

# General Python FAQ

## Contents

- [General Python FAQ](#)
  - [General Information](#)
    - [What is Python?](#)
    - [What is the Python Software Foundation?](#)
    - [Are there copyright restrictions on the use of Python?](#)
    - [Why was Python created in the first place?](#)
    - [What is Python good for?](#)
    - [How does the Python version numbering scheme work?](#)
    - [How do I obtain a copy of the Python source?](#)
    - [How do I get documentation on Python?](#)
    - [I've never programmed before. Is there a Python tutorial?](#)
    - [Is there a newsgroup or mailing list devoted to Python?](#)
    - [How do I get a beta test version of Python?](#)
    - [How do I submit bug reports and patches for Python?](#)
    - [Are there any published articles about Python that I can reference?](#)
    - [Are there any books on Python?](#)
    - [Where in the world is \[www.python.org\]\(http://www.python.org\) located?](#)
    - [Why is it called Python?](#)
    - [Do I have to like "Monty Python's Flying Circus"?](#)
  - [Python in the real world](#)
    - [How stable is Python?](#)
    - [How many people are using Python?](#)
    - [Have any significant projects been done in Python?](#)
    - [What new developments are expected for Python in the future?](#)
    - [Is it reasonable to propose incompatible changes to Python?](#)
    - [Is Python a good language for beginning programmers?](#)

## General Information

### What is Python?

Python is an interpreted, interactive, object-oriented programming language. It incorporates modules, exceptions, dynamic typing, very high level dynamic data types, and classes. Python combines remarkable power with very clear syntax. It has interfaces to many system calls and libraries, as well as to various window systems, and is extensible in C or C++. It is also usable as an extension language for applications that need a programmable interface. Finally, Python is portable: it runs on many Unix variants, on the Mac, and on Windows 2000 and later.

To find out more, start with [The Python Tutorial](#). The [Beginner's Guide to Python](#) links to other introductory tutorials and resources for learning Python.

## What is the Python Software Foundation?

The Python Software Foundation is an independent non-profit organization that holds the copyright on Python versions 2.1 and newer. The PSF's mission is to advance open source technology related to the Python programming language and to publicize the use of Python. The PSF's home page is at <https://www.python.org/psf/>.

Donations to the PSF are tax-exempt in the US. If you use Python and find it helpful, please contribute via [the PSF donation page](#).

## Are there copyright restrictions on the use of Python?

You can do anything you want with the source, as long as you leave the copyrights in and display those copyrights in any documentation about Python that you produce. If you honor the copyright rules, it's OK to use Python for commercial use, to sell copies of Python in source or binary form (modified or unmodified), or to sell products that incorporate Python in some form. We would still like to know about all commercial use of Python, of course.

See [the PSF license page](#) to find further explanations and a link to the full text of the license.

The Python logo is trademarked, and in certain cases permission is required to use it. Consult [the Trademark Usage Policy](#) for more information.

## Why was Python created in the first place?

Here's a very brief summary of what started it all, written by Guido van Rossum:

I had extensive experience with implementing an interpreted language in the ABC group at CWI, and from working with this group I had learned a lot about language design. This is the origin of many Python features, including the use of indentation for statement grouping and the inclusion of very-high-level data types (although the details are all different in Python).

I had a number of gripes about the ABC language, but also liked many of its features. It was impossible to extend the ABC language (or its implementation) to remedy my complaints – in fact its lack of extensibility was one of its biggest problems. I had some experience with using Modula-2+ and talked with the designers of Modula-3 and read the Modula-3

report. Modula-3 is the origin of the syntax and semantics used for exceptions, and some other Python features.

I was working in the Amoeba distributed operating system group at CWI. We needed a better way to do system administration than by writing either C programs or Bourne shell scripts, since Amoeba had its own system call interface which wasn't easily accessible from the Bourne shell. My experience with error handling in Amoeba made me acutely aware of the importance of exceptions as a programming language feature.

It occurred to me that a scripting language with a syntax like ABC but with access to the Amoeba system calls would fill the need. I realized that it would be foolish to write an Amoeba-specific language, so I decided that I needed a language that was generally extensible.

During the 1989 Christmas holidays, I had a lot of time on my hand, so I decided to give it a try. During the next year, while still mostly working on it in my own time, Python was used in the Amoeba project with increasing success, and the feedback from colleagues made me add many early improvements.

In February 1991, after just over a year of development, I decided to post to USENET. The rest is in the `Misc/HISTORY` file.

## What is Python good for?

Python is a high-level general-purpose programming language that can be applied to many different classes of problems.

The language comes with a large standard library that covers areas such as string processing (regular expressions, Unicode, calculating differences between files), Internet protocols (HTTP, FTP, SMTP, XML-RPC, POP, IMAP, CGI programming), software engineering (unit testing, logging, profiling, parsing Python code), and operating system interfaces (system calls, filesystems, TCP/IP sockets). Look at the table of contents for [The Python Standard Library](#) to get an idea of what's available. A wide variety of third-party extensions are also available. Consult [the Python Package Index](#) to find packages of interest to you.

## How does the Python version numbering scheme work?

Python versions are numbered A.B.C or A.B. A is the major version number – it is only incremented for really major changes in the language. B is the minor version number, incremented for less earth-shattering changes. C is the micro-level – it is in-

cremented for each bugfix release. See [PEP 6](#) for more information about bugfix releases.

Not all releases are bugfix releases. In the run-up to a new major release, a series of development releases are made, denoted as alpha, beta, or release candidate. Alphas are early releases in which interfaces aren't yet finalized; it's not unexpected to see an interface change between two alpha releases. Betas are more stable, preserving existing interfaces but possibly adding new modules, and release candidates are frozen, making no changes except as needed to fix critical bugs.

Alpha, beta and release candidate versions have an additional suffix. The suffix for an alpha version is "aN" for some small number N, the suffix for a beta version is "bN" for some small number N, and the suffix for a release candidate version is "cN" for some small number N. In other words, all versions labeled 2.0aN precede the versions labeled 2.0bN, which precede versions labeled 2.0cN, and *those* precede 2.0.

You may also find version numbers with a "+" suffix, e.g. "2.2+". These are unreleased versions, built directly from the CPython development repository. In practice, after a final minor release is made, the version is incremented to the next minor version, which becomes the "a0" version, e.g. "2.4a0".

See also the documentation for [sys.version](#), [sys.hexversion](#), and [sys.version\\_info](#).

## How do I obtain a copy of the Python source?

The latest Python source distribution is always available from python.org, at <https://www.python.org/downloads/>. The latest development sources can be obtained at <https://github.com/python/cpython/>.

The source distribution is a gzipped tar file containing the complete C source, Sphinx-formatted documentation, Python library modules, example programs, and several useful pieces of freely distributable software. The source will compile and run out of the box on most UNIX platforms.

Consult the [Getting Started section of the Python Developer's Guide](#) for more information on getting the source code and compiling it.

## How do I get documentation on Python?

The standard documentation for the current stable version of Python is available at <https://docs.python.org/3/>. PDF, plain text, and downloadable HTML versions are also available at <https://docs.python.org/3/download.html>.

The documentation is written in reStructuredText and processed by [the Sphinx documentation tool](#). The reStructuredText source for the documentation is part of the Python source distribution.

## I've never programmed before. Is there a Python tutorial?

There are numerous tutorials and books available. The standard documentation includes [The Python Tutorial](#).

Consult [the Beginner's Guide](#) to find information for beginning Python programmers, including lists of tutorials.

## Is there a newsgroup or mailing list devoted to Python?

There is a newsgroup, *comp.lang.python*, and a mailing list, [python-list](#). The newsgroup and mailing list are gatewayed into each other – if you can read news it's unnecessary to subscribe to the mailing list. *comp.lang.python* is high-traffic, receiving hundreds of postings every day, and Usenet readers are often more able to cope with this volume.

Announcements of new software releases and events can be found in *comp.lang.python.announce*, a low-traffic moderated list that receives about five postings per day. It's available as [the python-announce mailing list](#).

More info about other mailing lists and newsgroups can be found at <https://www.python.org/community/lists/>.

## How do I get a beta test version of Python?

Alpha and beta releases are available from <https://www.python.org/downloads/>. All releases are announced on the *comp.lang.python* and *comp.lang.python.announce* newsgroups and on the Python home page at <https://www.python.org/>; an RSS feed of news is available.

You can also access the development version of Python through Git. See [The Python Developer's Guide](#) for details.

## How do I submit bug reports and patches for Python?

To report a bug or submit a patch, please use the Roundup installation at <https://bugs.python.org/>.

You must have a Roundup account to report bugs; this makes it possible for us to contact you if we have follow-up questions. It will also enable Roundup to send you

updates as we act on your bug. If you had previously used SourceForge to report bugs to Python, you can obtain your Roundup password through Roundup's [password reset procedure](#).

For more information on how Python is developed, consult [the Python Developer's Guide](#).

## Are there any published articles about Python that I can reference?

It's probably best to cite your favorite book about Python.

The very first article about Python was written in 1991 and is now quite outdated.

Guido van Rossum and Jelke de Boer, "Interactively Testing Remote Servers Using the Python Programming Language", CWI Quarterly, Volume 4, Issue 4 (December 1991), Amsterdam, pp 283–303.

## Are there any books on Python?

Yes, there are many, and more are being published. See the python.org wiki at <https://wiki.python.org/moin/PythonBooks> for a list.

You can also search online bookstores for "Python" and filter out the Monty Python references; or perhaps search for "Python" and "language".

## Where in the world is [www.python.org](http://www.python.org) located?

The Python project's infrastructure is located all over the world. [www.python.org](http://www.python.org) is graciously hosted by [Rackspace](#), with CDN caching provided by [Fastly](#). [Upfront Systems](#) hosts [bugs.python.org](http://bugs.python.org). Many other Python services like [the Wiki](#) are hosted by [Oregon State University Open Source Lab](#).

## Why is it called Python?

When he began implementing Python, Guido van Rossum was also reading the published scripts from "[Monty Python's Flying Circus](#)", a BBC comedy series from the 1970s. Van Rossum thought he needed a name that was short, unique, and slightly mysterious, so he decided to call the language Python.

## Do I have to like "Monty Python's Flying Circus"?

No, but it helps. :)

# Python in the real world

## How stable is Python?

Very stable. New, stable releases have been coming out roughly every 6 to 18 months since 1991, and this seems likely to continue. Currently there are usually around 18 months between major releases.

The developers issue “bugfix” releases of older versions, so the stability of existing releases gradually improves. Bugfix releases, indicated by a third component of the version number (e.g. 2.5.3, 2.6.2), are managed for stability; only fixes for known problems are included in a bugfix release, and it’s guaranteed that interfaces will remain the same throughout a series of bugfix releases.

The latest stable releases can always be found on the [Python download page](#). There are two recommended production-ready versions at this point in time, because at the moment there are two branches of stable releases: 2.x and 3.x. Python 3.x may be less useful than 2.x, since currently there is more third party software available for Python 2 than for Python 3. Python 2 code will generally not run unchanged in Python 3.

## How many people are using Python?

There are probably tens of thousands of users, though it’s difficult to obtain an exact count.

Python is available for free download, so there are no sales figures, and it’s available from many different sites and packaged with many Linux distributions, so download statistics don’t tell the whole story either.

The comp.lang.python newsgroup is very active, but not all Python users post to the group or even read it.

## Have any significant projects been done in Python?

See <https://www.python.org/about/success> for a list of projects that use Python. Consulting the proceedings for [past Python conferences](#) will reveal contributions from many different companies and organizations.

High-profile Python projects include [the Mailman mailing list manager](#) and [the Zope application server](#). Several Linux distributions, most notably [Red Hat](#), have written part or all of their installer and system administration software in Python. Companies that use Python internally include Google, Yahoo, and Lucasfilm Ltd.

## What new developments are expected for Python in the future?

See <https://www.python.org/dev/peps/> for the Python Enhancement Proposals (PEPs). PEPs are design documents describing a suggested new feature for Python, providing a concise technical specification and a rationale. Look for a PEP titled “Python X.Y Release Schedule”, where X.Y is a version that hasn’t been publicly released yet.

New development is discussed on [the python-dev mailing list](#).

## Is it reasonable to propose incompatible changes to Python?

In general, no. There are already millions of lines of Python code around the world, so any change in the language that invalidates more than a very small fraction of existing programs has to be frowned upon. Even if you can provide a conversion program, there’s still the problem of updating all documentation; many books have been written about Python, and we don’t want to invalidate them all at a single stroke.

Providing a gradual upgrade path is necessary if a feature has to be changed. [PEP 5](#) describes the procedure followed for introducing backward-incompatible changes while minimizing disruption for users.

## Is Python a good language for beginning programmers?

Yes.

It is still common to start students with a procedural and statically typed language such as Pascal, C, or a subset of C++ or Java. Students may be better served by learning Python as their first language. Python has a very simple and consistent syntax and a large standard library and, most importantly, using Python in a beginning programming course lets students concentrate on important programming skills such as problem decomposition and data type design. With Python, students can be quickly introduced to basic concepts such as loops and procedures. They can probably even work with user-defined objects in their very first course.

For a student who has never programmed before, using a statically typed language seems unnatural. It presents additional complexity that the student must master and slows the pace of the course. The students are trying to learn to think like a computer, decompose problems, design consistent interfaces, and encapsulate data. While



learning to use a statically typed language is important in the long term, it is not necessarily the best topic to address in the students' first programming course.

Many other aspects of Python make it a good first language. Like Java, Python has a large standard library so that students can be assigned programming projects very early in the course that *do* something. Assignments aren't restricted to the standard four-function calculator and check balancing programs. By using the standard library, students can gain the satisfaction of working on realistic applications as they learn the fundamentals of programming. Using the standard library also teaches students about code reuse. Third-party modules such as PyGame are also helpful in extending the students' reach.

Python's interactive interpreter enables students to test language features while they're programming. They can keep a window with the interpreter running while they enter their program's source in another window. If they can't remember the methods for a list, they can do something like this:

```
>>> L = []
>>> dir(L)
['__add__', '__class__', '__contains__', '__delattr__', '__delitem__',
 '__dir__', '__doc__', '__eq__', '__format__', '__ge__',
 '__getattr__', '__getitem__', '__gt__', '__hash__', '__iadd__',
 '__imul__', '__init__', '__iter__', '__le__', '__len__', '__lt__',
 '__mul__', '__ne__', '__new__', '__reduce__', '__reduce_ex__',
 '__repr__', '__reversed__', '__rmul__', '__setattr__', '__setitem__',
 '__sizeof__', '__str__', '__subclasshook__', 'append', 'clear',
 'copy', 'count', 'extend', 'index', 'insert', 'pop', 'remove',
 'reverse', 'sort']
>>> [d for d in dir(L) if '_' not in d]
['append', 'clear', 'copy', 'count', 'extend', 'index', 'insert', 'pop']

>>> help(L.append)
Help on built-in function append:

append(...)
    L.append(object) -> None -- append object to end

>>> L.append(1)
>>> L
[1]
```

With the interpreter, documentation is never far from the student as they are programming.

There are also good IDEs for Python. IDLE is a cross-platform IDE for Python that is written in Python using Tkinter. PythonWin is a Windows-specific IDE. Emacs users will be happy to know that there is a very good Python mode for Emacs. All of these

programming environments provide syntax highlighting, auto-indenting, and access to the interactive interpreter while coding. Consult [the Python wiki](#) for a full list of Python editing environments.

If you want to discuss Python's use in education, you may be interested in joining [the edu-sig mailing list](#).