01. Introduction

September 30, 2025

1 1. Introduction

2 1.1 Overview of Python and its applications

2.1 History

Python was created by Guido van Rossum and first published in 1991. He started working on Python in the search of a scripting language that is easy to learn and use, yet powerful and expressive. Van Rossum shouldered sole responsibility for the project, as the lead developer, until 12 July 2018, when he announced his "permanent vacation" from his responsibilities as Python's "benevolent dictator for life", a title the Python community bestowed upon him to reflect his long-term commitment as the project's chief decision-maker. In January 2019, active Python core developers elected a five-member Steering Council to lead the project.

In this time, several major Python versions were released. Python 2.0 was released on 16 October 2000, with many major new features such as list comprehensions, cycle-detecting garbage collection, reference counting, and Unicode support. We commonly refer to 2.x versions of Python as Python 2. Python 2 gained much popularity and was extensively used accross various domains, but Guido van Rossum started questioning some of his early decisions. This is why, he began working on a new, non-backwards-compatible version of Python: Python 3. Python 3 was designed to rectify fundamental design flaws in the language – the changes required could not be implemented while retaining full backwards compatibility with the 2.x series, which necessitated a new major version number. The guiding principle of Python 3 was: "reduce feature duplication by removing old ways of doing things".

Python 3.0 was released on December 3, 2008, and it would take almost 12 years for it to replace Python 2. The end-of-life of Python 2 was January 1, 2020, date after which no further updates are to be made to Python 2. This big period of time between the launch of Python 3 and the demise of Python 2 was needed for companies and individuals to port their code between the two Python versions.

As of October 2023, Python 3.12 is the stable release, and 3.12 and 3.11 are the only versions with active (as opposed to just security) support. Notable changes in 3.11 from 3.10 include increased program execution speed and improved error reporting. Python 3.11 claims to be between 10 and 60% faster than Python 3.10, and Python 3.12 adds another 5% on top of that. It also has improved error messages, and many other changes.

2.2 Why Python

What makes Python so popular and why it's a good idea to learn it and use it in your projects?

2.2.1 1. Easy to learn and use, even for non-programmers

Python is a beginner-friendly programming language that is effortless to learn and use, even for those without experience. Many find it handy for its simplified English-like syntax, ensuring an easy learning curve for everyone.

Python coding is easy to write and you can implement programs faster than other leading programming. In fact, it is designed to be a general-purpose language, and it is used in many schools and universities for introductory courses in programming.

The basics usually take newbies two to six months to grasp, but once you do, you can construct a simple program in a matter of minutes.

2.2.2 2. Suitable for developing a broad range of applications

Python is also known for its versatility, allowing you to use it for numerous tasks. Key domains where it shines:

Web Development Python is used to build web applications and websites using popular frameworks like Django and Flask. These frameworks simplify the development of web applications, making it easier to handle tasks such as routing, authentication and database interactions.

Data Analysis and Visualization Python is widely used in data analysis and visualization. Libraries like Pandas, NumPy and Matplotlib provide powerful tools for data manipulation, analysis, and the creation of informative charts and graphs. Python is a go-to language for data scientists and analysts.

Machine Learning and Artificial Intelligence Python has become the de facto language for machine learning and artificial intelligence. Libraries like TensorFlow, Keras, PyTorch and scikit-learn make it easy to build and train machine learning models for tasks like image recognition, natural language processing and predictive analytics.

Scientific Computing Python is used in various scientific disciplines for simulations, data analysis and visualization. Libraries like SciPy, SymPy and matplotlib are essential tools for researchers and scientists in fields such as physics, biology, chemistry and various types of engineering.

Automation and Scripting Python's simplicity and readability make it an excellent choice for automation and scripting tasks. It can be used to automate repetitive tasks, manage files and directories, as well as to interact with operating system functions.

Game Development Python is used in game development, primarily for creating 2D games. Libraries like Pygame provide game developers with the tools needed to build interactive games and simulations.

Web Scraping and Data Extraction Python is widely used for web scraping and data extraction. Developers use libraries like BeautifulSoup and Scrapy to extract data from websites, which can be used for various purposes, including data analysis, market research and content aggregation.

Software Testing And Prototyping Software developers can manage software testing easier via Python automation tools like Requestium and Green. It helps handle control building, bug tracking, defect reporting, and testing.

2.2.3 3. Matured Community

There are over 10 million Python developers in the world (Python comes in second, after Javascript, in the top of the most popular programming languages). That means you're not alone in the Python-coding journey. There's a robust community at your back to support you throughout the adventure.

You learn faster when you surround yourself with others who know Python instead of learning it by yourself. The mature and supportive community can give you more encouragement and motivation to help you achieve your goal faster.

Additionally, most of them are willing to share their helpful experience, solutions, tips, and tricks when you post your Python-related issues on the community forums and blogs.

2.2.4 4. Libraries And Frameworks

Python is free, and you can always access a vast ecosystem of open source resources, packages, and libraries. There are thousands of useful libraries to make your coding journey smoother and easier.

No developer can go far with basic programming language knowledge and a code editor. The purpose of programming is not to build everything from ground zero; instead, you develop on top of pre-developed components, which are libraries and frameworks, in this case.

3 1.2 Setting up the development environment

In order to start coding in Python, you need to have **Python installed on your computer** and to have a **code editor/IDE** suitable for Python development.

3.1 Installing Python

Python works on any operating system and the most straightforward method to install it is platform independent. Go to https://www.python.org/downloads/ and choose the appropriate installer for your operating system (the website should detect the operating system automatically, but make sure the correct option is selected).

Then, run the installer. You can go with the default options, but checking the `Add python.exe to PATH` option is a good idea, because you will be able to use Python from the command line easily.

3.2 Choosing and installing an IDE (Integrated Development Environment)

Any code editor (or text editor, as a matter of fact) works for writing Python code. However, an IDE is a better idea, because you will have an *integrated environment* where you can write, run and debug your programs. IDEs will tipically also include integrations with other useful tools, like **git** (version control system, for keeping track of the changes in your code and working collaboratively with a team), **pip** (package manager for Python 3rd party packages), **venv** (a solution for creating isolated Python environments) and others.

The two most popular IDEs for Python development at the moment are **PyCharm** and **Visual Studio Code**. Let's dive into the pros and cons for each of these two IDEs (this comparison can also be generalised as a comparison between IDEs built specifically for Python and general purpose IDEs).

3.2.1 PyCharm

PyCharm is a fully-featured, out of the box IDE developed specifically for Python. This means that after installing, you will be ready to dive into Python coding, without any extra configuration.

Pros: - beginner-friendly - all Python-specific tools readily available (virtual environments, pip, run configurations, debugging, interactive console) - the Community Edition is free and is enough for most projects

Cons: - it uses more memory than VSCode, which may make it slower on certain computers

To install PyCharm, go to the download page, scroll down to the bottom of the page to **PyCharm Community Edition** and download the appropriate installer for your operating system.

Run the installer, open PyCharm, select *Create a new project*, choose a name for your project (e.g. `beginner-course`), press *Create*.

3.2.2 Visual Studio Code

Visual Studio code is an IDE suitable for development in any programming language.

Pros: - more lightweight - if you're already using it for development in other languages it's probably a good idea to also use it for Python

Cons: - Python extensions need to be installed and configured

A very good tutorial on how to use VSCode for Python development ca be found in the official VSCode documentation.

4 1.3 Running Python scripts. The Python shell

The Python interpreter is the program you'll need to run Python code and scripts. Technically, the interpreter is a layer of software that works between your program and your computer hardware to get your code running. After installing Python, you'll be able to use the interpreter to run Python code.

4.1 Source File

Python source files use the extension `.py`. In order to run a Python program, you will actually run the Python interpreter with the Python source file name as an argument. These are system commands and should be run in a command-line interpreter - e.g. **cmd** (Windows), **Terminal** (macOS, Ubuntu).

```
/usr/local/bin/python3 /path/to/script.py
```

Usually, the Python interpreter is linked to a shorter command name, like `py`, `python` or `python3`. You don't have to provide the absolute path to the source file if the file is at the current working location. Usually, you'll run a Python file like this:

```
python script.py
```

Inside the `script.py` file you should include valid Python instructions:

```
print('Hello world!')
```

Sometimes, you'll notice the first line in a Python file looking like something like this `#!/usr/bin/env python3`. This is called a *shebang* and it's used to specify the path to the executable for that script. The shebang line in any script determines the script's ability to be executed like a standalone executable. So, if you place the following lines inside `script.py` and you make the file executable (`chmod +x script.py` on Unix-like systems):

```
#!/usr/local/bin/python3
print('Hello world!')
```

you'll be able to directly run the file in the command-line, like this:

```
./script.py
```

4.2 Exercises 1.3.1

1. Let's create our first Python program. Create a Python file `first_script.py` in your PyCharm project (or add the file in VSCode, if that's your IDE of choice). Write the following lines inside the file:

   ```
   name = "world"
   print("Hello " + name + "!")
   ```

2. Run the file from inside the IDE. Take a look at the command that has been run and the output.
3. Change the value of `name` to your name.
4. Run the program again and notice the change.
5. Copy the command that the IDE runs when you press the *Run* button and paste it in the terminal window of the IDE and press Enter. What do you notice?
6. Now, open the *cmd* / *Terminal* app on your computer, paste the same command here and press Enter. What happens?

4.3 The Python shell

The Python shell or the interactive interpreter is a very powerful tool allowing developers to inspect the code and experiment with small blocks of code before moving it to a file.

The way you can start the Python shell depends on the operating system you're working on and on the settings you chose when you installed Python. It comes down to the way you can reference the Python interpreter, just like we did when executing Python programs from source file. So, the main options here are: - using the absolute path of the Python interpreter (this may be something like `/usr/local/bin/python3.12` or `C:\Users\YourUser\AppData\Local\Programs\Python\Python312`) - using the alias created on installation (this may be `py`, `python`, `python3`, `python3.12`, etc)

So, enter the full path or the alias in a command-line interpreter - e.g. **cmd** (Windows), **Terminal** (macOS, Ubuntu), press Enter, and you're all set! Read more about the Python Interpreter in the official documentation.

```
Python 3.7.5 (default, Nov  1 2019, 02:16:32)
[Clang 11.0.0 (clang-1100.0.33.8)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> print("hello world")
hello world
>>>
```

4.4 Exercises 1.3.2

1. Now, let's start an interactive Python interpreter. There are several options to do that, so let's try them all:

    - inside the IDE: locate the Python Console tab in PyCharm / Python Interactive window in VSCode
    - the IDLE app (Integrated Development and Learning Environment) that came with the Python installation
    - invoking the Python interpreter in a command line interpreter (like *cmd*, *PowerShell* or *Terminal*)

2. Run the same lines as above, this time in the interactive interpreter. Make sure to press Enter after each line:

    ```
    name = "world"
    print("Hello " + name + "!")
    ```

3. Another thing you can do in the Python shell is inspecting the value of a variable. Type `name` and Enter in the interactive interpreter.

4. Now, let's change the value of a previously defined variable:

    ```
    name = "your name"
    ```

5. Did the command above have any effect? In order to see that the variable really changed its value, you will have to either re-inspect its value (`name`) or print its value (`print(name)`), or print a more complex message that contains it (`print("Hello " + name + "!")`)