http://www.python.org/dev/peps/pep-0255/. Retrieved 9 February 2012.
9. ^ Smith, Kevin D.; Jewett, Jim J.; Montanaro, Skip; Baxter, Anthony (2 September 2004). "PEP 318 – Decorators for Functions and Methods" (http://www.python.org/dev/peps/pep-0318/#why) . *Python Enhancement Proposals*. Python Software Foundation. http://www.python.org/dev/peps/pep-0318/#why. Retrieved 24 February 2012.
10. ^ "More Control Flow Tools" (http://docs.python.org/py3k/tutorial/controlflow.html#lambda-forms) . *Python 3 documentation*. Python Software Foundation. http://docs.python.org/py3k/tutorial/controlflow.html#lambda-forms. Retrieved 5 August 2012.
11. ^ 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.
12. ^ Dave Kuhlman. "A Python Book: Beginning Python, Advanced Python, and Python Exercises" (http://cutter.rexx.com/~dkuhlman/python_book_01.html) . http://cutter.rexx.com/~dkuhlman/python_book_01.html. "Python is a high-level general purpose programming language"
13. ^ Mark Summerfield. *Rapid GUI Programming with Python and Qt*. "If you are new to Python: Welcome! You are about to discover a language that is clear to read and write, and that is concise without being cryptic."
14. ^ "The Python Wiki" (http://wiki.python.org/moin/) . http://wiki.python.org/moin/. Retrieved 2012-09-12."Python combines remarkable power with very clear syntax."
15. ^ Mark Summerfield. *Rapid GUI Programming with Python and Qt*. "Python is a very easy-to-use

15. ^ Mark Summerfield. *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 in, say, C++ or Java."
16. ^ "About Python" (http://www.python.org/about) . Python Software Foundation. http://www.python.org/about. 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."
17. ^ ***a b*** Venners, Bill (13 January 2003). "The Making of Python" (http://www.artima.com/intv/pythonP.html) . *Artima Developer*. Artima. http://www.artima.com/intv/pythonP.html. Retrieved 22 March 2007.
18. ^ 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. http://python-history.blogspot.com/2009/01/brief-timeline-of-python.html. Retrieved 20 January 2009.
19. ^ 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*. http://mail.python.org/pipermail/python-dev/2000-August/008881.html. Retrieved 13 March 2011.
20. ^ Kuchling, A. M.; Zadka, Moshe (16 October 2000). "What's New in Python 2.0" (http://docs.python.org/whatsnew/2.0.html) . Python Software Foundation. http://docs.python.org/whatsnew/2.0.html. Retrieved 11 February 2012.
21. ^ "Python 3.0 Release" (http://python.org/download/releases/3.0/) . Python Software Foundation. http://python.org/download/releases/3.0/. Retrieved 8 July 2009.
22. ^ van Rossum, Guido (5 April 2006). "PEP 3000 – Python 3000" (http://www.python.org/dev/peps/pep-3000/) . *Python Enhancement Proposals*. Python Software Foundation. http://www.python.org/dev/peps/pep-3000/. Retrieved 27 June 2009.
23. ^ "TIOBE Programming Community Index for March 2012" (http://www.tiobe.com/index.php/content/paperinfo/tpci/) . TIOBE Software. March 2012. http://www.tiobe.com/index.php/content/paperinfo/tpci/. Retrieved 25 March 2012.
24. ^ The Cain Gang Ltd.. "Python Metaclasses: Who? Why? When?" (http://www.webcitation.org/5lubkaJRc) (PDF). Archived from the original (http://www.python.org/community/pycon/dc2004/papers/24/metaclasses-pycon.pdf) on 10 December 2009. http://www.webcitation.org/5lubkaJRc. Retrieved 27 June 2009.
25. ^ "3.3. Special method names" (http://docs.python.org/3.0/reference/datamodel.html#special-method-names) . *The Python Language Reference*. Python Software Foundation. http://docs.python.org/3.0/reference/datamodel.html#special-method-names. Retrieved 27 June 2009.
26. ^ "PyDBC: method preconditions, method postconditions and class invariants for Python" (http://www.nongnu.org/pydbc/) . http://www.nongnu.org/pydbc/. Retrieved 24 September 2011.
27. ^ "Contracts for Python" (http://www.wayforward.net/pycontract/) . http://www.wayforward.net/pycontract/. Retrieved 24 September 2011.
28. ^ "PyDatalog" (https://sites.google.com/site/pydatalog/) . https://sites.google.com/site/pydatalog/. Retrieved 22 July 2012.
29. ^ "6.5 itertools – Functions creating iterators for efficient looping" (http://docs.python.org/lib/module-itertools.html) . Docs.python.org. http://docs.python.org/lib/module-itertools.html. Retrieved 24 November 2008.
30. ^ ***a b*** Peters, Tim (19 August 2004). "PEP 20 – The Zen of Python" (http://www.python.org/dev/peps/pep-0020/) . *Python Enhancement Proposals*. Python Software Foundation. http://www.python.org/dev/peps/pep-0020/. Retrieved 24 November 2008.
31. ^ Python Culture (http://www.python.org/dev/culture/)
32. ^ "Python Patterns – An Optimization Anecdote" (http://www.python.org/doc/essays/list2str/) . *Python Essays*. Python Software Foundation. http://www.python.org/doc/essays/list2str/. Retrieved 19 February 2012.
33. ^ ***a b*** "Whetting Your Appetite" (http://docs.python.org/tutorial/appetite.html) . *The Python Tutorial*. Python Software Foundation. http://docs.python.org/tutorial/appetite.html. Retrieved 20 February 2012.
34. ^ Shaw, Zed A.. "Learn Python the Hard Way" (http://learnpythonthehardway.org) . learnpythonthehardway.org. http://learnpythonthehardway.org. Retrieved 20 February 2012.
35. ^ "In Python, should I use else after a return in an if block?" (http://stackoverflow.com/questions/5033906/in-python-should-i-use-else-after-a-return-in-an-if-block) . *Stack Overflow*. Stack Exchange. 17 February 2011. http://stackoverflow.com/questions/5033906/in-python-should-i-use-else-after-a-return-in-an-if-block. Retrieved 6

May 2011.
36. ^ David Goodger. "Code Like a Pythonista: Idiomatic Python" (http://python.net/~goodger/projects/pycon/2007/idiomatic/handout.html) . http://python.net/~goodger/projects/pycon/2007/idiomatic/handout.html.; "How to think like a Pythonista" (http://python.net/crew/mwh/hacks/objectthink.html) . http://python.net/crew/mwh/hacks/objectthink.html.
37. ^ Documentation of the PSP Scripting API can be found at *JASC Paint Shop Pro 9: Additional Download Resources* (http://www.jasc.com/support/customercare/articles/psp9components.asp)
38. ^ "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. http://webhelp.esri.com/arcgisdesktop/9.2/index.cfm?TopicName=About_getting_started_with_writing_geoprocessing_scripts. Retrieved 11 February 2012.
39. ^ porkbelly (23 July 2007). "Stackless Python 2.5" (http://www.webcitation.org/5ru5w3vSR) . *Eve Insider Dev Blog*. CCP Games. Archived from the original (http://myeve.eve-online.com/devblog.asp?a=blog&bid=488) on 10 August 2010. http://www.webcitation.org/5ru5w3vSR. "As you may well know, your favorite space-game owes its existence to the programming language Python"
40. ^ 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. http://www.webcitation.org/5ru5VItfv. "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."
41. ^ "Python Language Guide (v1.0)" (http://www.webcitation.org/5ru5FHxfV) . *Google Documents List Data API v1.0*. Google. Archived from the original (http://code.google.com/apis/documents/docs/1.0/developers_guide_python.html) on 10 August 2010. http://www.webcitation.org/5ru5FHxfV.
42. ^ "Natural Language Toolkit" (http://www.nltk.org) . http://www.nltk.org. Retrieved 31 July 2012.
43. ^ Paine, Jocelyn, ed. (August 2005). "AI in Python" (http://www.ainewsletter.com/newsletters/aix_0508.htm#python_ai_ai) . *AI Expert Newsletter* (Amzi!). http://www.ainewsletter.com/newsletters/aix_0508.htm#python_ai_ai. Retrieved 11 February 2012.
44. ^ Stratton, Cort. "PyAIML 0.8.5 – An interpreter package for AIML, the Artificial Intelligence Markup Language" (http://pypi.python.org/pypi/PyAIML) . Python Software Foundation. http://pypi.python.org/pypi/PyAIML. Retrieved 11 February 2012.
45. ^ 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. http://aima.cs.berkeley.edu/. Retrieved 11 February 2012.
46. ^ "Pardus: (http://www.pardus.org.tr/eng/projects/comar/PythonInPardus.html) TÜBİTAK/UEKAE". pardus.org.tr. http://www.pardus.org.tr/eng/projects/comar/PythonInPardus.html. Retrieved 24 November 2008.
47. ^ "Welcome to Immunity Debugger" (http://www.immunitysec.com/products-immdbg.shtml) . Immunity. http://www.immunitysec.com/products-immdbg.shtml. Retrieved 24 November 2008. "CORE Security Technologies' open source software repository" (http://oss.coresecurity.com/) . Core Security Technologies. http://oss.coresecurity.com/. Retrieved 11 February 2012.
48. ^ Surribas, Nicolas. "Wapiti – Web application vulnerability scanner / security auditor" (http://wapiti.sourceforge.net/) . SourceForge. http://wapiti.sourceforge.net/. Retrieved 24 November 2008.
49. ^ ***a*** ***b*** "Quotes about Python" (http://www.python.org/about/quotes/) . Python Software Foundation. http://www.python.org/about/quotes/. Retrieved 8 January 2012.
50. ^ Figgins, Stephen (17 July 2003). "BitTorrent Style" (http://www.onlamp.com/pub/a/python/2003/7/17/pythonnews.html) . *ONLamp.com*. O'Reilly Media. http://www.onlamp.com/pub/a/python/2003/7/17/pythonnews.html. Retrieved 24 September 2011.
51. ^ "SPOTIFY AND PYTHON: LOVE AT FIRST SIGHT" (https://ep2012.europython.eu/conference/talks/spotify-and-python-love-first-sight) . 2011. https://ep2012.europython.eu/conference/talks/spotify-and-python-love-first-sight. Retrieved 5 July 2012.
52. ^ "Organizations Using Python" (http://wiki.python.org/moin/OrganizationsUsingPython) . Python Software

Foundation. http://wiki.python.org/moin/OrganizationsUsingPython. Retrieved 15 January 2009.
53. ^ "Python : the holy grail of programming" (http://cdsweb.cern.ch/journal/CERNBulletin/2006/31/News%20Articles/974627?ln=en) . *CERN Bulletin* (CERN Publications) (31/2006). 31 July 2006. http://cdsweb.cern.ch/journal/CERNBulletin/2006/31/News%20Articles/974627?ln=en. Retrieved 11 February 2012.
54. ^ Shafer, Daniel G. (17 January 2003). "Python Streamlines Space Shuttle Mission Design" (http://www.python.org/about/success/usa/) . Python Software Foundation. http://www.python.org/about/success/usa/. Retrieved 24 November 2008.
55. ^ Fortenberry, Tim (17 January 2003). "Industrial Light & Magic Runs on Python" (http://www.python.org/about/success/ilm/) . Python Software Foundation. http://www.python.org/about/success/ilm/. Retrieved 11 February 2012.
56. ^ 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. http://www.eweek.com/c/a/Application-Development/Python-Slithers-into-Systems/. Retrieved 24 September 2011.
57. ^ "What is Sugar?" (http://sugarlabs.org/go/Sugar) . Sugar Labs. http://sugarlabs.org/go/Sugar. Retrieved 11 February 2012.
58. ^ TIOBE Software Index (2012). "TIOBE Programming Community Index Python" (http://www.tiobe.com/index.php/paperinfo/tpci/Python.html) . http://www.tiobe.com/index.php/paperinfo/tpci/Python.html. Retrieved 30 April 2012.
59. ^ "Is Python a good language for beginning programmers?" (http://docs.python.org/faq/general.html#is-python-a-good-language-for-beginning-programmers) . *General Python FAQ*. Python Software Foundation. http://docs.python.org/faq/general.html#is-python-a-good-language-for-beginning-programmers. Retrieved 21 March 2007.
60. ^ "Myths about indentation in Python" (http://www.secnetix.de/~olli/Python/block_indentation.hawk) . Secnetix.de. http://www.secnetix.de/~olli/Python/block_indentation.hawk. Retrieved 19 April 2011.
61. ^ 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. http://www.artima.com/weblogs/viewpost.jsp?thread=147358. Retrieved 21 March 2007.
62. ^ van Rossum, Guido; Eby, Phillip J. (10 May 2005). "PEP 342 – Coroutines via Enhanced Generators" (http://www.python.org/dev/peps/pep-0342/) . *Python Enhancement Proposals*. Python Software Foundation. http://www.python.org/dev/peps/pep-0342/. Retrieved 19 February 2012.
63. ^ Hettinger, Raymond (30 January 2002). "PEP 289 – Generator Expressions" (http://www.python.org/dev/peps/pep-0289/) . *Python Enhancement Proposals*. Python Software Foundation. http://www.python.org/dev/peps/pep-0289/. Retrieved 19 February 2012.
64. ^ van Rossum, Guido; Hettinger, Raymond (7 February 2003). "PEP 308 – Conditional Expressions" (http://www.python.org/dev/peps/pep-0308/) . *Python Enhancement Proposals*. Python Software Foundation. http://www.python.org/dev/peps/pep-0308/. Retrieved 13 July 2011.
65. ^ "Why must 'self' be used explicitly in method definitions and calls?" (http://docs.python.org/faq/design.html#why-must-self-be-used-explicitly-in-method-definitions-and-calls) . *Design and History FAQ*. Python Software Foundation. http://docs.python.org/faq/design.html#why-must-self-be-used-explicitly-in-method-definitions-and-calls. Retrieved 19 February 2012.
66. ^ "The Python Language Reference, section 3.3. New-style and classic classes, for release 2.7.1" (http://docs.python.org/reference/datamodel.html#new-style-and-classic-classes) . http://docs.python.org/reference/datamodel.html#new-style-and-classic-classes. Retrieved 12 January 2011.
67. ^ Zadka, Moshe; van Rossum, Guido (11 March 2001). "PEP 237 – Unifying Long Integers and Integers" (http://www.python.org/dev/peps/pep-0237/) . *Python Enhancement Proposals*. Python Software Foundation. http://www.python.org/dev/peps/pep-0237/. Retrieved 24 September 2011.
68. ^ "Why Python's Integer Division Floors" (http://python-history.blogspot.com/2010/08/why-pythons-integer-division-floors.html) . http://python-history.blogspot.com/2010/08/why-pythons-integer-division-floors.html. Retrieved 25 August 2010.

69. ^ "round" (http://docs.python.org/library/functions.html#round) , *The Python standard library, release 2.7, §2: Built-in functions*, http://docs.python.org/library/functions.html#round, retrieved 14 August 2011
70. ^ "round" (http://docs.python.org/py3k/library/functions.html#round) , *The Python standard library, release 3.2, §2: Built-in functions*, http://docs.python.org/py3k/library/functions.html#round, retrieved 14 August 2011
71. ^ Python Essential Reference, David M Beazley
72. ^ van Rossum, Guido (5 June 2001). "PEP 7 – Style Guide for C Code" (http://www.python.org/dev/peps/pep-0007/) . *Python Enhancement Proposals*. Python Software Foundation. http://www.python.org/dev/peps/pep-0007/. Retrieved 24 November 2008.
73. ^ "CPython byte code" (http://docs.python.org/lib/bytecodes.html) . Docs.python.org. http://docs.python.org/lib/bytecodes.html. Retrieved 19 April 2011.
74. ^ "Python 2.5 internals" (http://www.troeger.eu/teaching/pythonvm08.pdf) (PDF). http://www.troeger.eu/teaching/pythonvm08.pdf. Retrieved 19 April 2011.
75. ^ "O'Reilly – An Interview with Guido van Rossum" (http://www.oreilly.com/pub/a/oreilly/frank/rossum_1099.html) . Oreilly.com. http://www.oreilly.com/pub/a/oreilly/frank/rossum_1099.html. Retrieved 24 November 2008.
76. ^ "Plans for optimizing Python" (http://code.google.com/p/unladen-swallow/wiki/ProjectPlan) . *Google Project Hosting*. Google. 15 December 2009. http://code.google.com/p/unladen-swallow/wiki/ProjectPlan. Retrieved 24 September 2011.
77. ^ "PythonCE" (http://pythonce.sourceforge.net/) . Pythonce.sourceforge.net. http://pythonce.sourceforge.net/. Retrieved 19 April 2011.
78. ^ "PyMite: Python-on-a-chip" (http://wiki.python.org/moin/PyMite) . Wiki.python.org. 19 April 2009. http://wiki.python.org/moin/PyMite. Retrieved 19 April 2011.
79. ^ "PyMite" (http://deanandara.com/PyMite/2010-State.html) . Deanandara.com. http://deanandara.com/PyMite/2010-State.html. Retrieved 19 April 2011.
80. ^ "PyMite" (http://pythononachip.org/) . Python-on-a-Chip. http://pythononachip.org/. Retrieved 19 April 2011.
81. ^ "android-scripting – Scripting Layer for Android brings scripting languages to Android" (http://code.google.com/p/android-scripting/) . *Google Project Hosting*. Google. http://code.google.com/p/android-scripting/. Retrieved 24 September 2011.
82. ^ "python-for-android - A framework allowing to produce Python applications for android, provide a GUI interface ((http://github.com/kivy/python-for-android/) Kivy)". Github. http://github.com/kivy/python-for-android/. Retrieved 26 July 2012.
83. ^ "kivy-ios - iOS port of Kivy, allowing to create python applications on iOS" (http://github.com/kivy/kivy-ios) . Github. http://github.com/kivy/kivy-ios. Retrieved 26 July 2012.
84. ^ "Introduction to Psyco" (http://psyco.sourceforge.net/introduction.html) . Psyco.sourceforge.net. http://psyco.sourceforge.net/introduction.html. Retrieved 19 April 2011.
85. ^ Warsaw, Barry; Hylton, Jeremy; Goodger, David (13 June 2000). "PEP 1 – PEP Purpose and Guidelines" (http://www.python.org/dev/peps/pep-0001/) . *Python Enhancement Proposals*. Python Software Foundation. http://www.python.org/dev/peps/pep-0001/. Retrieved 19 April 2011.
86. ^ van Rossum, Guido (7 March 2002). "Parade of the PEPs" (http://www.python.org/doc/essays/pepparade/) . *Python Essays*. Python Software Foundation. http://www.python.org/doc/essays/pepparade/. Retrieved 19 February 2012.
87. ^ Cannon, Brett. "Guido, Some Guys, and a Mailing List: How Python is Developed" (http://classic-web.archive.org/web/20080229153753/http://www.python.org/dev/intro/) . *python.org*. Python Software Foundation. http://classic-web.archive.org/web/20080229153753/http://www.python.org/dev/intro/. Retrieved 27 June 2009.
88. ^ Norwitz, Neal (8 April 2002). "[Python-Dev] Release Schedules (was Stability & change)" (http://mail.python.org/pipermail/python-dev/2002-April/022739.html) . http://mail.python.org/pipermail/python-dev/2002-April/022739.html. Retrieved 27 June 2009.
89. ^ Aahz; Baxter, Anthony (15 March 2001). "PEP 6 – Bug Fix Releases" (http://www.python.org/dev/peps/pep-0006/) . *Python Enhancement Proposals*. Python Software Foundation. http://www.python.org/dev/peps/pep-0006/. Retrieved 27 June 2009.
90. ^ "Python Buildbot" (http://python.org/dev/buildbot/) . *Python Developer's Guide*. Python Software Foundation.

90. ^ "Python Buildbot" (http://python.org/dev/buildbot) . *Python Developer's Guide*. Python Software Foundation. http://python.org/dev/buildbot/. Retrieved 24 September 2011.
91. ^ 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. http://www.oracle.com/technetwork/articles/piotrowski-pythoncore-084049.html. Retrieved 12 March 2012.
92. ^ "About Python" (http://www.python.org/about/) . *python.org*. Python Software Foundation. http://www.python.org/about/. Retrieved 27 June 2009.
93. ^ Batista, Facundo (17 October 2003). "PEP 327 – Decimal Data Type" (http://www.python.org/dev/peps/pep-0327/) . *Python Enhancement Proposals*. Python Software Foundation. http://www.python.org/dev/peps/pep-0327/. Retrieved 24 November 2008.
94. ^ Eby, Phillip J. (7 December 2003). "PEP 333 – Python Web Server Gateway Interface v1.0" (http://www.python.org/dev/peps/pep-0333/) . *Python Enhancement Proposals*. Python Software Foundation. http://www.python.org/dev/peps/pep-0333/. Retrieved 19 February 2012.
95. ^ "Gotchas for Python Users" (http://boo.codehaus.org/Gotchas+for+Python+Users) . *boo.codehaus.org*. Codehaus Foundation. http://boo.codehaus.org/Gotchas+for+Python+Users. Retrieved 24 November 2008.
96. ^ Esterbrook, Charles. "Acknowledgements" (http://cobra-language.com/docs/acknowledgements/) . *cobra-language.com*. Cobra Language. http://cobra-language.com/docs/acknowledgements/. Retrieved 7 April 2010.
97. ^ Esterbrook, Charles. "Comparison to Python" (http://cobra-language.com/docs/python/) . *cobra-language.com*. Cobra Language. http://cobra-language.com/docs/python/. Retrieved 7 April 2010.
98. ^ "Proposals: iterators and generators [ES4 Wiki]" (http://wiki.ecmascript.org/doku.php?id=proposals:iterators_and_generators) . wiki.ecmascript.org. http://wiki.ecmascript.org/doku.php?id=proposals:iterators_and_generators. Retrieved 24 November 2008.
99. ^ 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. http://www.techcrunch.com/2009/11/10/google-go-language/. Retrieved 29 January 2010.
100. ^ James Strachan (29 August 2003). "Groovy – the birth of a new dynamic language for the Java platform" (http://radio.weblogs.com/0112098/2003/08/29.html) . http://radio.weblogs.com/0112098/2003/08/29.html.
101. ^ Lin, Mike. ""The Whitespace Thing" for OCaml" (http://people.csail.mit.edu/mikelin/ocaml+twt/) . Massachusetts Institute of Technology. http://people.csail.mit.edu/mikelin/ocaml+twt/. Retrieved 12 April 2009.
102. ^ An Interview with the Creator of Ruby (http://www.linuxdevcenter.com/pub/a/linux/2001/11/29/ruby.html)
103. ^ Alore programming language web site (http://www.alorelang.org/)
104. ^ Kupries, Andreas; Fellows, Donal K. (14 September 2000). "TIP #3: TIP Format" (http://www.tcl.tk/cgi-bin/tct/tip/3.html) . *tcl.tk*. Tcl Developer Xchange. http://www.tcl.tk/cgi-bin/tct/tip/3.html. Retrieved 24 November 2008.
105. ^ Gustafsson, Per; Niskanen, Raimo (29 January 2007). "EEP 1: EEP Purpose and Guidelines" (http://www.erlang.org/eeps/eep-0001.html) . erlang.org. http://www.erlang.org/eeps/eep-0001.html. Retrieved 19 April 2011.

Further reading

- Downey, Allen B. (Version 1.6.6 - May 2012). *Think Python: How to Think Like a Computer Scientist* (http://www.greenteapress.com/thinkpython/html/) . ISBN 978-0-521-72596-5. http://www.greenteapress.com/thinkpython/html/.
- Hamilton, Naomi (5 August 2008). "The A-Z of Programming Languages: Python" (http://www.computerworld.com.au/index.php/id;66665771) . *Computerworld*. http://www.computerworld.com.au/index.php/id;66665771. Retrieved 31 March 2010. – An interview with Guido Van Rossum on Python
- Lutz, Mark (2009). *Learning Python* (4th ed.). O'Reilly Media. ISBN 978-0-596-15806-4.
- Pilgrim, Mark (2004). *Dive Into Python* (http://diveintopython.net) . Apress. ISBN 978-1-59059-356-1. http://diveintopython.net.

- Pilgrim, Mark (2009). *Dive Into Python 3* (http://diveintopython3.net) . Apress. ISBN 978-1-4302-2415-0. http://diveintopython3.net.
- Summerfield, Mark (2009). *Programming in Python 3* (http://www.qtrac.eu/py3book.html) (2nd ed.). Addison-Wesley Professional. ISBN 978-0-321-68056-3. http://www.qtrac.eu/py3book.html.

External links

- Learning materials related to Python Programming at Wikiversity
- Media related to Python (programming language) at Wikimedia Commons
- Official website (http://www.python.org)
- comp.lang.python (news://comp.lang.python) newsgroup (Google Groups archive (http://groups.google.com/group/comp.lang.python/topics)) / python-list mailing list (http://mail.python.org/mailman/listinfo/python-list)
- Python (http://www.dmoz.org/Computers/Programming/Languages/Python/) at the Open Directory Project

Categories: Python (programming language) | Class-based programming languages | Cross-platform free software | Dynamically typed programming languages | High-level programming languages | Object-oriented programming languages | Programming languages created in 1991 | Scripting languages | Text-oriented programming languages

---