Python – a tool, not a reptile

Due to historical reasons, languages designed to be utilized in the interpretation manner are often called **scripting languages**, while the source programs encoded using them are called **scripts**.

What is Python?

Python is a widely-used, interpreted, object-oriented, and high-level programming language with dynamic semantics, used for general-purpose programming.

And while you may know the python as a large snake, the name of the Python programming language comes from an old BBC television comedy sketch series called **Monty Python's Flying Circus**.

At the height of its success, the Monty Python team were performing their sketches to live audiences across the world, including at the Hollywood Bowl.

Since Monty Python is considered one of the two fundamental nutrients to a programmer (the other being pizza), Python's creator named the language in honor of the TV show.

Who created Python?

One of the amazing features of Python is the fact that it is actually one person's work. Usually, new programming languages are developed and published by large companies employing lots of professionals, and due to copyright rules, it is very hard to name any of the people involved in the project. Python is an exception.

There are not many languages whose authors are known by name. Python was created by **Guido van Rossum**, born in 1956 in Haarlem, the Netherlands. Of course, Guido van Rossum did not develop and evolve all the Python components himself.

The speed with which Python has spread around the world is a result of the continuous work of thousands (very often anonymous) programmers, testers, users (many of them aren't IT specialists) and enthusiasts, but it must be said that the very first idea (the seed from which Python sprouted) came to one head - Guido's.

A hobby programming project

The circumstances in which Python was created are a bit puzzling. According to Guido van Rossum:

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).*Guido van Rossum*

Python goals

In 1999, Guido van Rossum defined his goals for Python:

- an **easy and intuitive** language just as powerful as those of the major competitors;
- **open source**, so anyone can contribute to its development;
- code that is as **understandable** as plain English;
- **suitable for everyday tasks**, allowing for short development times.

About 20 years later, it is clear that all these intentions have been fulfilled. Some sources say that Python is the most popular programming language in the world, while others claim it's the third or the fifth.

Either way, it still occupies a high rank in the top ten of the PYPL PopularitY of Programming Language and the TIOBE Programming Community Index.

Python isn't a young language. It is **mature and trustworthy**. It's not a one-hit wonder. It's a bright star in the programming firmament, and time spent learning Python is a very good investment.

What makes Python special?

How does it happen that programmers, young and old, experienced and novice, want to use it? How did it happen that large companies adopted Python and implemented their flagship products using it?

There are many reasons - we've listed some of them already, but let's enumerate them again in a more practical manner:

- it's **easy to learn** - the time needed to learn Python is shorter than for many other languages; this means that it's possible to start the actual programming faster;
- it's **easy to teach** - the teaching workload is smaller than that needed by other languages; this means that the teacher can put more emphasis on general (language-independent) programming techniques, not wasting energy on exotic tricks, strange exceptions and incomprehensible rules;

- it's **easy to use** for writing new software - it's often possible to write code faster when using Python;
- it's **easy to understand** - it's also often easier to understand someone else's code faster if it is written in Python;
- it's **easy to obtain, install and deploy** - Python is free, open and multiplatform; not all languages can boast that.

Of course, Python has its drawbacks, too:

- it's not a speed demon - Python does not deliver exceptional performance;
- in some cases it may be resistant to some simpler testing techniques - this may mean that debugging Python's code can be more difficult than with other languages; fortunately, making mistakes is always harder in Python.

It should also be stated that Python is not the only solution of its kind available on the IT market.

It has lots of followers, but there are many who prefer other languages and don't even consider Python for their projects.

Python rivals?

Python has two direct competitors, with comparable properties and predispositions. These are:

- **Perl -** a scripting language originally authored by Larry Wall;
- **Ruby -** a scripting language originally authored by Yukihiro Matsumoto.

The former is more traditional, more conservative than Python, and resembles some of the good old languages derived from the classic C programming language.

In contrast, the latter is more innovative and more full of fresh ideas than Python. Python itself lies somewhere between these two creations.

The Internet is full of forums with infinite discussions on the superiority of one of these three over the others, should you wish to learn more about each of them.

Where can we see Python in action?

We see it every day and almost everywhere. It's used extensively to implement complex **Internet services** like search engines, cloud storage and tools, social media and so on. Whenever you use any of these services, you are actually very close to Python, although you wouldn't know it.

Many **developing tools** are implemented in Python. More and more **everyday use applications** are being written in Python. Lots of **scientists** have abandoned expensive proprietary tools and switched to Python. Lots of IT project **testers** have started using Python to carry out repeatable test procedures. The list is long.

Why not Python?

Despite Python's growing popularity, there are still some niches where Python is absent, or is rarely seen:

- **low-level programming** (sometimes called "close to metal" programming): if you want to implement an extremely effective driver or graphical engine, you wouldn't use Python;
- **applications for mobile devices**: although this territory is still waiting to be conquered by Python, it will most likely happen someday.

There is more than one Python

There are two main kinds of Python, called Python 2 and Python 3.

Python 2 is an older version of the original Python. Its development has since been intentionally stalled, although that doesn't mean that there are no updates to it. On the contrary, the updates are issued on a regular basis, but they are not intended to modify the language in any significant way. They rather fix any freshly discovered bugs and security holes. Python 2's development path has reached a dead end already, but Python 2 itself is still very much alive.

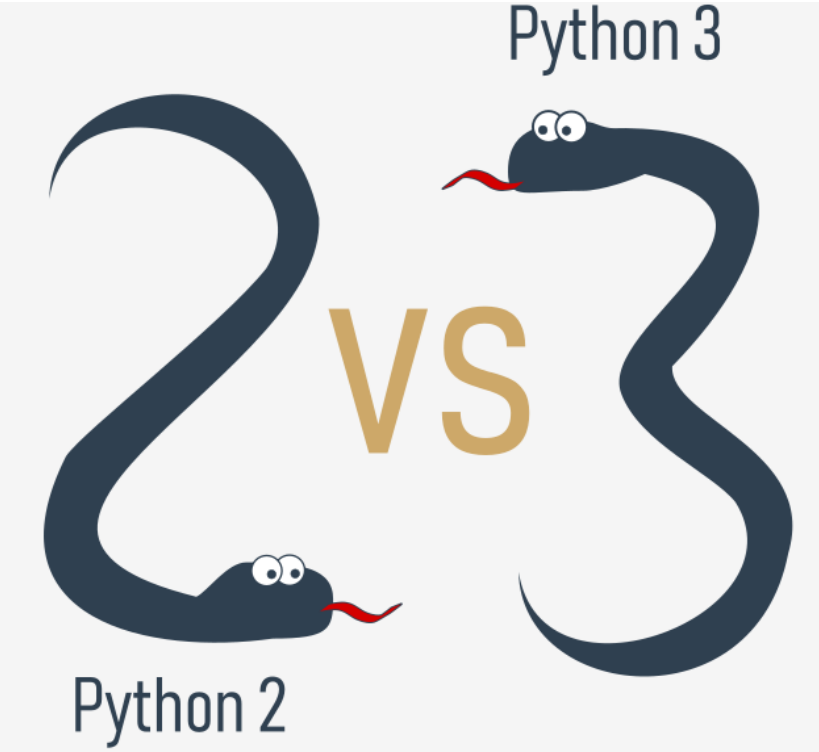
Python 3 is the newer (to be precise, the current) version of the language. It's going through its own evolution path, creating its own standards and habits.

The former is more traditional, more conservative than Python, and resembles some of the good old languages derived from the classic C programming language.

These two versions of Python aren't compatible with each other. Python 2 scripts won't run in a Python 3 environment and vice versa, so if you want the old Python 2 code to be run by a Python 3 interpreter, the only possible solution is to rewrite it, not from scratch, of course, as large parts of the code may remain untouched, but you do have to revise all the code to find all possible incompatibilities. Unfortunately, this process cannot be fully automatized.

It's too hard, too time-consuming, too expensive, and too risky to migrate an old Python 2 application to a new platform. It's possible that rewriting the code will introduce new bugs to it. It's easier and more sensible to leave these systems alone and to improve the existing interpreter, instead of trying to work inside the already functioning source code.

Python 3 isn't just a better version of Python 2 - it is a completely different language, although it's very similar to its predecessor. When you look at them from a distance, they appear to be the same, but when you look closely, though, you notice a lot of differences.



If you're modifying an old existing Python solution, then it's highly likely that it was coded in Python 2. This is the reason why Python 2 is still in use. There are too many existing Python 2 applications to discard it altogether.

NOTE

If you're going to start a new Python project, **you should use Python 3, and this is the version of Python that will be used during this course.**

It is important to remember that there may be smaller or bigger differences between subsequent Python 3 releases (e.g., Python 3.6 introduced ordered dictionary keys by default

under the CPython implementation) - the good news, though, is that all the newer versions of Python 3 are **backwards compatible** with the previous versions of Python 3. Whenever meaningful and important, we will always try to highlight those differences in the course.

All the code samples you will find during the course have been tested against Python 3.4, Python 3.6, and Python 3.7.

Python aka CPython

In addition to Python 2 and Python 3, there is more than one version of each.

First of all, there are the Pythons which are maintained by the people gathered around the PSF (Python Software Foundation), a community that aims to develop, improve, expand, and popularize Python and its environment. The PSF's president is Guido von Rossum himself, and for this reason, these Pythons are called **canonical**. They are also considered to be **reference Pythons**, as any other implementation of the language should follow all standards established by the PSF.

Guido van Rossum used the "C" programming language to implement the very first version of his language and this decision is still in force. All Pythons coming from the PSF are written in the "C" language. There are many reasons for this approach and it has many consequences. One of them (probably the most important) is that thanks to it, Python may be easily ported and migrated to all platforms with the ability to compile and run "C" language programs (virtually all platforms have this feature, which opens up many expansion opportunities for Python).

This is why the PSF implementation is often referred to as **CPython**. This is the most influential Python among all the Pythons in the world.

Cython

Another Python family member is **Cython**.

Cython is one of a possible number of solutions to the most painful of Python's trait - the lack of efficiency. Large and complex mathematical calculations may be easily coded in Python (much easier than in "C" or any other traditional language), but the resulting code's execution may be extremely time-consuming.

How are these two contradictions reconciled? One solution is to write your mathematical ideas using Python, and when you're absolutely sure that your code is correct and produces valid results, you can translate it into "C". Certainly, "C" will run much faster than pure Python.

This is what Cython is intended to do - to automatically translate the Python code (clean and clear, but not too swift) into "C" code (complicated and talkative, but agile).

Jython

Another version of Python is called **Jython**.

"J" is for "Java". Imagine a Python written in Java instead of C. This is useful, for example, if you develop large and complex systems written entirely in Java and want to add some Python flexibility to them. The traditional CPython may be difficult to integrate into such an environment, as C and Java live in completely different worlds and don't share many common ideas.

Jython can communicate with existing Java infrastructure more effectively. This is why some projects find it usable and needful.

Note: the current Jython implementation follows Python 2 standards. There is no Jython conforming to Python 3, so far.

PyPy and RPython

Take a look at the logo below. It's a rebus. Can you solve it?

It's a logo of the **PyPy** - a Python within a Python. In other words, it represents a Python environment written in Python-like language named **RPython** (Restricted Python). It is actually a subset of Python. The source code of PyPy is not run in the interpretation manner, but is instead translated into the C programming language and then executed separately.

This is useful because if you want to test any new feature that may be (but doesn't have to be) introduced into mainstream Python implementation, it's easier to check it with PyPy than with CPython. This is why PyPy is rather a tool for people developing Python than for the rest of the users.

This doesn't make PyPy any less important or less serious than CPython, of course.

In addition, PyPy is compatible with the Python 3 language.

There are many more different Pythons in the world. You'll find them if you look, but **this course will focus on CPython**.