HyperionDev

Your First Computer Program

Task

Visit our website

Introduction

In this lesson, we will start with the basics of programming using Python, one of the most popular and accessible programming languages today. Python is used by developers worldwide for various applications, from simple scripts to complex machine-learning algorithms. Its simple syntax and readability make it an ideal language for beginners.

This lesson will guide you through setting up your programming environment, writing your first Python scripts, and running them to see the immediate results of your code.

What is programming?

Programmers write code statements to create programs. Programs are executable files that perform the instructions given by the programmer.

Code can be written in different programming languages, such as Python, Java, JavaScript, and C++. In this course, you will start by learning Python.

Each coding language has a different file type. For example, after writing Python code you need to save it in a Python file. A Python file has the following file naming format:

filename.**py**

The filename can be any valid filename and .**py** is the file extension.

You can then "run" the Python file. In this process, the Python program you have written is executed and displays the outcomes that may result based on what the code statements say.

There is more information about how to run Python files in the example file (**example_first_program.py**) that accompanies this task. But **before we get there**, we need to talk about why we're using Python, show you how to write basic code in Python, and perform some basic operations.

Why Python?

Python is a widely used programming language. It is consistently ranked in the top 10 most popular programming languages as measured by the **TIOBE programming community index**. Many familiar organisations make use of Python, such as Wikipedia, Google, Yahoo!, the National Aeronautics and Space Administration (NASA), and Reddit (which is written entirely in Python).

Python is a high-level programming language, similar to other popular languages such as Java, C#, and Ruby. High-level programming languages are closer to human languages than machine code. They're called "high level" as they are several steps removed from the actual code that runs on a computer's processor.

Python is hot!

The demand for Python programmers is only growing. Python boasts the highest year-on-year increase in terms of demand by employers (as reflected in job descriptions online) as well as popularity among developers. Python developers are also one of the highest-paid categories of programmers! The demand for Python is only set to grow further with its extensive use in analytics, data science, and machine learning.

In comparison with Java, Python is known for its simplicity and ease of use. Python's syntax is clear and concise, making it easier to learn and use, especially for beginners. Here are a few more reasons to use Python:

- **Simple, yet powerful:** Python's design focuses on readability and ease of use. Unlike more complex languages like C++ or Java, Python's clean and straightforward syntax helps developers write and maintain code faster, making it ideal for both small tasks and large projects.
- **From child's play to big business:** Python's simplicity does not limit its power. It is used by some of the biggest tech firms such as Google, Instagram, Spotify, and Dropbox. This shows Python's wide adoption in both small-scale projects and large, complex systems. Its versatility ensures job opportunities for Python developers in many industries.
- **Python is on the web:** Python is widely used in web development, especially with frameworks like Django. Popular websites such as Instagram and Pinterest rely on Django to manage their platforms efficiently. This demonstrates Python's ability to support large-scale web applications with flexibility and speed.
- **Dropbox used Python extensively:** Python was a key component in Dropbox's early development, playing a major role in building its desktop client and server. As Dropbox grew, it began using other languages like Go and JavaScript to manage performance and scalability, but Python remains an important part of its tech stack!

How do you write code?

Before you get started, we suggest you install Visual Studio Code (VS Code) and use it as your integrated development environment (IDE) to open all text files (.txt) and Python files (.py).

Setting up your development environment

Please use the instructions in the additional reading entitled **Installation, Sharing, and Collaboration** to install Python and Visual Studio Code. The guide will also provide guidance and tools for you to install additional software as we progress through the bootcamp.

Please note: It is crucial to consult this reading and set up your development environment before beginning any practical coding components.

Now that you have your development environment set up, you are ready to start coding!

Learning the rules

All programming languages have syntax rules. Syntax is the "spelling and grammar setup" of a programming language and determines how you write correct, well-formed code statements. If you make a mistake by breaking the “spelling and grammar” rules for coding in Python, this is called a **syntax error**.

Consider the following Python code. We’ll get into more detail about what it means later, but for now, it’s enough to know that this code will output "Hello, World!".

```
print("Hello, World!")
```

A common syntax error you could make in this code would be forgetting to add a closing quotation mark (") at the end of the “`Hello, World!`”. The text inside the brackets is a **string**. With strings, all opening quotation marks (") require a closing one – the opening one shows the string is starting, and the closing one shows where it ends.

Another common syntax error that you could make in the example above is forgetting to add a closing bracket “`)`”. Remember that all opening brackets “`(`“ require a matching closing one, “`)`”!

Any program you write must be exactly correct. All code is case-sensitive. This means that “`Print`” is not the same as “`print`”. If you enter an invalid Python command, misspell a command, or misplace a punctuation mark, you will get a syntax error when trying to run your Python program.

Errors appear in the Python shell when you try to run a program and it fails. Be sure to read all errors carefully to discover what the problem is. Error reports in the Python shell will even tell you what line of your program had an error. The process of resolving errors in code is known as debugging.

Creating output

You may want your program to display or output information to the user. The most common way to view program output is to use the print function. To use print, we enter the print command followed by one or more arguments in brackets. Let's first take a moment to understand what arguments are, as well as two other new terms, parameters and variables.

A **variable** is a computer programming term that is used to refer to the storage locations for data in a program. Each variable has a name that can be used to refer to some stored information known as a **value**.

Functions in Python are blocks of code that perform one or more specific tasks. A **parameter** is a variable in a function definition that tells you the sort of data you will need to give the function to work with when it runs. When a function is actually called (in other words, the programmer instructs the block of code making up the function to run), the **arguments** are the data you pass into the function's parameters (i.e., the actual data you give to the function to work on when it runs).

Now that you understand these basic concepts, let's review how print works. You've seen the following code before:

```
# This code will output "Hello, World!"

print("Hello, World!")
```

We enter the print command into a Python file, followed by one or more arguments in brackets. In programming, a **command** is an instruction given by a user telling a computer to do something. Together, a command and an argument are known as a **statement**.

```
# This code will output "Hello, World!"

print("Hello, World!")
```

When you run the above program, the computer will output the string "`Hello, World!`" that was passed into the input parameter. Note that the argument is enclosed in double quotes ("…"). This is because "`Hello, World!`" is a type of variable called a string, i.e., a list of characters. You'll learn more about strings and other variable types later.

Note that the Python shell (the window that is displayed when you run a Python program) only shows the output of the program. Other statements in your code that don't create output will be executed but not displayed in the Python shell.

Getting input

Sometimes you want a user to enter data through the keyboard that will be used by your program. To do this, you use the input command.

When the program runs, the input command, which can be seen in the example below, will show the text "`Enter your name:` " in the output box of the program. The program will then halt until the user enters something with their keyboard and presses enter.

```
# Prompt the user to input their name

name = input("Enter your name: ")

# Store the user's input in the “name” variable
```

The variable name stores what the user entered into the box as a string. Storing and declaring variables doesn't produce any output.

Documenting code

Did you notice the code above included some text that describes what the code does? This line is a **comment**.

Comments are useful because they help explain what the code does, why certain choices were made, and they can suggest what needs to be improved. This is especially helpful when working with others or when you need to understand your own code after some time. Essentially, comments make it easier for anyone (including your future self) to understand and work with the code, making them a very important part of programming.

In Python, comments are used to write notes right inside the code. To make comments in Python, you start the line with the `#` symbol. This tells Python to ignore that line when running the program.

Python provides two kinds of comments: block and inline comments.

```
# This is a block comment,
# it is used to describe a logical block of code
greeting = "Hello, World!"
print(greeting)
```

```
greeting = "Hello, World!"
print(greeting)  # This is an inline comment, it describes a line of code
```

Note that it's not necessary to describe each line of code with an inline comment. These are reserved for highlighting a specific line of code where appropriate.

Here is a list of functions comments perform to document information about the code:

1. **Explaining the code:** for a particularly tricky piece of code, this is intended to help someone reading it follow your logic and understand the process.
2. **Summarising the code:** this is a simple summary of the overall process of the code.
3. **Describing what the code is meant to be doing:** this indicates to the reader that a piece of the code is not doing what it should be, and possibly needs to be reworked.
4. **Code markers:** these are used to show where code still needs to be completed, and is usually indicated by a string of ##########, **********, or !!!!!!!!!!. Code markers can also be used to separate logical sections of code.
5. **Commented out code for debugging:** this is code that is commented out until it is time to debug. For instance, where a variable is usually assigned a value from user input, that line could be commented out and the value could be hard-coded for the sake of debugging. Once the debugging process is completed, these comments will usually be deleted. See below:

```
# number = int(input("Please enter a number"))
number = 6
print(number)
```

6. **Commented out code that is not meant to be run (at this time):** if a programmer is working on a piece of code, they may be trying different approaches to find the most efficient or understandable way forward. At this time, there might be pieces of code that are commented out. Once they have decided on the best way, the unused code will be deleted.
7. **Information that needs to be in the code, but cannot be expressed by the code itself:** these include comments about copyright, confidentiality, etc.

Extra resource

Read the **PEP8 style guide section on comments** for commenting best practices.

What's next?

This lesson is continued in the example file (**example_first_program.py**) provided in this task folder. Open this file using VS Code. The context and explanations provided in the examples should help you better understand some simple basics of Python.

You may run the examples to see the output. The instructions on how to do this are inside each example file. Feel free to write and run your own example code before attempting the task to become more comfortable with Python.

Try to write comments in your code to explain what you're doing in your program (read the example files for more information, and to see examples of how to write comments).

You are not required to read the entirety of the **additional reading**. It is purely for extra reference. That said, do take a look – you could find it very useful!

Now it's time to try your hand at a practical task.

Take note

The task below is **auto-graded**. An auto-graded task still counts towards your progression and graduation. Give it your best attempt and submit it when you're ready.

When you select "Request review", the task is automatically complete, you do not need to wait for it to be reviewed by a mentor.

You will then receive an email with a link to a model answer, as well as an overview of the approach taken to reach this answer.

Take some time to review and compare your work against the model answer. This exercise will help solidify your understanding and provide an opportunity for reflection on how to apply these concepts in future projects.

In the same email, you'll also receive a link to a survey, which you can use to self-assess your submission.

Once you've done that, feel free to progress to the next task.

Auto-graded task

Follow these steps:

- Create a new Python file in the folder for this task and call it **hello_world.py**.
- First, provide pseudocode as comments in your Python file, outlining how you will solve this problem (you'll need to read the rest of this practical task first, of course!).
- Now, inside your **hello_world.py** file, write Python code to take in a user's name using the `input()` function and then print out the name.
- Use the same `input()` function and output approach to take in a user's age and print it out.
- Finally, print the string "`Hello, World!`" on a new line (the new line will happen by default if you use a separate print statement to the one you used immediately above to print out the age, because each print statement automatically inserts an "enter", or newline instruction, at the end).

Important: Be sure to upload all files required for the task submission inside your task folder and then click "Request review" on your dashboard.

Take note

Make sure that you have installed and set up Python and your editor correctly.

If you are not using Windows, please ask one of our mentors for alternative instructions.

Share your thoughts

Please take some time to complete this short feedback **form** to help us ensure we provide you with the best possible learning experience.