

Chapter 1

Introduction



1.1 What Is Python?

Python is a general-purpose programming language in a similar vein to other programming languages that you might have heard of such as C++, JavaScript or Microsoft's C# and Oracle's Java.

It has been around for some considerable time having been originally conceived back in the 1980s by Guido van Rossum at Centrum Wiskunde & Informatica (CWI) in the Netherlands. The language is named after one of Guido's favourite programs "Monty Pythons Flying Circus", a classic and somewhat anarchic British comedy sketch show originally running from 1969 to 1974 (but which has been rerun on various stations ever since) and with several film spin offs. You will even find various references to this show in the documentation available with Python.

As a language it has gained in interest over recent years, particularly within the commercial world, with many people wanting to learn the language. This increased interest in Python is driven by several different factors:

1. Its flexibility and simplicity which makes it easy to learn.
2. Its use by the Data Science community where it provides a more standard programming language than some rivals such as R.
3. Its suitability as a scripting language for those working in the DevOps field where it provides a higher level of abstraction than alternative languages traditionally used.
4. Its ability to run on (almost) any operating system, but particularly the big three operating systems Windows, MacOS and Linux.
5. The availability of a wide range of libraries (modules) that can be used to extend the basic features of the language.
6. It is free!

Python itself is now managed by the not-for-profit Python Software Foundation (see https://en.wikipedia.org/wiki/Python_Software_Foundation) which was launched in March 2001. The mission of the foundation is to foster development of the Python community; it is also responsible for various processes within the Python community, including developing the core Python distribution, managing intellectual rights and supporting developer conferences including PyCon.

1.2 Python Versions

Currently there are two main versions of Python called Python 2 and Python 3.

- Python 2 was launched in October 2000 and has been, and still is, very widely used.
- Python 3 was launched in December 2008 and is a major revision to the language that is not backward compatible.

The issue between the two versions can be highlighted by the simple print facility:

- In Python 2 this is written as `print 'Hello World'`
- In Python 3 this is written as `print ('Hello World')`

It may not look like much of a difference but the inclusion of the `' () '` marks a major change and means that any code written for one version of Python will probably not run on the other version. There are tools available, such as the 2–3 utility, that will (partially) automate translation from Python 2 to Python 3 but in general you are still left with significant work to do.

This then raises the question which version to use?

Although interest in Python 3 is steadily increasing there are many organisations that are still using Python 2. Choosing which version to use is a constant concern for many companies.

However, the Python 2 end of life plan was initially announced back in 2015 and although it has been postponed to 2020 out of concern that a large body of existing code could not easily be forward-ported to Python 3, it is still living on borrowed time. Python 3 is the future of the Python language and it is this version that has introduced many of the new and improved language and library features (that have admittedly been back ported to Python 2 in many cases). This book is solely focussed on Python 3.

In the remainder of this book when we refer to Python we will always be referring to Python 3.

1.3 Python Programming

There are several different programming paradigms that a programming language may allow developers to code in, these are:

- **Procedural Programming** in which a program is represented as a sequence of instructions that tell the computer what it should do explicitly. Procedures and/or functions are used to provide structure to the program; with control structures such as *if statements* and *loop constructs* to manage which steps are executed and how many times. Languages typifying this approach include C and Pascal.
- **Declarative Programming** languages, such as Prolog, that allow developers to describe how a problem should be solved, with the language/environment determining how the solution should be implemented. SQL (a database query language) is one of the most common declarative languages that you are likely to encounter.
- **Object Oriented Programming** approaches that represent a system in terms of the objects that form that system. Each object can hold its own data (also known as state) as well as define behaviour that defines what the object can do. A computer program is formed from a set of these objects co-operating together. Languages such as Java and C# typify the object oriented approach.
- **Functional Programming** languages decompose a problem into a set of functions. Each function is independent of any external state, operating only on the inputs they received to generate their outputs. The programming language Haskell is an example of a functional programming language.

Some programming languages are considered to be *hybrid* languages; that is they allow developers to utilise a combination of different approaches within the same program. Python is an example of a hybrid programming language as it allows you to write very procedural code, to use objects in an object oriented manner and to write functional programs. Each of these approaches is covered in this book.

1.4 Python Libraries

As well as the core language, there are very many libraries available for Python. These libraries extend the functionality of the language and make it much easier to develop applications. These libraries cover

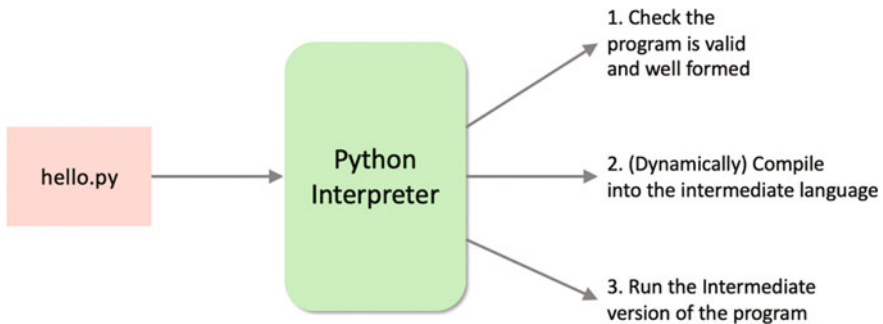
- web frameworks such as Django/Flask,
- email clients such as smtplib (a SMTP email client) and imaplib (an IMAP4 email client),
- content management operations such as the Zope library,
- lightweight concurrency (running multiple operations at the same time) using the Stackless library,
- the Generation of Microsoft Excel files using the Python Excel library,
- graphics libraries such as Matplotlib and PyOpenGL,
- machine learning using libraries such as SKLearn and TensorFlow.

A very useful resource to look at, which introduces many of these libraries (also known as modules), is the ‘Python 3 module of the Week’ web site which can be found at <https://pymotw.com/3>. This lists many of the libraries/modules available and provides a short introduction to what they do and how to use them.

1.5 Python Execution Model

Python is not a precompiled language in the way that some other languages you may have come across are (such as C++). Instead it is what is known as an interpreted language (although even this is not quite accurate). An interpreted language is one that does not require a separate compilation phase to convert the human readable format into something that can be executed by a computer. Instead the plain text version is fed into another program (generally referred to as the interpreter) which then executes the program for you.

Python actually uses an intermediate model in that it actually converts the plain text English style Python program into an intermediate 'pseudo' machine code format and it is this intermediate format that is executed. This is illustrated below:



The way in which the Python interpreter processes a Python program is broken down into several steps. The steps shown here are illustrative (and simplified) but the general idea is correct.

1. First the program is checked to make sure that it is valid Python. That is a check is made that the program follows all the rules of the language and that each of the commands and operations etc. is understood by the Python environment.
2. It then translates the plain text, English like commands, into a more concise intermediate format that is easier to execute on a computer. Python can store this intermediate version in a file which is named after the original file but with a `.pyc` extension instead of a `.py` extension (the 'c' in the extension indicates it contains the compiled version of the code).
3. The compiled intermediate version is then executed by the interpreter.

When this program is rerun, the Python interpreter checks to see if a `.pyc` file is present. If no changes have been made to the source file since the `.pyc` was

created, then the interpreter can skip steps 1 and 2 and immediately run the '.pyc' version of the program.

One interesting aspect of Python's usage is that it can be (and often is) used in an interactive fashion (via the REPL), with individual commands being entered and executed one at a time, with context information being built up. This can be useful in debugging situations.

1.6 Running Python Programs

There are several ways in which you can run a Python program, including

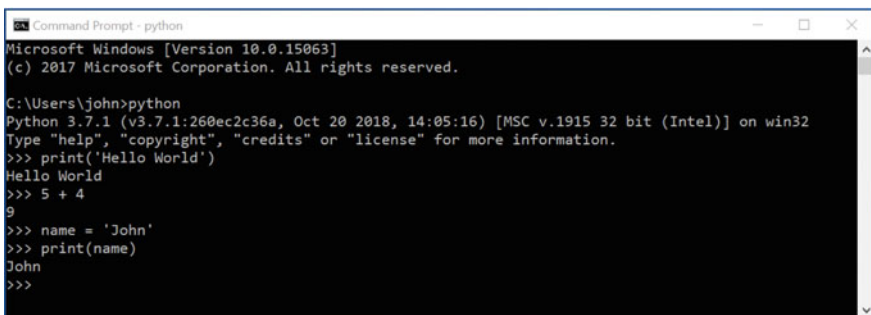
- Interactively using the Python interpreter
- Stored in a file and run using the Python command
- Run as a script file specifying the Python interpreter to use within the script file
- From within a Python IDE (Integrated Development Environment) such as PyCharm.

1.6.1 *Interactively Using the Python Interpreter*

It is quite common to find that people will use Python in interactive mode. This uses the Python REPL (named after **Read Evaluate Print Loop** style of operation).

Using the REPL, Python statements and expressions can be typed into the Python prompt and will then be executed directly. The values of variables will be remembered and may be used later in the session.

To run the Python REPL, Python must have been installed onto the computer system you are using. Once installed you can open a Command Prompt window (Windows) or a Terminal window (Mac) and type `python` into the prompt. This is shown for a Windows machine below:



```
Command Prompt - python
Microsoft Windows [Version 10.0.15063]
(c) 2017 Microsoft Corporation. All rights reserved.

C:\Users\john>python
Python 3.7.1 (v3.7.1:260ec2c36a, Oct 20 2018, 14:05:16) [MSC v.1915 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> print('Hello World')
Hello World
>>> 5 + 4
9
>>> name = 'John'
>>> print(name)
John
>>>
```

In the above example, we interactively typed in several Python commands and the Python interpreter 'Read' what we have typed in, 'Evaluated' it (worked out what it should do), 'Printed' the result and then 'Looped' back ready for further input. In this case we

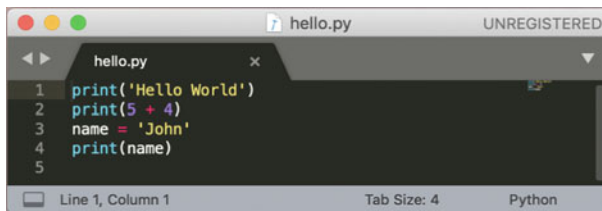
- Printed out the string 'Hello World'.
- Added 5 and 4 together and got the result 9.
- Stored the string 'John' in a variable called `name`.
- Printed out the contents of the variable `name`.

To leave the interactive shell (the REPL) and go back to the console (the system shell), press Ctrl-Z and then Enter on Windows, or Ctrl-D on OS X or Linux. Alternatively, you could also run the Python command `exit()` or `quit()`.

1.6.2 Running a Python File

We can of course store the Python commands into a file. This creates a program file that can then be run as an argument to the `python` command.

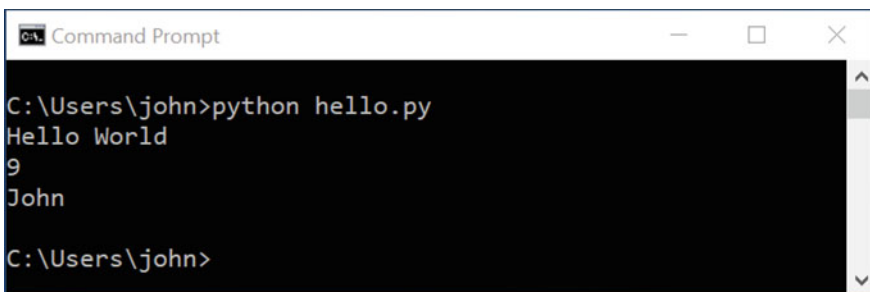
For example, given a file containing the following file (called `hello.py`) with the 4 commands in it:

A screenshot of a code editor window titled 'hello.py' with 'UNREGISTERED' in the top right corner. The editor shows five lines of Python code:

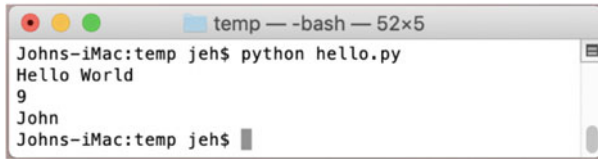
```
1 print('Hello World')
2 print(5 + 4)
3 name = 'John'
4 print(name)
5
```

 The status bar at the bottom indicates 'Line 1, Column 1', 'Tab Size: 4', and 'Python'.

To run the `hello.py` program on a PC using Windows we can use the `python` command followed by the name of the file:

A screenshot of a Windows Command Prompt window. The prompt shows the command `C:\Users\john>python hello.py` being executed. The output of the program is displayed on the next three lines: `Hello World`, `9`, and `John`. The prompt then returns to `C:\Users\john>`.

We can also run the same program on an Apple Mac using MacOS via the `python` interpreter. For example on a Mac we can do the following:

A screenshot of a terminal window titled "temp --bash-- 52x5". The prompt is "Johns-iMac:temp jeh\$". The user has entered "python hello.py". The output is "Hello World", "9", and "John". The prompt is now "Johns-iMac:temp jeh\$".

```
temp --bash-- 52x5
Johns-iMac:temp jeh$ python hello.py
Hello World
9
John
Johns-iMac:temp jeh$
```

This makes it very easy to create Python programs that can be stored in files and run when needed on whatever platform is required (Windows, Linux or Mac). This illustrates the cross platform nature of Python and is just one of the reasons why Python is so popular.

1.6.3 Executing a Python Script

It is also possible to transform a file containing a stored Python program into a Script. A script is a stand-alone file that can be run directly without the need to (explicitly) use the `python` command.

This is done by adding a special line to the start of the Python file that indicates the Python command (or interpreter) to use with the rest of the file. This line must start with `'#!'` and must come at the start of the file.

To convert the previous section's file into a Script we would need to add the path to the `python` interpreter. Here *path* refers to the route that the computer must take to find the specified Python interpreter (or executable).

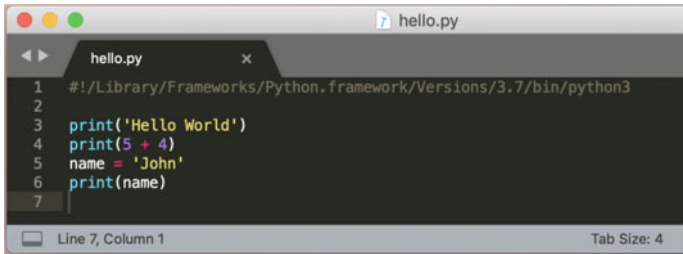
The exact location of the Python interpreter on your computer depends on what options were selected when you (or whoever installed Python) set it up. Typically on a Windows PC Python will be found in the 'Program Files' directory or it might be installed in its own 'Python' directory.

Whatever the location of the Python interpreter, to create a script we will need to add a first line to our `hello.py` file. This line must start with a `#!`. This combination of characters is known as a shebang and indicates to Linux and other Unix like operating systems (such as MacOS) how the remainder of the file should be executed.

For example, on a Apple Mac we might add:

```
/Library/Frameworks/Python.framework/Versions/3.7/bin/python3
```

When added to the `hello.py` file we now have:

A screenshot of a code editor window titled 'hello.py'. The code inside is as follows:

```
1 #!/Library/Frameworks/Python.framework/Versions/3.7/bin/python3
2
3 print('Hello World')
4 print(5 + 4)
5 name = 'John'
6 print(name)
7
```

The editor shows line numbers 1 through 7 on the left. At the bottom, it indicates 'Line 7, Column 1' and 'Tab Size: 4'.

However, we cannot just run the file as it stands. If we tried to run the file without any changes then we will get an error indicating that the permission to execute the file has been denied:

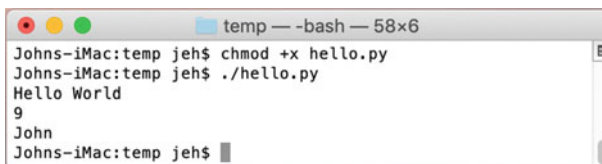
```
$ ./hello.py
-bash: ./hello.py: Permission denied
$
```

This is because by default you can't just run a file. We need to mark it as executable. There are several ways to do this, however one of the easiest on a Mac or Linux box is to use the `chmod` command (which can be used to modify the permissions associated with the file). To make the file executable we can change the file permissions to allow the file to be run by using the following command from a terminal window when we are in the same directory as the `hello.py` file:

```
$ chmod +x hello.py
```

Where `+x` indicates that we want to add the executable permission to the file.

Now if we try to run the file directly it executes and the results of the commands within the file are printed out:

A screenshot of a terminal window titled 'temp — -bash — 58x6'. The terminal shows the following commands and output:

```
Johns-iMac:temp jeh$ chmod +x hello.py
Johns-iMac:temp jeh$ ./hello.py
Hello World
9
John
Johns-iMac:temp jeh$
```

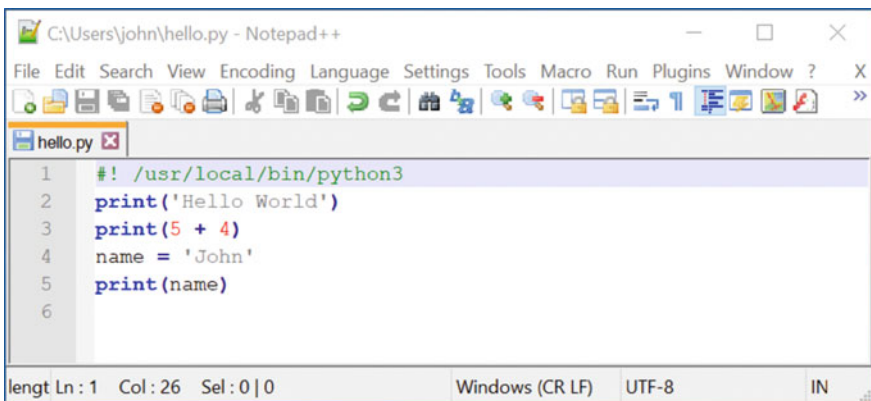
Note the use of the `./` preceding the file name in the above; this is used on Linux and Macs to tell the operating system to look in the current directory for the file to execute.

Different systems will store Python in different locations and thus might need different first lines, for example on a Linux we might write:


```
#!/usr/local/bin/python3
print('Hello, world')
print(5 + 4)
name = 'John'
print(name)
```

By default Windows does not have the same concept. However, to promote cross platform portability, the Python Launcher for Windows can also support this style of operation. It allows scripts to indicate a preference for a specific Python version using the same `#!` (Shebang) format as Unix style operating systems. We can now indicate that the rest of the file should be interpreted as a Python script; if multiple versions of Python are installed this may require Python 3 to be explicitly specified. The launcher also understands how to translate the Unix version into Windows versions so that `/user/local/bin/python3` will be interpreted as indicating that python3 is required.

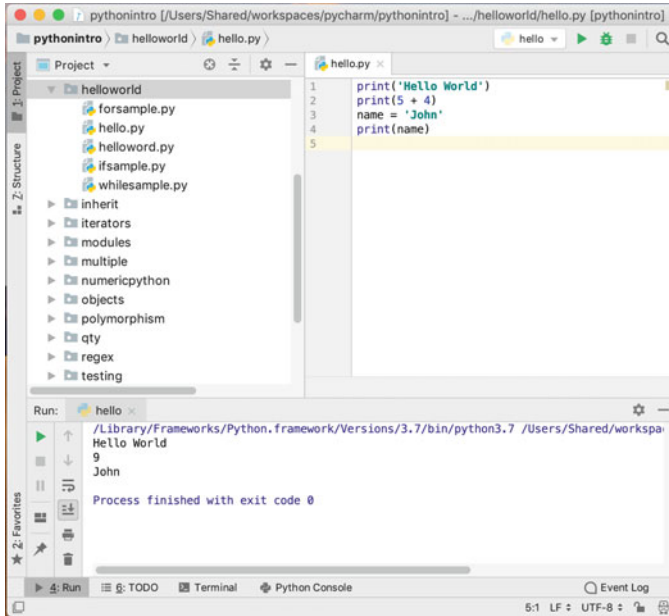
An example of the `hello.py` script for a Windows or Linux machine is given below using Notepad++ on a Windows box.



When the launcher was installed it should have been associated with Python files (i.e. files that have a `.py` extension). This means that if you double-click on one of these files from the Windows Explorer, then the Python launcher will be used to run the file.

1.6.4 Using Python in an IDE

We can also use an IDE such as PyCharm to writing and execute our Python program. The same program is shown using PyCharm below:



In the above figure, the simple set of commands are again listed in a file called `hello.py`. However, the program has been run from within the IDE and the output is shown in an output console at the bottom of the display.

1.7 Useful Resources

There are a wide range of resources on the web for Python; we will highlight a few here that you should bookmark. We will not keep referring to these to avoid repetition but you can refer back to this section whenever you need to:

- https://en.wikipedia.org/wiki/Python_Software_Foundation Python Software Foundation.
- <https://docs.python.org/3/> The main Python 3 documentation site. It contains tutorials, library references, set up and installation guides as well as Python how-tos.
- <https://docs.python.org/3/library/index.html> A list of all the builtin features for the Python language—this is where you can find online documentation for the various class and functions that we will be using throughout this book.
- <https://pymotw.com/3/> the Python 3 Module of the week site. This site contains many, many Python modules with short examples and explanations of what the modules do. A python module is a library of features that build on and expand

the core Python language. For example, if you are interested in building games using Python then pyjama is a module specifically designed to make this easier.

- <https://www.fullstackpython.com/email.html> is a monthly newsletter that focusses on a single Python topic each month, such as a new library or module.
- <http://www.pythonweekly.com/> is a free weekly summary of the latest Python articles, projects, videos and upcoming events.