



PYTHON PROGRAMMING FOR BEGINNERS

Your Personal Guide For Getting Into Programming,
Level Up Your Coding Skills From Scratch And Use
Python Like A Mother Language

PROGRAMMING IN PYTHON

MICHAEL & ERIC SCRATCH

Michael and Eric Scratch

PYTHON PROGRAMMING FOR BEGINNERS

*Your Personal Guide for Getting into Programming, Level Up
Your Coding Skills from Scratch and Use Python Like A Mother
Language*

© 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 information contained within this document, including, but not limited to, errors, omissions, or inaccuracies.

Table of Contents

INTRODUCTION

CHAPTER 1: INTRODUCING PYTHON

WHAT IS PYTHON

WHY LEARN PYTHON

INSTALLING PYTHON

GOOD PROGRAMMING PRACTICES

WRITING COMMENTS IN PYTHON

CHAPTER 2: VARIABLES AND OPERATIONS

DATA TYPES

STRING MANIPULATION AND FORMATTING

TYPE CASTING

CHAPTER 3: DATA STRUCTURES

LISTS

TUPLES

DICTIONARIES

DATA STRUCTURES EXERCISES

CHAPTER 4: INPUTS, PRINTING, AND FORMATTING OUTPUTS

INPUTS

INPUT AND FORMATTING

CHAPTER 5: CONDITIONAL STATEMENTS AND CONTROL FLOW STATEMENT

HANDLING CONDITIONS

CONDITIONAL STATEMENTS

CONTROL FLOW STATEMENTS

BREAK/CONTINUE

PREDEFINED ERRORS

CHAPTER 6: FUNCTIONS AND MODULES

FUNCTIONS

CREATING FUNCTIONS

VARIABLE SCOPE

DEFAULT PARAMETERS

VARIABLE LENGTH ARGUMENTS

IMPORTING MODULES

CREATING MODULES

USEFUL BUILT-IN FUNCTIONS AND METHODS

FUNCTIONS AND IMPORTS EXERCISES

CHAPTER 7: OBJECT-ORIENTED PROGRAMMING: CLASSES AND INSTANCES

[WHAT ARE CLASSES?](#)

[CREATING CLASSES](#)

[CLASS VARIABLES AND INSTANCE VARIABLES](#)

[CLASS METHODS AND STATIC METHODS](#)

CHAPTER 8: OBJECTS-ORIENTED PROGRAMMING: INHERITANCE, CHILD CLASSES, AND SPECIAL METHODS

[INHERITANCE AND MULTIPLE INHERITANCE](#)

[IMPORTING CLASSES](#)

[PYTHON SPECIAL METHODS](#)

[CLASSES AND METHODS EXERCISES](#)

CHAPTER 9: FILES

[OPENING, READING, WRITING TEXT FILES](#)

[BUFFER](#)

[BINARY FILES](#)

[DELETING AND RENAMING](#)

CHAPTER 10: INTERMEDIATE AND ADVANCED CONCEPTS

[LAMBDA FUNCTIONS](#)

[DICTIONARY HANDLING](#)

[THREADING](#)

[PACKAGES AND THE PIP PACKAGE MANAGER](#)

[LIBRARIES](#)

[SQLITE](#)

[JSON DATA](#)

[UNIT TESTING](#)

CONCLUSION

Introduction

The world is IN a state of constant evolution towards more complexity with each passing day. Now, there are machines for almost anything. There are machines that cook, clean, and aid living even. Years ago, this would have been seen as nothing but sorcery. But is it? Of course not. Whatever machine you can think of, be it ones as complex as factory robots or ones as simple as personal computers, each requires some programming to function. This program functions as a brain which directs the system according to the rules of its programming. So, like a matrix, this program is all around us; in our phones, homes, offices, school, you name it. This program, however, comes in different forms. But for this book, we are focusing on Python, one of the most advanced forms of programming languages.

The only reason why computers have become better than humans is that they do tasks effectively without wasting any time. However, we can consider computers as dumb machines. If they are not provided with a solution to the problem beforehand, they can't solve it. Even after twenty years of tremendous research, robotics is not a successful field because it is challenging to make machines that make decisions by themselves.

In the early stages of computer development, people used to use instruction sets to complete tasks. These instructions sets are allowed to use only for scientific and military purposes.

Several multinational companies started to experiment with their employees to create a whole new way to pass instructions to computers. They dreamt of an easy way to make things work out.

After a few years of excellent research, computer scientists introduced the concept of programming to the technological world with the help of programming languages. Programming languages provide a set of defined libraries that can help you to create programs.

Programs are considered an analytical representation of algorithms, and the definite task of creating valid programs that can be understood by a machine is known as programming.

But why Python? Why not any other language? Well, it's an easy guess. Python is arguably the most straightforward programming language to learn because of its close association with the English language in its scripting. So, unlike most other languages that are quite complicated for the average non-programmer, one can understand a few things about Python by merely picking up a book. This ease of readability is no mistake in any sense. As a matter of fact, it is in its very design. Tons of multinational companies like Google, or even NASA, use Python in their programming because it is easier to maintain and reuse. Python has only come this far as a programming language because of its features, which are integral to the future. That is why everyone must have a basic knowledge of it.

So, come along on this fun adventure of learning Python as we take you through the basics of the language.

Have fun!

Chapter 1:

Introducing Python

What Is Python

This is a programming language that is object-oriented and of high level and uses semantics. It is a high level in terms of structures in data and in combination of dynamic typing and binding. This is what makes it attractive to be used for Rapid Application Development and for connecting different elements.

Python, with its simplicity and easy-learning helps in reading the programming language, and that is why it reduces the cost to maintain the program. Python encourages the program modularity and code reuse; this is because it supports different packages and modules. The standard library and the Python interpreter can be found in binary form. It is not necessary to charge all the available platforms and can be distributed freely.

Most programmers love the Python program because it offers great productivity. The edit-test debug is a cycle that is fast and does not need any compilation process. It is easier to debug a Python program; it will not cause any segmentation fault. An exception is raised when an error is discovered by the interpreter. When the exception is not known by the program, the interpreter prints a trace. The debugger, on a level of sourcing, will allow being inspecting any variables. There will be a settling of breakpoints, arbitrary expressions, and stepping on the code at any time. Python is what writes the debugger, an easy and quick debugging method of adding prints on the source and statements.

It is like Perl and Ruby, Python is helped by several imaging programs; users are able to create customized extensions. There are different web applications supporting Python API, like Blender and GIMP.

This information given on Python programming is beneficial for both the newbies and the experienced ones. Most of the experienced programmers can

easily learn and use Python. There is an easier way to install Python, most distributors of UNIX and Linux have the recent Python. That is the reason why most computers come already installed with Python. Before you start using Python, you need to know which IDEs and text editors work best with Python. To get more help and information, you can peruse through introductory books and code samples. The Python idea was discovered in 1980 after the ABC language. When Python 2.0 was introduced, it had features like garbage collection and list comprehensions, which are used in reference cycle collection. When Python 3.0 was released in 2008, it brought about a complete language revision. Python is primarily used for developing software and webs, for mathematics and scripting systems. The latest version of Python is known as Python 3, while Python 2 is still popular. Python was developed to help in reading and similar aspects to different languages like English and with emphasis on mathematics.

A new line is used to complete a Python command, as opposed to other programming languages that normally use semi-colons. It depends on indentation, whitespace, and defining the scope.

Why Learn Python

The neat thing about working with Python is that it has something for everyone to enjoy along the way. There are tons of benefits that come with it, and it really does not matter if you have worked with programming in the past or not. You will still find something to love about Python, and it is something that is easy to work with for all levels of programming. Some of the different reasons why you may want to work with the Python language overall include:

It Has Some Code That Is Maintainable and Readable

While you are writing out some of the applications for the software, you will need to focus on the quality of source code in order to simplify some of the updates and the maintenance. The syntax rules of Python are going to give you a way to express the concepts without having to write out any additional codes. At the same time, Python, unlike some of the other coding languages out there, is going to emphasize the idea of the readability of the code and can allow us to work with keywords in English instead of working with different types of punctuations to do that.

Comes With Many Programming Paradigms

Another benefit that we will see is the multiple programming paradigms. Like some of the other coding languages that we can find, Python is going to support more than one programming paradigm inside of it. This is going to be a language that can support structured and oriented programming to the fullest. In addition, a language will feature some support for various concepts when it comes to functional and aspect-oriented programming.

Along with all of this, the Python language is going to feature a kind of system that is dynamically typed and some automatic management of the memory. The programming paradigms and language features will help us to work with Python to develop complex and large software applications when we want to.

Compatible With Most Major Systems and Platforms

Right now, Python is able to support many different operating systems. It is even possible to work with interpreting to run the code on some of the specific tools and platforms that we want to use. In addition, since this is known as a language that is interpreted, it is going to allow us to go through and run the exact same code on many different platforms without the need of doing any recompilation.

Because of this, you are not required to recompile the code when you are done altering it. You can go through and run the application code that you modified without recompiling and check the impact of the changes that happened to that code right away. The feature makes it a lot easier to go through and make some changes to the code without having to worry about the development time along the way.

It Can Simplify Some of the Work That You Are Doing

Python is seen as a programming language that is general-purpose in nature. This means that you are able to use this language for all of the different processes and programs that you want to, from web applications to developing things like desktop applications as needed. We can even take it further and use this language to help develop complex scientific and numeric applications.

Python was designed with a lot of features that are there to facilitate the data analysis and visualizations that we will talk about in this guidebook. In

addition, you can take advantage of these features in Python to create some custom big data solutions without having to put in the extra effort or time.

As we can see, there are a number of benefits that we are able to enjoy when it comes to using the Python language, and this is just the beginning. As we go through and learn more about how to work in this language and what it is able to do for us, we will be able to see more and more of the benefits at the same time, and it will not take long working with your own data analysis to understand exactly how great this can be for our needs.

Installing Python

Follow the instructions below to download and install Python on your operating system by referring to the relevant section. The latest version of Python released in the middle of 2019 is Python 3.8.0. Make sure to download and install the most recent and stable version of Python at the time.

Windows

1. From the official Python website, click on the “Downloads” icon and select Windows.
2. Click on the “Download Python 3.8.0” button to view all the downloadable files.
3. On the subsequent screen, select the Python version you would like to download. In this book, we will be using the Python 3 version under “Stable Releases.” So scroll down the page and click on the “Download Windows x86-64 executable installer” link, as shown in the picture below.

- [Python 3.8.0 - Oct. 14, 2019](#)

Note that Python 3.8.0 *cannot* be used on Windows XP or earlier.

- Download [Windows help file](#)
- Download [Windows x86-64 embeddable zip file](#)
- Download [Windows x86-64 executable installer](#)
- Download [Windows x86-64 web-based installer](#)
- Download [Windows x86 embeddable zip file](#)
- Download [Windows x86 executable installer](#)
- Download [Windows x86 web-based installer](#)

4. A pop-up window titled “python-3.8.0-amd64.exe” will be shown.
5. Click on the “Save File” button to start downloading the file.
6. Once the download has completed, double click the saved file icon, and a “Python 3.8.0 (64-bit) Setup” pop window will be shown.
7. Make sure that you select the “Install Launcher for all users (recommended)” and the “Add Python 3.8 to PATH” checkboxes. Note – If you already have an older version of Python installed on your system, the “Upgrade Now” button will appear instead of the “Install Now” button, and neither of the checkboxes will be shown.
8. Click on “Install Now” and a “User Account Control” pop up window will be shown.
9. A notification stating, “Do you want to allow this app to make changes to your device” will be shown. Click on “Yes.”
 10. A new pop up window titled “Python 3.8.0 (64-bit) Setup” will be shown containing a setup progress bar.
 11. Once the installation has been completed, a “Set was successful” message will be shown. Click on “Close.”
 12. To verify the installation, navigate to the directory where you installed Python and double click on the

python.exe file.

Macintosh

1. From the official Python website, click on the “Downloads” icon and select Mac.
2. Click on the “Download Python 3.8.0” button to view all the downloadable files.
3. On the subsequent screen, select the Python version you would like to download. In this book, we will be using the Python 3 version under “Stable Releases.” So scroll down the page and click on the “Download macOS 64-bit installer” link under Python 3.8.0, as shown in the picture below.

- 
- [Python 3.7.5 - Oct. 15, 2019](#)
 - Download [macOS 64-bit/32-bit installer](#)
 - Download [macOS 64-bit installer](#)
 - [Python 3.8.0 - Oct. 14, 2019](#)
 - Download [macOS 64-bit installer](#)
 - [Python 3.7.4 - July 8, 2019](#)
 - Download [macOS 64-bit/32-bit installer](#)
 - Download [macOS 64-bit installer](#)
 - [Python 3.6.9 - July 2, 2019](#)

4. A pop up window titled “python-3.8.0-macosx10.9.pkg” will be shown.
5. Click “Save File” to start downloading the file.
6. Once the download has completed, double click the saved file icon, and an “Install Python” pop window will be shown.
7. Click “Continue” to proceed, and the terms and conditions pop-up window will appear.
8. Click “Agree” and then click “Install.”

9. A notification requesting administrator permission and password will be shown. Enter your system's password to start the installation.
10. Once the installation has finished, an "Installation was successful" message will appear. Click on the "Close" button, and you are all set.
11. To verify the installation, navigate to the directory where you installed Python and double click on the python launcher icon that will take you to the Python Terminal.

Definitions: Python Interpreter, Idle, and the Shell

Python Interpreter

The python interpreter is the program responsible for executing the scripts you write. The interpreter converts the .py script files into bytecode instructions and then processes them according to the code written in the file.

Python IDLE

IDLE is the Python Integrated Development and Learning Environment. It contains all of the tools you will need to develop programs in Python, including the shell, a text editor, and debugging tools.

Depending on your Python version and operating system, IDLE can be very basic or have an extensive array of options that can be set up.

For example, on Mac OS X, the text editor can be set up with several code indentations and highlighting options, which can make your programs much easier to read and work with.

If the text editor in IDLE does not offer the sophistication you need, there are several aftermarket text editors that support Python script highlighting, autocomplete, and other features that make script writing easier.

Python Shell

The shell is an interactive, command-line driven interface to the Python interpreter.

In the Python shell, commands are entered at the `>>>` prompt. Anything that is entered at the prompt must be in proper Python syntax, incorrect entries will return a syntax error like:

```
SyntaxError: invalid syntax
```

When a command is entered, it is specific to that shell and has the lifetime of the shell.

For example, if you assign a variable a value such as:

```
>>>X=10
```

Then the variable is assigned an integer value of 10.

That value will be maintained until the shell is closed, restarted, or the value is changed.

If another shell window is opened, the value of X will not be accessible in the new window.

When a command is entered and accepted, the code is executed. If the entered code generates a response, the response will be output to the specified device. If it does not, simply assigning a variable as above, then another prompt (`>>>`) will be shown, and additional commands can be entered.

This can be useful for a number of simple tasks, testing simple functions and getting a feel for how commands work.

As an example, enter the following:

```
>>>X=10
>>>Y=5
>>>print(X)
10
>>>print(Y)
5
>>>print(X+Y)
15
```

This demonstrates a couple of things.

First, we assign the two variables X and Y values. Both variables retain their

values within the shell. It also shows that the way we defined the variables was acceptable. If it is acceptable in the shell, it will be acceptable in a script.

If a command is not acceptable, it will return an error or exception.

For example, if we ask for the length of X with the following command.

```
>>>print(len(X))
```

Then the following is returned:

Traceback (most recent call last):

File "<pyshell#12>", line 1, in <module>

```
print(len(X))
```

```
TypeError: object of type 'int' has no len()
```

The error returned usually will provide some valuable information as to why the error occurred. In this case, it is telling us that we assigned an integer value to X.

The len() command gives the length of a string, so we are getting this error because the type of data held by the variable does not match the requirements of the function called.

If instead we had used:

```
>>>print(len(str(X)))
```

```
2
```

In this case, we are using the str() command to convert the value of X into a string.

We are then using len() to get the length of that string, which is 2 characters.

This can be loosely translated into:

```
X=12 → str(X)='12' → len('12')=2
```

We can continue to use the shell to explore other things like different ways to assign variable values.

For example, rather than explicitly assigning values on a per line basis,

variables can be assigned as comma separated groups.

```
>>>X,Y = 20,12
```

```
>>>print(X,Y)
```

```
20 12
```

Script Editor

To create our first program, open the text editor.

To open it in a GUI OS like OS X or Windows, select File->New from the IDLE menus.

In non-GUI implementations .py files can be created in a terminal text editor like VI or VIM. Please see the documentation on those programs for information on working in them.

Once a text window is open, we can simply enter our program code.

In this case, we will write a quick program for calculating the volume of a cylinder. The formula is $V=(\pi r^2)*h$ where r is the radius, and h are the height.

While this program will be extremely simple, and could easily be done just using the shell, it will show several fundamentally important things in Python programming.

The first step will be to import the math library.

Many functions available in Python are stored in libraries. These libraries typically house functions which are grouped by tasks, such as math.

If the proper library is not loaded when prior to making a call to that library, we will get an error such as:

```
Traceback (most recent call last):
```

```
File "<pyshell#22>", line 1, in <module>
```

```
print(math.exp(2))
```

```
NameError: name 'math' is not defined
```

This error is telling you that math is not defined.

Since math is part of a Python standard library, this tells you that the library was not imported prior to the execution of the request for the math keyword.

In the text editor, enter the lines as follows

```
# import math library  
  
import math
```

The # is the python comment symbol. Anything between that and the end of the line is ignored by the interpreter.

One of the key advantages of Python scripting is readability, so it is very important (as it is in all programming) to be diligent about commenting.

Comments will make it easier to debug your code later and will make it easier for someone else to look at your work and see how the program works.

In many cases, it also forces you to slow down and think out the programming process as you go, which will lead to cleaner and better-organized code.

Next, we need to set up our variables.

This can be done anywhere within a script as long as they are defined before calling for their value.

If a variable is called before it is defined, a 'name not defined' exception will be displayed, and program execution will halt.

```
# assign variables  
  
r=5 # radius  
  
h=10 # height  
  
V=0 # volume
```

While V does not need to be explicitly defined, here it is considered good practice to do so because it makes the code easier to understand.

Next, we do the actual volume calculation.

```
# calculate volume of a cylinder  
  
V=(math.pi*math.pow(r,2))*h # volume=( $\pi$ *r2)*h
```

Next, to see the result we use the print function, which will output the result to the console.

```
# output the result
```

```
print(V)
```

The complete program looks like this:

```
# import math
```

```
import math
```

```
# assign variables
```

```
r=5 # radius
```

```
h=10 # height
```

```
V=0 # volume
```

```
# calculate volume of a cylinder
```

```
V=(math.pi*math.pow(r,2))*h # volume=( $\pi$ *r2)*h
```

```
# output the result
```

```
print(V)
```

You can save the program to your harddrive, let's call it cylinder.py.

Python views files ending in .py as script files, so it is important to always save your scripts with the .py extension. Once we have a saved script file, we can go ahead and run it.

Using 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 menus, customization options, and GUI functions, while the Python shell is

devoid of those and only offers 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.

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.

Good Programming Practices

Think about why you want to learn to code in the first place. For a lot of people, it is because they want to make something that just doesn't exist yet. Others may want to do it because they know software developers are high-paying jobs and happen to be high in demand by tech and non-tech companies alike. Identify what drives you and then start looking for sources of inspiration that encourage you to stay focused on your goals. For a coder, inspiration would be something like working for Apple or building the next competitor of Twitter. Identifying your vision is necessary for steering you towards the right tools. Like if you want to get involved with artificial intelligence and tinker with robots, Python is a functional starting language and has a lot of flexibility. If you're going to build the next Candy Crush addiction, you will need to focus on mobile apps and learn Swift, Objective-C, Kotlin, and Java.

Writing Comments in Python

Comments are always used for programmer's convenience in any programming language. A comment is used to describe the features of a program. When we write a program then we can write comments for classes, functions, statements, etc. Comments can also help programmers to find errors in large programs. If there is any error occurring in a large program (let say thousands of lines), then the programmer can try to find the error with the help of comments with less effort. Python has two types of comments – single-line comment and multi-line comment.

Single-Line Comment

A single-line comment starts with the symbol (#). A # denotes that this line is a comment and should not be executed by Python. Single-line comments should be used when we just have to comment a single line. Class and Static Methods

Methods in classes can be categorized into two: class and static methods. By default, all methods are automatically set as class methods.

The main difference between the two is that class methods automatically pass the calling object as an argument by reference while static methods do not. This is the reason it is required to have a self or any parameter that will receive the reference in class methods.

Without a parameter to catch the reference to the calling object, Python will return an error. For example:

```
>>> def sampleClass():
    def sampleFunction():
        print("Nothing")
```

```
>>> x = sampleClass()
```

```
>>> x.sampleFunction()
```

Traceback (most recent call last):

```
File "<stdin>", line 1, in <module>
```

TypeError: sampleFunction() takes 0 positional arguments but 1 was given

```
>>> _
```

Static methods, on the other hand, do not pass the any argument by default, so you can create methods that do not contain parameters.

In order to define class and static methods you need to use the decorator operator (@). For example:

```
>>> class classA():
    @classmethod
```

```
def methodA():
    print("This is a Class Method.")
```

```
@staticmethod
def methodB():
    print("This is a Static Method.")
```

```
>>> class classB(classA):
    pass
```

```
>>> x = classB()
```

```
>>> x.methodB()
```

This is a Static Method.

```
>>> x.methodA()
```

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

TypeError: sampleFunction() takes 0 positional arguments but 1 was given

```
>>> _
```

The decorator operator (@) has other functionalities other than defining methods as static or class.

```
x = 10 # store 10 into x
```

```
y = 20 # store 20 into y
```

Multi-Line Comments

We can also create multi-line comments that will span multiple lines. Triple double quotes (""") or triple single quotes (''') are used to create multi-line or block comments. We must write quotes ("""" or ''') at the beginning and

ending of the comment. If there are many comments in our program, then creating single-line comments is a very tedious job, so we can use multi-line comments instead. For example:

```
# a program to add two numbers
# first store 2 integer numbers
# store 5 into x and 10 into y
# add x and y and store into z
x=5
y=10
z= x + y
```

As you can see, we want to write many comments in the same place, so using the # symbol will be a very tedious job so we can use multi-line comments instead, like this:

```
""" a program to add two numbers
first store 2 integer numbers
store 5 into x and 10 into y
add x and y and store into z """
x=5
y=10
z= x+ y
```

Chapter 2:

Variables and Operations

Data Types

Data is very important in every field, so it should be handled very carefully. When we write programs, then we perform operations on data.

Data type refers to the type of data. Type of data must be known before storing data into system because if data type is not known then it may lead to various problems such as undesirable output.

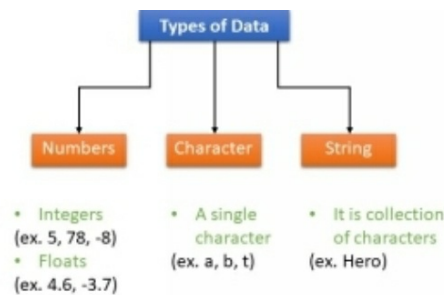


Figure 3.1

These are the common types of data that we generally use. There are many more data types in Python that we will discuss later in this chapter. In programming languages like C, C++ and Java, we must define the type of data before using it. But in Python, we do not need to worry about it, since Python dynamically checks for data type, and that is why we called Python a dynamically-typed language (see strengths/features of Python in chapter 1). We need not declare anything, but we should know the type of data as a programmer.

For example, if I want to store a number in Python, then I will write

something like this:

```
>>> x=20
```

As you can see we did not specify anything to Python before storing number 20 into x (here x is a variable which we will discuss soon). Python will find out automatically that this is an Integer number. But in other languages such as C, we must tell the C compiler the data type –

```
int x=20;
```

See? We told the C compiler that x is an Integer, but don't worry about it in Python. Just store any data without worrying about its type.

String Manipulation and Formatting

The string Format operator (%) is arguably one of the fanciest features in Python. The operator shares a uniqueness to string and makes up for sharing functions with the print() family in C. For example:

```
#!/use/bin/python
```

```
print 'My name is %s, and weight is %d kilograms!' % ("Amos", 25)
```

When this line of code is executed, the following is outputted:

```
'My name is Amos and weight is 25 kilograms.'
```

Below is a complete list of symbols which can be combined with the format operator:

%c: To convert into character.

%s: To make string conversion through str() before formatting is done.

%i: To convert into signed decimal integers.

%d: To convert into signed decimal integers.

%u: To convert into unsigned decimal integers.

%o: To convert into octal integers.

%x: To convert into lowercase hexadecimal integers.

%X: To convert into uppercase hexadecimal integers.

`%e`: To convert exponential notations with a lowercase “e.”

`%E`: To convert exponential notations with an uppercase “E.”

`%f`: To convert into a floating-point real number.

`%g`: To convert into the shorter form of `%f` and `%e`

`%G`: To convert into the shorter form of `%f` and `%E`.

There are also many symbols supported by the format operator with different functionalities. They are:

`(*)`: The argument indicates precision or width.

`(-)`: Stands for left justification.

`(+)`: Used to display the sign.

`(<sp>)`: Used to leave the space before a positive number blank.

`(#)`: Used to add the hexadecimal leading 'OX' or 'Ox', or Octal leading “0” (zero) depending on the use of either 'X' or 'x.'

`(0)`: Used to pad towards the right with zeros rather than spaces.

`(%)`: “%%” yields only a single literal “%.”

`(var)`: Used as a mapping variable (dictionary elements).

`(m.n.)`: m refers to the minimum total width while n represents the number of digits which is displayed after a decimal point when applied.

Type Casting

Type casting 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 an integer into a float number. Python has two types of conversion: implicit type conversion and explicit type 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

Converting 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))
```

Chapter 3:

Data structures

Lists

Python lists offer changeable and ordered data and written while accompanying square brackets, for example, "an apple," "cherry." Accessing an already existing list by referring to the index number while having the ability to write negative indexes such as '-1' or '-2'.

You can also maneuver within your list and select a specific category of indexes by first determining your starting and endpoints.

The return value will therefore be the range of specified items. You can also specify a scale of negative indexes, alter the value of the current item, loop between items on the list, add or remove items, and confirming if items are available. In Python, lists are collections of data types that can be changed, organized, and include duplicate values. Lists are written within square brackets, as shown in the syntax below.

```
X = ["string1", "string2", "string3"]  
print (X)
```

The same concept of position applies to lists as the string data type, which dictates that the first string is considered to be at position 0. Subsequently, the strings that will follow are given positions 1, 2, and so on.

You can selectively display the desired string from a list by referencing the position of that string inside square bracket in the print command as shown below.

```
X = ["string1", "string2", "string3"]  
print (X [2])
```

OUTPUT – [string3]

Similarly, the concept of **negative indexing** is also applied to Python list.

Let's look at the example below:

```
X = ["string1", "string2", "string3"]  
print (X [-2])
```

OUTPUT – [string2]

You will also be able to specify a **range of indexes** by indicating the start and end of a range.

The result in values of such command on a Python list would be a new list containing only the indicated items.

Here is an example for your reference.

```
X = ["string1", "string2", "string3", "string4", "string5", "string6"]  
print (X [3 : 5])
```

OUTPUT – ["string4", "string5"]

Remember the first item is at position 0, and the final position of the range (4) is not included.

Now, if you do not indicate the start of this range, it will default to the position 0, as shown in the example below:

```
X = ["string1", "string2", "string3", "string4", "string5", "string6"]  
print (X [: 4])
```

OUTPUT – ["string1", "string2", "string3", "string4"]

Similarly, if you do not indicate the end of this range, it will display all the items of the list from the indicated start range to the end of the list, as shown in the example below:

```
X = ["string1", "string2", "string3", "string4", "string5", "string6"]  
print (X [4 : ])  
OUTPUT – ["string5", "string6"]
```

You can also specify a **range of negative indexes** to Python Lists, as shown in the example below:

```
X = ["string1", "string2", "string3", "string4", "string5", "string6"]
```

```
print (X [-4 : -1])
```

OUTPUT – [“string3”, “string4”, “string5”]

Remember the last item is at position -1, and the final position of this range (-1) is not included in the output.

There might be instances when you need to **change the data value** for a Python List.

This can be accomplished by referring to the index number of that item and declaring the new value.

Let’s look at the example below:

```
X = [“string1”, “string2”, “string3”, “string4”, “string5”, “string6”]  
X [2] = “newstring”  
print (X)
```

OUTPUT – [“string1”, “string2”, “newstring”, “string4”, “string5”, “string6”]

You can also determine the **length** of a Python List using the “len()” function, as shown in the example below:

```
X = [“string1”, “string2”, “string3”, “string4”, “string5”]  
print (len (X))
```

OUTPUT – 5

Python Lists can also be changed by **adding new items** to an existing list using the built-in “append ()” method, as shown in the example below:

```
X = [“string1”, “string2”, “string3”, “string4”, “string5”]  
X.append (“newstring”)  
print (X)
```

OUTPUT – [“string1”, “string2”, “string3”, “string4”, “string5”, “newstring”]

You can also, add a new item to an existing Python List at a specific position using the built-in “insert ()” method, as shown in the example below:

```
X = [“string1”, “string2”, “string3”, “string4”]
```

```
X.insert (3, "newstring")  
print (X)
```

OUTPUT – ["string1", "string2", "string3", "newstring"]

There might be instances when you need to **copy** an existing Python list. This can be accomplished by using the built-in "copy ()" method or the "list ()" method, as shown in the example below:

```
X = ["string1", "string2", "string3", "string4", "string5", "string6"]  
Y = X.copy( )  
print (Y)
```

OUTPUT – ["string1", "string2", "string3", "string4", "string5", "string6"]

```
X = ["string1", "string2", "string3", "string4", "string5", "string6"]  
Y = list (X)  
print (Y)
```

There are multiple built-in methods to **delete items** from a Python list.

To selectively delete a specific item, the "remove ()" method can be used.

```
X = ["string1", "string2", "string3", "string4"]  
X.remove ("string2")  
print (X)
```

OUTPUT - ["string1", "string3", "string4"]

To delete a specific item from the list, the "pop ()" method can be used with the position of the value. If no index has been indicated, the last item of the index will be removed.

```
X = ["string1", "string2", "string3", "string4"]  
X.pop ( )  
print (X)
```

OUTPUT - ["string1", "string2", "string3"]

To delete a specific index from the list, the "del ()" method can be used, followed by the index within square brackets.

```
X = ["string1", "string2", "string3", "string4"]
```

```
del X [2]
```

```
print (X)
```

```
OUTPUT - ["string1", "string2", "string4"]
```

To delete the entire List variable, the “del ()” method can be used, as shown below.

```
X = ["string1", "string2", "string3", "string4"]
```

```
del X
```

```
OUTPUT - .....
```

To delete all the string values from the List without deleting the variable itself, the “clear ()” method can be used, as shown below.

```
X = ["string1", "string2", "string3", "string4"]
```

```
X.clear()
```

```
print (X)
```

```
OUTPUT - [ ]
```

Concatenation of Lists

You can join multiple lists with the use of the “+” logical operator or by adding all the items from one list to another using the “append ()” method. The “extend ()” method can be used to add a list at the end of another list. Let’s look at the examples below to understand these commands.

```
X = ["string1", "string2", "string3", "string4"]
```

```
Y = [11, 22, 33, 40]
```

```
Z = X + Y
```

```
print (Z)
```

```
OUTPUT - ["string1", "string2", "string3", "string4", 11, 22, 33, 40]
```

```
X = ["string1", "string2", "string3", "string4"]
```

```
Y = [11, 22, 33, 40]
```

```
For x in Y:
```



```
X.append (x)
```

```
print (X)
```

```
OUTPUT – [“string1”, “string2”, “string3”, “string4”, 11, 22, 33, 40]
```

```
X = [“string1”, “string2”, “string3”]
```

```
Y = [11, 22, 33]
```

```
X.extend (Y)
```

```
print (X)
```

```
OUTPUT – [“string1”, “string2”, “string3”, 11, 22, 33]
```

EXERCISE: Create a list “A” with string data values as “red, jade, teal, violet, yellow” and display the item at -2 position.

USE YOUR DISCRETION HERE AND WRITE YOUR CODE FIRST

Now, check your code against the correct code below:

```
A = [“red,” “jade,” “teal,” “violet,” “yellow”]
```

```
print (A [-2])
```

```
OUTPUT – [“violet”]
```

EXERCISE: Create a list “A” with string data values as “red, jade, teal, violet, yellow” and display the items ranging from the string on the second position to the end of the string.

USE YOUR DISCRETION HERE AND WRITE YOUR CODE FIRST

Now, check your code against the correct code below:

```
A = [“red,” “jade,” “teal,” “violet,” “yellow”]
```

```
print (A [2 : ])
```

```
OUTPUT – [“red,” “teal,” “teal,” “violet,” “yellow”]
```

EXERCISE: Create a list “A” with string data values as “red, jade, teal, violet, yellow” and replace the string “jade” to “teal.”

USE YOUR DISCRETION HERE AND WRITE YOUR CODE FIRST

Now, check your code against the correct code below:

```
A = [“red,” “jade,” “teal,” “violet,” “yellow”]
```

```
A [1] = ["teal"]
```

```
print (A)
```

```
OUTPUT – ["teal," "violet," "yellow"]
```

EXERCISE: Create a list "A" with string data values as "red, jade, teal, violet, yellow" and copy the list "A" to create list "B."

USE YOUR DISCRETION HERE AND WRITE YOUR CODE FIRST

Now, check your code against the correct code below:

```
A = ["red," "jade," "teal," "violet," "yellow"]
```

```
B = A.copy ( )
```

```
print (B)
```

```
OUTPUT – ["red," "jade," "teal," "violet," "yellow"]
```

EXERCISE: Create a list "A" with string data values as "red, jade, teal, violet, yellow" and delete the strings "red" and "violet."

USE YOUR DISCRETION HERE AND WRITE YOUR CODE FIRST

Now, check your code against the correct code below:

```
A = ["red," "jade," "teal," "violet," "yellow"]
```

```
del.A [0, 2]
```

```
print (A)
```

```
OUTPUT – ["jade," "teal," "yellow"]
```

Tuples

In Python, Tuples are collections of data types that cannot be changed but can be arranged in a specific order. Tuples allow for duplicate items and are written within round brackets, as shown in the syntax below.

```
Tuple = ("string1", "string2", "string3")
```

```
print (Tuple)
```

Similar to the Python list, you can selectively display the desired string from a Tuple by referencing the position of that string inside a square bracket in the print command, as shown below.

```
Tuple = ("string1", "string2", "string3")  
print (Tuple [1])
```

OUTPUT – ("string2")

The concept of **negative indexing** can also be applied to Python tuple, as shown in the example below:

```
Tuple = ("string1", "string2", "string3", "string4", "string5")  
print (Tuple [-2])
```

OUTPUT – ("string4")

You will also be able to specify a **range of indexes** by indicating the start and end of a range. The result in values of such command on a Python tuple would be a new tuple containing only the indicated items, as shown in the example below:

```
Tuple = ("string1", "string2", "string3", "string4", "string5", "string6")  
print (Tuple [1:5])
```

OUTPUT – ("string2", "string3", "string4", "string5")

Remember the first item is at position 0, and the final position of the range, which is the fifth position in this example, is not included. You can also specify a **range of negative indexes** to Python tuples, as shown in the example below:

```
Tuple = ("string1", "string2", "string3", "string4", "string5", "string6")  
print (Tuple [-4: -2])
```

OUTPUT – ("string4", "string5")

Remember the last item is at position -1 and the final position of this range, which is the negative fourth position. In this example, it is not included in the Output. Unlike Python lists, you cannot directly **change the data value of Python tuples** after they have been created. However, conversion of a tuple into a list and then modifying the data value of that list will allow you to

subsequently create a tuple from that updated list. Let's look at the example below:

```
Tuple1 = ("string1", "string2", "string3", "string4", "string5", "string6")
```

```
List1 = list (Tuple1)
```

```
List1 [2] = "update this list to create new tuple" Tuple1 = tuple (List1)
```

```
print (Tuple1)
```

```
OUTPUT – ("string1", "string2", "update this list to create new tuple,"  
"string4", "string5", "string6")
```

You can also determine the **length** of a Python Tuple using the "len()" function, as shown in the example below:

```
Tuple = ("string1", "string2", "string3", "string4", "string5", "string6")
```

```
print (len (Tuple))
```

```
OUTPUT – 6
```

You cannot selectively delete items from a tuple, but you can use the "del" keyword to **delete the tuple** in its entirety, as shown in the example below:

```
Tuple = ("string1", "string2", "string3", "string4")
```

```
del Tuple
```

```
print (Tuple)
```

```
OUTPUT – name 'Tuple' is not defined
```

You can **join multiple tuples** with the use of the "+" logical operator.

```
Tuple1 = ("string1", "string2", "string3", "string4")
```

```
Tuple2 = (101, 202, 303)
```

```
Tuple3 = Tuple1 + Tuple2
```

```
print (Tuple3)
```

```
OUTPUT – ("string1", "string2", "string3", "string4", 101, 202, 303)
```

You can also use the "tuple ()" constructor to create a tuple, as shown in the example below:

```
Tuple1 = tuple(("string1", "string2", "string3", "string4"))  
print (Tuple1)
```

EXERCISE: Create a Tuple "X" with string data values as "corn, cilantro, carrot, potato, onion" and display the item at -2 position.

USE YOUR DISCRETION HERE AND WRITE YOUR CODE FIRST

Now, check your code against the correct code below:

```
X = ("corn," "cilantro," "carrot," "potato," "onion")  
print (X [-2])
```

OUTPUT – ("potato")

EXERCISE: Create a Tuple "X" with string data values as "corn, cilantro, carrot, potato, onion" and display items ranging from -1 to -3.

USE YOUR DISCRETION HERE AND WRITE YOUR CODE FIRST

Now, check your code against the correct code below:

```
X = ("corn," "cilantro," "carrot," "potato," "onion")  
print (X [-3 : -1])
```

OUTPUT – ("carrot," "potato")

EXERCISE: Create a Tuple "X" with string data values as "corn, cilantro, carrot, potato, onion" and change its item from "potato" to "pepper" using the list function.

USE YOUR DISCRETION HERE AND WRITE YOUR CODE FIRST

Now, check your code against the correct code below:

```
X = ("corn", "cilantro", "carrot", "potato", "onion")  
Y = list (X)  
Y [4] = "pepper"  
X = tuple (Y)  
print (X)
```

OUTPUT – ("corn", "cilantro", "carrot", "potato", "pepper")

EXERCISE: Create a Tuple “X” with string data values as “corn, cilantro, carrot” and another Tuple “Y” with numeric data values as (2, 12, 22), then join them together.

USE YOUR DISCRETION HERE AND WRITE YOUR CODE FIRST

Now, check your code against the correct code below:

```
X = (“corn,” “cilantro,” “carrot”)
```

```
Y = (3, 13, 23)
```

```
Z = X + Y
```

```
print (Z)
```

OUTPUT – (“peas,” “carrots,” “potato,” 3, 13, 23)

Dictionaries

Python dictionaries comprise of indexed, changeable, but unordered items typically written while with curly brackets with keys and values. Some of the activities involved include item access by use of a keyword inside the parentheses; conduct value changes, loop, check critical availability, length of the dictionary, and both adding and removing unwanted items. Besides, Python allows you to copy the dictionary by writing 'dict2 = dict1'. 'dict2' will become a representation to 'dict1' therefore makes any necessary changes automatically. Another way of creating a copy is also by using a built-in Dictionary technique, that is, 'copy.' In Python, Dictionaries are collections of data types that can be changed and indexed but are not arranged in any order. Each item in a Python Dictionary will comprise of a key and its value. Dictionaries do not allow for duplicate items and must be written within curly brackets, as shown in the syntax below.

```
dict = {
```

```
“key01”: “value01”,
```

```
“key02”: “value02”,
```

```
“key03”: “value03”,
```

```
}
```

```
print (dict)
```

You can selectively display desired item value from a Dictionary by referencing to its key inside square brackets in the print command as shown below.

```
dict = {  
    "key01": "value01",  
    "key02": "value02",  
    "key03": "value03",  
}  
X = dict ["key02"]  
print (X)
```

OUTPUT – value02

You can also use the “get ()” method to view the value of a key, as shown in the example below:

```
dict = {  
    "key01": "value01",  
    "key02": "value02",  
    "key03": "value03",  
}  
X = dict.get ("key01")  
print (X)
```

OUTPUT – value01

There might be instances when you need to **change the value** of a key in a Python Dictionary. This can be accomplished by referring to the key of that item and declaring the new value. Let’s look at the example below:

```
dict = {  
    "key01": "value01",
```

```
“key02”: “value02”,
“key03”: “value03”,
}
dict [“key03”] = “NEWvalue”
print (dict)
```

OUTPUT – {“key01”: “value01”, “key02”: “value02”, “key03”: “NEWvalue”}

You can also determine the **length** of a Python dictionary using the “len()” function, as shown in the example below:

```
dict = {
“key01”: “value01”,
“key02”: “value02”,
“key03”: “value03”,
“key4”: “value4”,
“key5”: “value5”
}
print (len (dict))
```

OUTPUT – 5

A Python dictionary can also be changed by **adding** new index key and assigning a new value to that key, as shown in the example below:

```
dict = {
“key01”: “value01”,
“key02”: “value02”,
“key03”: “value03”,
}
dict [“NEWkey”] = “NEWvalue”
```



```
print (dict)
```

```
OUTPUT – {“key01”: “value01”, “key02”: “value02”, “key03”: “value03”,  
“NEWkey”: “NEWvalue”}
```

There are multiple built-in methods to **delete items** from a Python dictionary.

To selectively delete a specific item value, the “pop ()” method can be used with the indicated key name.

```
dict = {  
“key01”: “value01”,  
“key02”: “value02”,  
“key03”: “value03”,  
}  
dict.pop (“key01”)
```

```
print (dict)
```

```
OUTPUT – { “key02”: “value02”, “key03”: “value03” }
```

To selectively delete the item value that was last inserted, the “popitem ()” method can be used with the indicated key name.

```
dict = {  
“key01”: “value01”,  
“key02”: “value02”,  
“key03”: “value03”,  
}  
dict.popitem ( )
```

```
print (dict)
```

```
OUTPUT – { “key01”: “value01”, “key02”: “value02” }
```

To selectively delete a specific item value, the “del” keyword can also be

used with the indicated key name.

```
dict = {  
    "key01": "value01",  
    "key02": "value02",  
    "key03": "value03",  
}
```

```
del dict ("key03")
```

```
print (dict)
```

OUTPUT – { "key01": "value01", "key02": "value02" }

To delete a Python dictionary in its entirety, the "del" keyword can also be used as shown in the example below:

```
dict = {  
    "key01": "value01",  
    "key02": "value02",  
    "key03": "value03",  
}
```

```
del dict
```

```
print (dict)
```

OUTPUT – name 'dict' is not defined

To delete all the items from the dictionary without deleting the dictionary itself, the "clear ()" method can be used as shown below.

```
dict = {  
    "key01": "value01",  
    "key02": "value02",  
    "key03": "value03",
```

```
}  
dict.clear ( )
```

```
print (dict)
```

```
OUTPUT – { }
```

There might be instances when you need to **copy** an existing Python dictionary. This can be accomplished by using the built-in “copy ()” method or the “dict ()” method, as shown in the examples below:

```
dict = {  
    “key01”: “value01”,  
    “key02”: “value02”,  
    “key03”: “value03”,  
}
```

```
newdict = dict.copy ( )  
print (newdict)
```

```
OUTPUT – {“key01”: “value01”, “key02”: “value02”, “key03”: “value03”}
```

```
Olddict = {  
    “key01”: “value01”,  
    “key02”: “value02”,  
    “key03”: “value03”,  
}
```

```
newdict = dict (Olddict )  
print (newdict)
```

```
OUTPUT – {“key01”: “value01”, “key02”: “value02”, “key03”: “value03”}
```

There is a unique feature that supports multiple Python dictionaries to be **nested** within another Python dictionary. You can either create a dictionary containing child dictionaries, as shown in the example below:

```
McManiaFamilyDict = {  
    "burger1" : {  
        "name" : "VegWrap",  
        "price" : 3.99  
    },  
    "burger2" : {  
        "name" : "Burger",  
        "price" : 6  
    },  
    "burger3" : {  
        "name" : "CheeseBurger",  
        "price" : 2.99  
    }  
}  
  
print (McManiaFamilyDict)
```

OUTPUT - {"burger1" : { "name" : "VegWrap", "price" : 3.99}, "burger2" : {"name" : "Burger", "price" : 6}, "burger3" : {"name" : "CheeseBurger", "price" : 2.99}}

Or you can create a brand new dictionary that contains other dictionaries already existing on the system, your code will look like the one below:

```
burgerDict1 : {  
    "name" : "VegWrap",  
    "price" : 3.99  
}  
  
burgerDict2 : {  
    "name" : "Burger",
```

```

“price” : 6
}
burgerDict3 : {
“name” : “CheeseBurger”,
“price” : 2.99
}
McManiaFamilyDict = {
“burgerDict1” : burgerDict1,
“burgerDict2” : burgerDict2
“burgerDict3” : burgerDict3
}
print (McManiaFamilyDict)

```

OUTPUT - {“burger1” : { “name” : “VegWrap”, “price” : 3.99}, “burger2” : {“name” : “Burger”, “price” : 6}, “burger3” : {“name” : “CheeseBurger”, “price” : 2.99}}

Lastly, you can use the “dict ()” function to create a new Python dictionary. The key differences when you create items for the dictionary using this function are:

- Round brackets are used instead of the curly brackets.
- Equal to sign is used instead of the semi-colon.

Let’s look at the example below:

```

DictwithFunction = dict (key1 = “value1”, key2 = “value2”, key3 =
“value3”)
print (DictwithFunction)

```

OUTPUT – {“key1”: “value1”, “key2”: “value2”, “key3”: “value3”}

EXERCISE: Create a Dictionary “CoffeeShop” with items containing keys

as “type”, “size” and “price” with corresponding values as “cappuccino”, “large” and “5.99”. Then add a new item with key as “syrup” and value as “caramel”.

USE YOUR DISCRETION HERE AND WRITE YOUR CODE FIRST

Now, check your code against the correct code below:

```
CoffeeShop = {  
    “type” : “cappuccino”,  
    “size” : “large”,  
    “price” : 5.99  
}  
CoffeeShop [“syrup”] = “caramel”  
print (CoffeeShop)
```

OUTPUT – {“type” : “cappuccino”, “size” : “large”, “price” : 5.99, “syrup” : “caramel”}

EXERCISE: Create a Dictionary “CoffeeShop” with items containing keys as “type”, “size” and “price” with corresponding values as “cappuccino”, “large” and “5.99”. Then use a function to remove the last added item.

USE YOUR DISCRETION HERE AND WRITE YOUR CODE FIRST

Now, check your code against the correct code below:

```
CoffeeShop = {  
    “type” : “cappuccino”,  
    “size” : “large”,  
    “price” : 5.99  
}  
CoffeeShop.popitem ( )  
print (CoffeeShop)
```

OUTPUT – {“type” : “cappuccino”, “size” : “large”}

EXERCISE: Create a Dictionary “CoffeeShop” with a nested dictionary as listed below:

Dictionary Name Key Value

Coffee1 name cappuccino

size xlarge

Coffee2 name espresso

size large

Coffee3 name mocha

size small

USE YOUR DISCRETION HERE AND WRITE YOUR CODE FIRST

Now, check your code against the correct code below:

```
CoffeeShop = {  
    “coffee1” : {  
        “name” : “cappuccino”,  
        “size” : “xlarge”  
    },  
    “coffee2” : {  
        “name” : “espresso”,  
        “size” : “large”  
    },  
    “coffee3” : {  
        “name” : “mocha”,  
        “size” : “small”  
    }  
}  
  
print (CoffeeShop)
```

OUTPUT - {“coffee1” : { “name” : “cappuccino”, “size” : “xlarge”}, “coffee2” : {“name” : “espresso”, “size” : “large”}, “coffee3” : {“name” :

“mocha”, “size” : “small”}}

EXERCISE: Use the “dict ()” function to create a dictionary “CoffeeShop” with items containing keys as “type”, “size” and “price” with corresponding values as “cappuccino”, “large” and “5.99”.

USE YOUR DISCRETION HERE AND WRITE YOUR CODE FIRST

Now, check your code against the correct code below:

```
CoffeeShop = dict (type = “cappuccino”, size = “large”, price = 5.99)
```

```
print (CoffeeShop)
```

OUTPUT – {“type” : “cappuccino”, “size” : “large”, “price” : 5.99, “syrup” : “caramel”}

Data Structures Exercises

Write a program to print all values of a list.

```
lst = [1,2,3,4,5]
n = len(lst)
i = 0
while i<n:
    print(lst[i])
    i+=1
```

Output –

```
1
2
3
4
5
```

Write a python program that inputs two tuples and creates a third that contains all elements of the first, followed by all elements of the second.


```
# eval() function can be used to take tuple from user
tp11 = eval(input("Enter first tuple: "))
tp12 = eval(input("Enter second tuple: "))
tp13 = tp11 + tp12
print("Result:", tp13)
```

Output –

```
Enter first tuple: 1,2,3
Enter second tuple: 4,5,6,7
Result: (1, 2, 3, 4, 5, 6, 7)
```

WAP to print the alternate elements of a tuple tpl.

```
tpl = (1,2,3,4,5,6,7,8,9,10)
for i in range(0, len(tpl), 2):
    print(tpl[i])
```

Output –

```
1
3
5
7
9
```

WAP to print every 3rd element of a tuple T, raised to power 3.

```
T = (1,2,3,4,5,6,7,8,9,10)
for i in range(0, len(T), 3):
    print(T[i]**3)
```

Output –

```
1
64
343
1000
```

WAP that input a number n and creates a tuple containing: n, 2n, 3n and 4n.

```
lst = [] # because tuple is immutable so we cannot add element
n = int(input('Enter a number: '))
for i in range(1,5):
    lst.append(n*i)
tpl = tuple(lst) # now convert list to tuple
print(tpl)
```

Output-

```
Enter a number: 2
(2, 4, 6, 8)
```

Chapter 4:

Inputs, Printing, and Formatting Outputs

Inputs

The job of a computer is to take input and give output. A computer accepts data from the user through an input device such as a keyboard, processes the data by a processor or CPU, and gives the output back to the user on an output device such as a computer screen. Without inputting the data, computer cannot output the result by itself. **To input data into computer through an input device, we need input statements. To get output from computer on an output device, we need output statements.** So input and output statements are mandatory parts of any programming language.

For example, let's assume that we have created a simple accounting program that calculates simple interest, compound interest, etc. Can we imagine that, without entering the data into that application, we can get output? The answer is no. We must have used input and output statements in that application so that we can enter the data, and the application tells us simple and compound interest, etc. as output. We cannot imagine our programs without input and output statements. Input statement – A statement or function which is used to take input from the user through an input device such as a keyboard.

Output statement – A statement or function which is used to give or print the output to the user on an output device such as a computer screen.

Let's write a simple program to input the user's name with the help of input statement and print the name on the screen with the help of an output statement:

```
# Program to take input from user through keyboard using an input function
# or statement and printing the name on screen using an output statement
name = input("Enter your name : ")#taking input from user using input() function
print(name) # printing name on screen using print() function
```

Output –

```
Enter your name : Pradeep choudhary
Pradeep choudhary
```

As you can see, we used an input statement or function `input()` to take input from the user and store the user's name into a string variable 'name,' then we used an output statement `print()` to print the name on the screen. See the output – the first line is asking the user to input his/her name. When he/she enters the name then it will be displayed on the screen. Now let's understand input and output statements and how they work in Python.

Input and Formatting

Input Statement

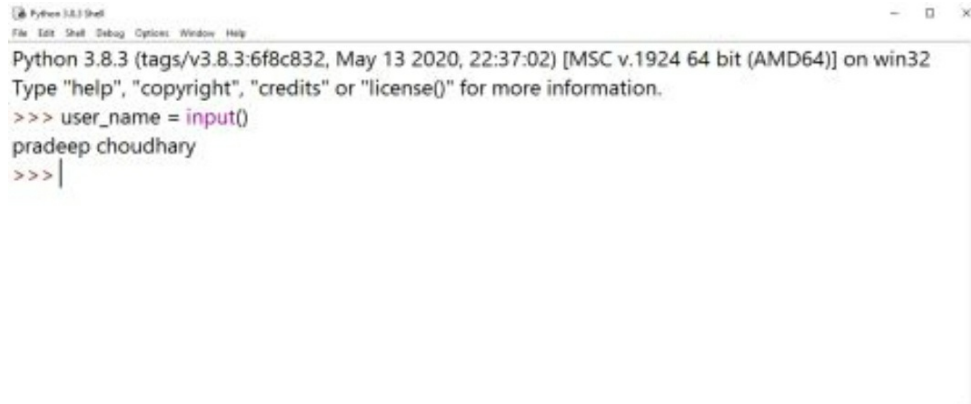
To take input from the user through the keyboard, Python provides the `input()` function. After taking input from the user, we should store that input into a variable or object so that we can use it later in our program. Syntax to use `input()` function is:

```
<object> = input("message")
```

Where the object can be a variable, list, string, tuple etc. to store the value input by the user, the message is optional, but we should always use it. Let's input the user's name in a variable `user_name` –

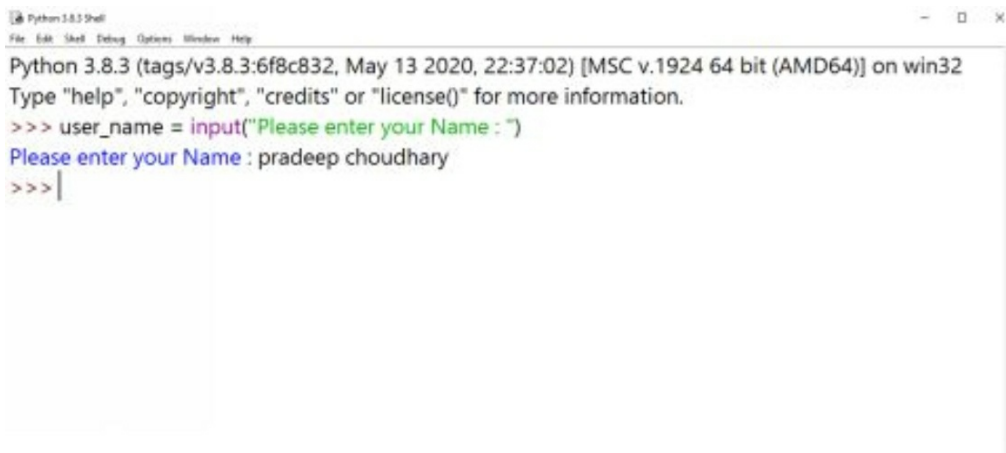
```
user_name = input()
```

By writing this statement, the user can input his name through keyboard, and then the input will be stored into variable `user_name`. But we didn't write the proper message so, the user will not be able to understand that what input we want from the user. See this:



```
Python 3.8.3 (tags/v3.8.3:6f8c832, May 13 2020, 22:37:02) [MSC v.1924 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> user_name = input()
pradeep choudhary
>>> |
```

See the line below the `input()` function, we did not display any message to the user that we want him/her to input his/her name. Now observe the following image where we will provide a proper message to the user:



```
Python 3.8.3 (tags/v3.8.3:6f8c832, May 13 2020, 22:37:02) [MSC v.1924 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> user_name = input("Please enter your Name : ")
Please enter your Name : pradeep choudhary
>>> |
```

The difference is clear, the second approach is better. We should always provide a proper message in the `input()` function.

Note: The `input()` function always takes a string as input by default.

No matter what the user inputs, the `input()` function will take it as a string. Let's input an integer number this time:

```
# Let's input an integer
number = input("Enter a number : ") # it will be string by default
print(number)
print(type(number)) # check the type of number
```

Output –

```
Enter a number : 45
45
<class 'str'>
```

See? Although we wanted an integer number, the input() function takes it as a string. To take an integer number, we can convert the input into an integer by using the int() function. We can also convert the input into a float by using the float() function. Just write the int() before the input() function. Let's take an example:

```
# Let's input an integer with int() function
number = int(input("Enter an integer: ")) # Let's convert the string into
int
print(number)
print(type(number)) # check the type of number
# Let's input a float with float() function
number = float(input("Enter a float : ")) # Let's convert the string into
float
print(number)
print(type(number)) # check the type of number
```

Output –

```
Enter an integer: 45
45
<class 'int'>
Enter a float : 55.67
55.67
<class 'float'>
```

:

So we can convert the input value into an int, float etc. by writing the appropriate function before the input(). Now you have enough knowledge to write basic programs which take inputs and produce output.

Chapter 5:

Conditional Statements and Control Flow Statement

Handling Conditions

The control flow in this language can be important. This control flow is there to ensure that you wrote out the code the proper way. There are some types of strings in your code that you may want to write out so that the compiler can read them the right way. But if you write out the string in the wrong manner, you are going to end up with errors in the system. We will take a look at many codes in this guidebook that follows the right control flow for this language, which can make it easier to know what you need to get done and how you can write out codes in this language.

Conditional Statements

These are going to be a simple thing to work on when it comes to Python. They are simply going to be strings of code that you are able to write out, and then you will tell the compiler to show that string on the computer string at the right time.

When you give the compiler the instructions that it needs to follow, you will find that there are going to be statements that come with it. As long as you write these statements out in the right manner, the compiler is going to be able to read them and will show the message that you have chosen on the computer screen. You are able to choose to write these statements out as long or as short as you would like, and it all is going to depend on the kind of code that you are trying to work on at the time.

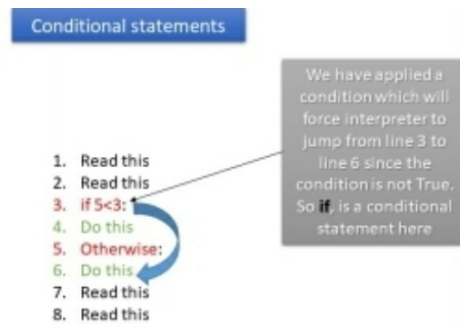
Control Flow Statements

These statements control or change the flow of the program by applying conditions. And based on those conditions, the interpreter may jump from

one statement to another statement. Python has three conditional statements:

- if statement
- if else statement
- if elif else statement

In which program goes sequentially without any jump and in which the interpreter can jump. We can force the interpreter to jump from one statement to another statement by using 'conditional statements'. See the image below stating how conditions can be written in the program and how the interpreter jumps based on those conditions:



Now let's understand all the three conditional statements one by one.

If Statement

This is a conditional statement which is used to make decisions based on conditions in a program. That is why it is also called decision making or testing statement. Remember, relational and logical operators are used to write conditions such as $5 < 3$, $(6 > 2)$ and $4 == 4$ etc. and then these conditions can be written in an 'if' statement. The syntax to write an if statement is:

if condition:

statements

Observe the spaces before statements, which tells that these statements

belong to this particular if block only. We can write a single statement or multiple statements. If we write multiple statements, then all the statements must have the same number of spaces since they will all belong to the same block. You can add any number of spaces, but at least 1 space is required.

Also, observe the colon (:) symbol after the condition. The colon represents that the body of 'if' has been started so we must put spaces after the colon symbol. Syntax to write multiple statements:

if condition:

Statement1

Statement2

Statement3

We can write any condition such as $5 < 7$, $8 < 0$, $5 != 56$ etc., which evaluates a Boolean type i.e., True or False. First, the condition will be checked, if the condition evaluates to True, then all these statements will be executed, and if the condition evaluates to False, then the interpreter will not execute these statements, it will ignore them. Let's take an example in which we will check whether a person can vote.

To do this, we will take a variable named 'age' and then we will check the condition that 'if age is greater or equal to 18', we will simply print 'you can vote' and if age is 'less than 18' then we will print 'you cannot vote.'

```
# to check whether a person can vote
age = int(input("Enter your age: "))
if age>=18:
    print("you can vote")
if age<18:
    print("you can not vote!")
```

Output –

```
Enter your age: 17
you cannot vote!

Enter your age: 19
you can vote

Enter your age: 18
you can vote
```

When the input was 17, the condition ‘age>=18’ becomes False, so the statement ‘`print("you can vote")`’ will not be printed. Then the next condition will be checked that is ‘age<18’ which evaluates to True because 17<18, so the statement ‘`print("you cannot vote!")`’ will be printed on the screen. The same concept will be applied to other inputs.

Let’s take one more example, this time with multiple statements:

```
# to check whether a person can vote with multiple statements
age = int(input("Enter your age: "))
if age>=18:
    print("yes! yor are an adult")
    print("you can vote")
    print("all the best.")
if age<18:
    print("sorry! you are a kid.")
    print("you can not vote!")
```

Output –

```
Enter your age: 21
yes! yor are an adult
you can vote
all the best.

Enter your age: 15
sorry! you are a kid.
you can not vote!
```

In the first ‘if’ condition, all three statements belong to the same block since they have the same indentation i.e. 4 spaces. So when the condition is True, all three statements will be printed. In the second ‘if’ condition, both statements belong to the same block since they have the same number of spaces. So when the input was 21, the first condition becomes True, and all three statements got printed, but the second condition becomes False, so it wasn’t printed.

And when the input was 15, the first condition becomes False, and all three statements will be skipped, but the second condition becomes True, so it gets printed.

If Else Statement

Statements in an ‘if’ statement get executed when the condition becomes True, but we cannot write statements for False condition i.e. we cannot tell ‘if’ statement what to do when the condition is False. But in an ‘if else’ statement, we can also write statements for False condition. Remember the ‘if’ statement syntax:

```
if AGE>=18:
```

```
    print("you can vote")
```

The print statement will be executed when the condition is True, but we did not write any statements if the condition is False i.e., we cannot tell the interpreter what to do if the condition becomes False. So we can do this by using the 'if else' statement. Syntax of 'if ... else' statement is:

if condition:

statements1

else

statements2

As you can see, we can write statements for False condition as well in an 'if else' statement. Let's take an example:

```
x = int(input("enter value of x: "))
y = int(input("enter value of y: "))
if x>y:
    print("yes x is greater than y")
    print("these will be printed if condition is True")
    print("these statements belong to same if block")
    print("these all will be printed")
else: # it represents the False part
    print("no! x is not greater than y")
    print("these will be printed if condition is False")
```

Output –

```
enter value of x: 10
enter value of y: 5
yes x is greater than y
these will be printed if condition is True
these statements belong to same if block
these all will be printed

enter value of x: 3
enter value of y: 7
no! x is not greater than y
these will be printed if condition is False
```

If ... Elif Else Statement

We can write an 'if' condition inside another 'if' condition using this statement.

This statement can be used if we have many conditions to check. The syntax is:

if condition1:

statements1

elif condition2:

statements2

elif condition3:

statements3

else:

statements4

Let's take an example – you are working in a company as a developer. A customer comes to you and says:

Salary	Bonus
If salary is less than 30000	Bonus = 6% of salary
If salary is greater than or equal to 30000 and less than 50000	Bonus = 4% of salary
If salary is greater than or equal to 50000 and less than 70000	Bonus = 3% of salary
If salary is greater than or equal to 70000	No bonus

```
# first input salary from user
salary = float(input("Enter salary: "))
# Now compute bonus based on given conditions
# formula of bonus = salary*bonus percentage/100
if salary<30000: # If salary is less than 30000
    bonus=salary*6/100
elif salary>=30000 and salary<50000:
    bonus=salary*4/100
elif salary>=50000 and salary<70000:
    bonus=salary*3/100
else:
    bonus=0
print("Your bonus: ",bonus)
```

Output –

```
Enter salary: 50000
your bonus: 1500.0

Enter salary: 20500
Your bonus: 1230.0

Enter salary: 35600
Your bonus: 1424.0

Enter salary: 70000
Your bonus: 0

Enter salary: 75000
Your bonus: 0
```

If vs if Else vs if Elif Else

<p>If</p> <p>Only one condition can be written at once, and the condition evaluates to True then the statements will be executed if the condition evaluates to True</p> <pre>if age>18: print("True")</pre>	<p>if else</p> <p>Only one condition can be written at once and if the condition evaluates to True then the statements following the if part will be executed, and if the condition becomes False then the statements after else will be executed</p> <pre>if age>18: print("True") else: print("False")</pre>	<p>if elif else</p> <p>We can write as many conditions as we want. else part will be executed only if the last elif condition becomes False</p> <pre>if age<18: print("False") elif age>=18: print("True") elif age>50: print("True") else: print("False")</pre>
--	--	---

Note: Whenever a condition evaluates to True, the interpreter will not check the other conditions. It will ignore all the other conditions and will come out from that if....elif....else statement. For example, if age=12, then the first condition is True i.e., 12<18 so, after printing “False”, the interpreter will not check other conditions. It will only go further if conditions become False. Let’s practice some programs on conditional statements.

Break/Continue

Sometimes it may be necessary to exit from a loop. We know that a loop executes several times.

We can use the 'break' statement to exit from a loop before the loop terminates. **'Break' can be used only in loops.** Let's take an example where we are using a loop to print numbers from 1 to 10.

```
for i in range(1,11):  
    print(i)
```

Output –

```
1  
2  
3  
4  
5  
6  
7  
8  
9  
10
```

In the above example, the loop prints numbers from 1 to 10 without any interruption. Now let's interrupt the loop with the 'break' statement.

We have 10 numbers, but we want the loop to print numbers only from 1 to 7, and when the number becomes 8, we will exit the loop.

```
for i in range(1,11):  
    if i==8: # when i is equal to 8, exit the loop  
        break  
    print(i)  
print("we came out of the loop")
```

Output –

```
1  
2  
3  
4  
5  
6  
7  
we came out of the loop
```

In the above example, we used 'break' statement to come out of the loop

when the value of *i* becomes 8, otherwise continue printing the values of *i*. Make sure that you use indentation properly, otherwise you will get an error.

Continue Statement

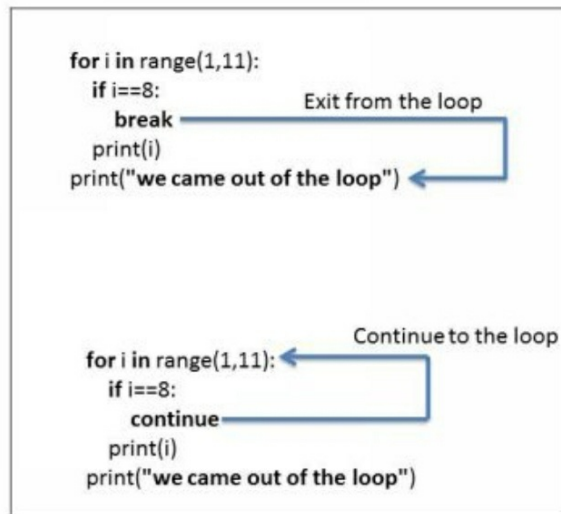
‘Continue’ statement is used to skip a particular iteration and go back to the beginning of the loop. The difference between ‘break’ and ‘continue’ is that ‘break’ exits the loop and ‘continue’ does not exit the loop but skips that iteration and continues the remaining loop. Let’s take the above example again with ‘continue:’

```
for i in range(1,11):
# when i is equal to 8, skips the iteration.
# then continues from next value i.e. 9
    if i==8:
        continue
    print(i)
print("we came out of the loop")
```

Output –

```
2
3
4
5
6
7
9
10
we came out of the loop
```

As you can see, when the value of *i* becomes 8, the loop will skip the statement ‘print(*i*)’ and won’t print 8. And after that, the loop continues and prints the values 9 and 10. Observe the image below.



Predefined Errors

Sometimes, errors happen during the program. This might be caused by a bad code or bad user input. Most of the time, it is the former.

Python immediately ends a program whenever errors are encountered. However, what if you want the show to continue despite these errors?

You might want to know what happens with the other code you have written after the line that produced the error. You want to know if they are also problematic. That is when error handling is useful.

Error handling is a programming process wherein you assume control of the program's errors from Python. Instead of just letting Python close your program, performing error handling can let you run code and continue with the program if an error is encountered.

Chapter 6:

Functions and Modules

Functions

Functions are like containers that store lines and lines of codes within themselves, just like a variable that contains one specific value. There are two types of functions we get to deal with within Python. The first ones are built-in or predefined, the other are custom-made or user-created functions.

Creating Functions

In order for us to create a function, we first need to ‘define’ the same. That is where a keyword called ‘def’ comes along. When you start typing ‘def,’ Python immediately knows you are about to define a function. You will see the color of the three letters change to orange (if using PyCharm as your IDE). That is another sign of confirmation that Python knows what you are about to do.

```
def say_hi():
```

Remember, keep your name descriptive so that it is understandable and easy to read for anyone. After you have named your function, follow it up with parentheses. Lastly, add the friendly old colon to let Python know we are about to add a block of code.

Press enter to start a new indented line. Now, we shall print out two statements for every user who will join the stream.

```
print("Hello there!")
```

```
print("Welcome to My Live Stream!")
```

After this, give two lines of space to take away those wiggly lines that appear the minute you start typing something else. Now, to have this printed out easily, just call the function by typing its name and run the program. In

our case, it would be:

```
say_hi()
```

Output:

Hello there!

Welcome to My Live Stream!

See how easily this can work for us in the future? We do not have to repeat this over and over again. Let's make this function a little more interesting by giving it a parameter. Right at the top line, where it says "def say_hi()"? Let us add a parameter here. Type in the word 'name' as a parameter within the parenthesis. Now, the word should be greyed out to confirm that Python has understood it as a parameter. Now, you can use this to your advantage and further personalize the greetings to something like this:

```
def say_hi(name):  
    print(f"Hello there, {user}!")  
    print("Welcome to My Live Stream!")  
    user = input("Please enter your name to begin: ")  
    say_hi(user)
```

The output would now ask the user regarding their name. This will then be stored into a variable called user. Since this is a string value, say_hi() should be able to accept this easily.

By using 'user' as an argument, we get this as an output:

Please enter your name to begin: Johnny

Hello there, Johnny!

Welcome to My Live Stream!

Now that's more like it! Personalized to perfection. We can add as many lines as we want, the function will continue to update itself and provide greetings to various users with different names.

There may be times where you may need more than just the user's first name.

You might want to inquire about the last name of the user as well. To add to that, add this to the first line and follow the same accordingly:

```
def say_hi(first_name, last_name):  
    print(f"Hello there, {first_name} {last_name}!")  
    print("Welcome to My Live Stream!")  
    first_name = input("Enter your first name: ")  
    last_name = input("Enter your last name: ")  
    say_hi(first_name, last_name)
```

Now, the program will begin by asking the user for their first name, followed by the last name. Once that is sorted, the program will provide a personalized greeting with both the first and last names.

However, these are positional arguments, meaning that each value you input is in order. If you were to change the positions of the names for John Doe, Doe would become the first name, and John would become the last name. You may want to be a little careful with that.

Hopefully, now you have a good idea of what functions are and how you can access and create them. Now, we will jump towards a more complex front of 'return' statements.

“Wait! There’s more?” Well, I could have explained this earlier, but back then, when we were discussing statements, you may not have understood it completely. Since we have covered all the bases, it is appropriate enough for us to see exactly what these are and how these gel along with functions.

Variable Scope

Since global and private variables have been mentioned, this section will discuss variable scopes.

There are two types of variables: global and local.

Global variables are available throughout the program. When using them inside functions, you use the keyword `global`. In some programming languages, global variables are referred to as public variables. Local variables are only available inside a code block where it was used. The previous

sections referred to it as private variables for simplification purposes. Local variables can be used by the function that declared or used it and are deleted once the function ends. For accuracy's sake, this book will now refer to global variables as global variables and local variables as local variables. Public and private variables can easily have a different connotation. And this can be confusing once you start dealing with modules.

Default Parameters

We say that a parameter is a type of value that receives the function when it is invoked. A function can receive one or more parameters (these must be separated by a comma "," to be invoked).

Example: We want to create a program that greets a specific person or in general.

We write our code:

```
1 def Hello(name="Paula"):
2     # This program will greet people
3     print("Hello " + name + "!")
4
5 Hello("Word")
6 Hello()
7
```

That will print:

```
Hello Word!
Hello Paula!
[Finished in 0.3s]
```

Variable Length Arguments

When invoking this function, these values to be sent are called "arguments"

and are divided into several types:

By name: When invoking the function in the arguments, we must indicate the value that each parameter will have from the name.

Example: Let's suppose that we want to calculate half of a numerical quantity.

We write our code:

```
1 def write_half():
2     half = (a + b) / 2
3     print(f"The half part of {a} and {b} its {half}")
4     return
5
6 a = 10
7 b = 5
8 write_half()
9 print("Finish program")
10
```

It will print:

```
The half part of 10 and 5 its 7.5
Finish program
[Finished in 0.3s]
```

By position: We can see in the following example that the program will take the data according to the order in which they are received:

```
1  def sum(a, b):
2      return a + b
3  sum(15, 2)
4
```

Importing Modules

Writing a huge program in one file can be cumbersome. A regular program like a calculator can have hundreds of functions, and each of those functions can contain five to nine statements. If one statement equals one line of code, writing a small program with a hundred functions can make you deal with 500 to 900 lines of codes.

What is a module anyway? A module is simply a Python (.py) file that consists of python code. In programming, the common hierarchy is this:

Program > Modules > Functions > Statements > Variables/Expressions/Data

A program contains modules. A module contains functions. A function contains statements. A statement contains variables, expressions, data, and data. Depending on the complexity of your program, that hierarchy can easily change.

The question now is, “How to create a module?” Creating a module in Python is simple. You just need to write or paste all the functions and statements you want to have, and save it as a *.py file. That is it. You have a new module.

Note that a module can use other modules. To use a module, you need to import them into the program using the **import** keyword.

For example, here is a function inside your module. And this module will be saved as sample.py.

```
def sampleFunction():
```

```
    print("Hello World")
def sampleFunction2():
    print("Hello Again")
```

Here is how to use the sampleFunction function from the sample.py module.

```
>>> import sample
>>> sample.sampleFunction()
Hello World
>>> _
```

Note that the module must be in the same directory as the program/main module (or the one importing it) for the module to be imported. If not, you will receive an error.

```
>>> import nonExistingModule
```

Traceback (most recent call last):

```
  File "<stdin>", line 1, in <module>
ModuleNotFoundError: No module named 'nonExistingModule'
>>> _
```

When you use the import keyword, you will have access to all the functions inside a module. Importing a module this way will make Python treat the functions in the module as methods of the module object. In simpler terms, you will need to mention the object and use the accessor operator (.) to call the function.

If you want to use only one or a specific number of functions from the module and integrate them in the current module as if they are defined in it, you can use the **from** keyword together with import. For example:

```
>>> from sample import sampleFunction
>>> sampleFunction()
Hello World
>>> _
```


You can import multiple functions by doing this:

```
>>> from sample import sampleFunction, sampleFunction2
```

```
>>> sampleFunction2()
```

Hello Again

```
>>> _
```

If you want to use the “from” keyword to get all the function from a module, you can use the asterisk or wildcard operator (*).

```
>>> from sample import *
```

```
>>> sampleFunction()
```

Hello World

```
>>> _
```

Note that you will get an error if you try to call a function without the module object and accessor if you only use import.

```
>>> import sample
```

```
>>> sampleFunction()
```

Traceback (most recent call last):

```
File "<stdin>", line 1, in <module>
```

NameError: name 'sampleFunction' is not defined

```
>>> _
```

Creating Modules

To create a module in Python, we don't need a lot; it's straightforward.

For example: if you want to create a module that prints a city, we write our code in the editor and save it as "mycity.py".

Once this is done, we will know that this will be the name of our module (omitting the .py sentence), which will be assigned to the global variable `__city__`.

This is a straightforward code designed for users of Python 2.

The print function is not in parentheses, so that's the way this Python version handles that function.

Useful Built-In Functions and Methods

The interpreter in Python is equipped with several functions that are almost always handy for use.

These functions are known as built-in functions. For instance, the `print()` function is used to print a specific object to the text stream file or the standard output device like the monitor.

Listed below are the methods in Python programming and their associated functions.

- **Python `abs()`** — Used to return the absolute value of a number.
- **Python `all()`** — Returns as true when all the elements in an iterable are true.
- **Python `any()`** — It checks if any of the elements of an iterable are true.
- **Python `ascii()`** — It returns the string containing printable representation.
- **Python `bin()`** — Used to convert integers into a binary string.
- **Python `bool()`** — It converts a specific value into boolean.
- **Python `bytearray()`** — Used to return arrays of specific byte size.
- **Python `bytes()`** — Used in returning an immutable bytes object.
- **Python `callable()`** — Used in checking whether or not an object is callable.

- **Python chr()** — Used to return a character, usually a string, from an integer.
- **Python classmethod()** — It returns the class method for a specific function.
- **Python compile()** — Used to return a Python code object.
- **Python complex()** — It is used to create a complex number.
- **Python delattr()** — Used to delete the attribute from an object.
- **Python dict()** — Used in creating a dictionary.
- **Python dir()** — Used in attempting to return attributes of an object.
- **Python divmod()** – Used to return a tuple of quotient and remainder.
- **Python enumerate()** — Used to return an enumerate object.
- **Python eval()** — Used to execute Python codes within the program.
- **Python exec()** — Used to run dynamically created programs.
- **Python filter()** — Used to construct an iterator from elements that are true.
- **Python float()** — Used to return the floating-point number from a number or string.
- **Python format()** — It returns a formatted representation of a value.
- **Python frozenset()** — It is used to return an immutable frozenset object.
- **Python getattr()** — Used to return the value of the named attribute of an object.
- **Python globals()** — Used to return the dictionary of the current global symbol table.

- **Python hasattr()** — Used to return whether the object has named attribute or not.
- **Python hash()** — Used to return the hash value of an object.
- **Python help()** — Used in invoking the built-in Help System.
- **Python hex()** — Used to convert integer into hexadecimal.
- **Python id()** — Used in returning the identity of an object.
- **Python input()** — Used to read and return a line of string.
- **Python int()** — Used in returning an integer from a number or string.
- **Python isinstance()** — Used to check if an object is an instance of a class.
- **Python isinstance()** — Used to check if an object is the subclass of a class.
- **Python iter()** — Used to return the iterator for an object.
- **Python len()** — Used to return the length of an object.
- **Python list() Function** — Used to create a list in Python.
- **Python locals()** — Used in returning the dictionary of a current local symbol table.
- **Python map()** —used to apply function and returns to a list.
- **Python max()** — Used in returning the largest element.
- **Python memoryview()** — Used to return the memory view of an argument.
- **Python min()** — Used in returning the smallest element.
- **Python next()** — Used to retrieve the next element from an iterator.
- **Python object()** —used to create a featureless object.
- **Python oct()** —used in converting integers to octal.

- **Python open()** — Used to return a file object.
- **Python ord()** — Used in returning the unicode code point for a unicode character.
- **Python pow()** — Used to return x to the power of y.
- **Python print()** — Used in the printing of a specific object.
- **Python property()** — Used to return a property attribute.
- **Python range()** — Used in returning a sequence of integers between start and stop.
- **Python repr()** — Used to return a printable representation of an object.

Functions and Imports Exercises

Functions are code blocks that are given an identifier. This identifier can be used to call the function. Calling a function makes the program execute the function regardless of where it is located within the code. To create a function, you need to use the “def” keyword. Def basically defines, and when you use it to create a function, you can call it as defining a function. For example:

```
>>> def doSomething():  
    print("Hello functioning world!")  
  
>>> doSomething()  
  
Hello functioning world!  
  
>>> _
```

Creating and calling a function is easy. The primary purpose of a function is to allow you to organize, simplify, and modularize your code. Whenever you have a set of code that you will need to execute in sequence from time to time, defining a function for that set of code will save you time and space in your program. Instead of repeatedly typing code or even copy-pasting, you simply define a function. The time has come for us to dive into a complex world of functions where we don't just learn how to use them effectively, but

we also look into what goes on behind these functions and how we can come up with our very own personalized function.

Chapter 7:

Object-Oriented Programming: Classes and Instances

What Are Classes?

We define classes as models in which we are going to build our objects; a mechanism of classes is in charge of creating them with the least amount of new syntax.

When a new class is created, a new object type is created, which allows creating new instances of this type (each instance of the class can contain attached attributes to be able to maintain its state, these instances could also contain methods previously defined by its class to modify its state). Its syntax is as shown below:

```
1  class ClassName:
2      <statement-1>
3      .
4      .
5      .
6      <statement-N>
7
```

Both class and function definitions must be executed before any effect is performed. We could place a class definition within a function or an if-type statement.

When a class is defined, a new namespace is created and used locally. This refers to the fact that, from this, the next assignments given to the local variables will be entered in this new namespace

Creating Classes

One of the things that you will really enjoy working on when it comes to Python is the organization that comes with it. This kind of coding language is going to spend time dividing everything up into classes and objects. This allows for more organization, ensures that every part of the code has a place and won't get lost, and really makes it easier for the beginner programmer to get things done without as much hassle.

Writing a class sounds more complicated than it really is, so let's stop here and look at an example of how you would write this out in Python. Then we can discuss what the parts mean and why they are important. A good example of creating a class with Python includes the following:

```
class Vehicle(object):  
  
    #constructor  
  
    def __init__(self, steering, wheels, clutch, breaks, gears):  
  
        self._steering = steering  
  
        self._wheels = wheels  
  
        self._clutch = clutch  
  
        self._breaks = breaks  
  
        self._gears = gears  
  
    #destructor  
  
    def __del__(self):  
  
        print("This is destructor....")  
  
        #member functions or methods  
  
    def Display_Vehicle(self):  
  
        print('Steering:' , self._steering)  
  
        print('Wheels:', self._wheels)  
  
        print('Clutch:', self._clutch)
```



```
print('Breaks:', self._breaks)
print('Gears:', self._gears)
#instantiate a vehicle option
myGenericVehicle = Vehicle('Power Steering', 4, 'Super Clutch', 'Disk
Breaks', 5)
myGenericVehicle.Display_Vehicle()
```

To see how this is going to work well for your needs and how the class can be set up, you should open up your compiler and type in the code that we have above.

This will ensure that you are going to be able to test out the code and see which kind of class you have been able to create.

This is a simple code that ensures we have the right objects that go into the different classes that we assigned and will make sure that when we open up that class, it is all going to open up and work the way that we would like.

Class Variables and Instance Variables

How to create a method using the student class.

```
>>> student1 = student("Johnny", "Male", "18", "1000121")
```

```
>>> student1.details()
```

```
Name: Johnny
```

```
ID: 1000121
```

```
Age: 18
```

```
Gender: Male
```

```
>>> _
```

The example created an object by using the class name and providing arguments to the parameters listed in the initializer method. By the way, the “self” identifier is not a keyword. It is a convention that you can use to refer to the object or the object that called the method. More about this will be explained later.

In the case of the init method, the self refers to the object itself. Also, since the value of self is already established as the object, you do not need to give it an argument. Just proceed with the other parameters.

Objects created from a class are called *instances*.

Classes and Instances

Classes and instances are handy to use when your program requires objects that contain similar number and kinds of variables and use similar functions. Most Content Management Systems used on websites (e.g., WordPress) make use of classes and instances. One of the classes that WordPress has is wp_post. Instances of this class often contain a single blog or page post in a WordPress website.

Some of the attributes that the wp_post has are id, post_author, post_title, post_date, and post_content. The wp_post class is written inside the wp-includes/class-wp-post.php module. By the way, WordPress is written in PHP.

Class Methods and Static Methods

Class methods can be defined as a method type which is called on a class instead of an instance. Class methods are mainly used as a part of an object meta-model. That is, an object meta-model is created for every class that is defined in an instance.

Methods in classes can be categorized into two: class and static methods. By default, all methods are automatically set as class methods.

The main difference between the two is that class methods automatically pass the calling object as an argument by reference while static methods do not. This is the reason it is required to have a self or any parameter that will receive the reference in class methods.

Without a parameter to catch the reference to the calling object, Python will return an error. For example:

```
>>> def sampleClass():
    def sampleFunction():
        print("Nothing")
```

```
>>> x = sampleClass()
```

```
>>> x.sampleFunction()
```

Traceback (most recent call last):

```
File "<stdin>", line 1, in <module>
```

TypeError: sampleFunction() takes 0 positional arguments but 1 was given

```
>>> _
```

As you can see, Python detected that the class method call automatically sent a positional argument in form of the self object reference. And since the method `sampleFunction()` did not have any parameter to catch the argument, Python returned a `Traceback TypeError`.

Static methods, on the other hand, do not pass the any argument by default, so you can create methods that do not contain parameters.

In order to define class and static methods you need to use the decorator operator (`@`). For example:

```
>>> class classA():
```

```
    @classmethod
```

```
    def methodA():
```

```
        print("This is a Class Method.")
```

```
    @staticmethod
```

```
    def methodB():
```

```
        print("This is a Static Method.")
```

```
>>> class classB(classA):
```

```
    pass
```

```
>>> x = classB()
```

```
>>> x.methodB()
```

This is a Static Method.

```
>>> x.methodA()
```

Traceback (most recent call last):

```
File "<stdin>", line 1, in <module>
```

TypeError: sampleFunction() takes 0 positional arguments but 1 was given

```
>>> _
```

The decorator operator (`@`) has other functionalities other than defining methods as static or class. However, it will not be discussed in this book.

Chapter 8:

Objects-Oriented Programming: Inheritance, Child Classes, and Special Methods

Object-Oriented Programming or OOP is a paradigm of programming language which uses in its interactions the concepts of objects to develop a program. In other words, it is a way of structuring our code, which organizes and reuses it effectively.

OOP differs from traditional structured programming because one of its main objectives is to seek the processing of input data to obtain other output data.

Inheritance and Multiple Inheritance

This happens when classes relate to each other and, little by little, form a type of classification hierarchy. Objects inherit the properties and behavior of all these classes. Inheritance is responsible for organizing and facilitating polymorphism and encapsulation, allowing objects to define themselves and be creators as specialized types of existing objects, allowing sharing and extending their behavior without the need to be implemented again. When an object inherits more than one class, it is told that it contains multiple inheritances, becoming an object of technical complexity.

To help make more sense out of these inheritances, how they work, and how they can help to keep your code clean and tidy and save you time, let's take a look at an example of how they look in your code:

```
#Example of inheritance  
  
#base class  
  
class Student(object):  
  
    def __init__(self, name, rollno):
```

```

self.name = name
self.rollno = rollno
#Graduate class inherits or derived from Student class
class GraduateStudent(Student):
def __init__(self, name, rollno, graduate):
Student__init__(self, name, rollno)
self.graduate = graduate
def DisplayGraduateStudent(self):
print"Student Name:", self.name)
print("Student Rollno:", self.rollno)
print("Study Group:", self.graduate)
#Post Graduate class inherits from Student class
class PostGraduate(Student):
def __init__(self, name, rollno, postgrad):
Student__init__(self, name, rollno)
self.postgrad = postgrad
def DisplayPostGraduateStudent(self):
print("Student Name:", self.name)
print("Student Rollno:", self.rollno)
print("Study Group:", self.postgrad)
#instantiate from Graduate and PostGraduate classes
objGradStudent = GraduateStudent("Mainu", 1, "MS-Mathematics")
objPostGradStudent = PostGraduate("Shainu", 2, "MS-CS")
objPostGradStudent.DisplayPostGraduateStudent()

```

When you type this into your interpreter, you are going to get the results:

```
('Student Name:', 'Mainu')
```

```
('Student Rollno:', 1)
```

```
('Student Group:', 'MSC-Mathematics')
```

```
('Student Name:', 'Shainu')
```

```
('Student Rollno:', 2)
```

```
('Student Group:', 'MSC-CS')
```

You are able to go through this process and use the base or parent class as many times as you would like.

It is easy to go down the line as many times as your code asks for, making changes along the way without worrying about ruining up any of the parent code along the way.

As long as you do not work on a circular inheritance here, you will be fine to keep adding in the base classes down to the child class, and you will be able to get things to change and stay the same as much as you want as well.

Importing Classes

To import classes, you can treat classes as if they are functions from a module. Of course, save the classes on a module first. By the way, some of the previous examples took advantage of importing, classes if you have noticed.

For example:

```
>>> class staff():
    name = ""
    gender = ""
    age = ""
    employee_id = ""
    def setDetails(self, name, gender, age, employee_id):
        self.name = name
```

```
self.gender = gender
self.age = age
self.employee_id = employee_id
```

```
def getDetails(self):
    print("Name: " + self.name)
    print("Gender: " + self.gender)
    print("Age: " + self.age)
    print("ID: " + self.employee_id)
```

```
>>> class supervisor(staff):
    team_members = []
```

```
>>> class cashier(staff):
    team_supervisor = ""
```

```
>>> class waiter(staff):
...     team_supervisor = ""
```

```
>>> _
```

Save this to class_staff.py. Then import it just like a module.

```
>>> import class_staff
```

```
>>> x = class_staff.cashier()
```

```
>>> x.getDetails()
```

Name:

Gender:

Age:

ID:

```
>>> _
```

However, that is a practice that may form a habit, a bad one. There are times when the codes for the parent and child classes are saved in different modules. When you only import the child class in that situation, it might result in an error.

For example, say that the code for the staff() class is saved on module1.py and the supervisor(staff) class is saved on module2.py.

```
>>> from module2 import supervisor
```

Traceback (most recent call last):

```
File "<stdin>", line 1, in <module>
```

```
File "<C:\Python\module2.py", line 1, in <module>
```

```
class supervisor(staff):
```

NameError: name 'staff' is not defined

```
>>> _
```

To solve this, you must make sure that you import the parent classes first even if they are in the same module to prevent any error from popping up.

```
>>> from module1 import staff
```

```
>>> from module2 import supervisor
```

```
>>> x = supervisor()
```

```
>>> x.getDetails()
```

Name:

Gender:

Age:

ID:

```
>>> _
```

By the way, note that you should name modules as if you are creating an identifier in Python. This means that letters, numbers, and underscores are the only characters that you are allowed to use in the module's file name. Here is what will happen if you do not follow this:

```
>>> import class-staff
```

```
File "<stdin>", line 1
```

```
    Import class-staff
```

```
        ^
```

```
SyntaxError: invalid syntax
```

```
>>> _
```

Python Special Methods

Unlike some of the other types of methods, special methods are quite distinct to every programming language. Thus, while one language may be capable of supporting no special method, others may support some or all of them. The compiler in a programming language may generate special methods by default, or a programmer may be granted the allowance to define special methods optionally. Although many special methods can't be directly invoked, the compiler is made to generate specific codes with which it invokes them at the right times. These operators are commonly used in loops to check for repeated variables or to see if an element is stored within others.

Operator In: This operator will return a 'True' if an element is stored within another.

E.g. `A = [14, 21]` `14 in A`

The result that returns will be true because the value 14 is positioned in A.

Operator Not In: This operator will return a True if an element is not stored inside another element.

E.g. `A = [5, 8]` `8 not in A`.

The result that returns will be False because the value 8 is positioned in A.

Operator Is: This operator will return a True if its values stored in variables are the same.

E.g. X= 14; Y= 14. → X is Y

The result that returns will be True because both variables contain the same stored value.

Operator Not Is: This operator will return a True if the values that are stored are not the same.

E.g. X= 20; Y=31. → X not is Y

The result that is going to return will be True because both variables contain stored different values; therefore, they are different.

Classes and Methods Exercises

As each object has certain attributes, they also have certain behaviors, all of which also belong to it.

To do this, the functions are created within each class called methods. Let's go back to the example of football. Suppose that the team has two actions or methods.

The first one is to train, the other one is to play, and both of them can be activated at any time.

However, if you understand the concept of attributes well, this will be easy-peasy for you since this is only about including certain behavior to the object. Let's see an example so you can understand everything better.

Example:

```

1  class car:
2      def __init__(self, color, dors, country):
3          self.color=color
4          self.dors=dors
5          self.country=country
6
7      def state(self, bol):
8          if bol:
9              print("The car is on")
10         else:
11             print("The car is off")
12
13     ford_blue=car("blue",2,"USA")
14     ford_blue.state(True)
15     toyota_red=car("red",4,"Japan")
16     print("We create a car")
17
18

```

We can see a pretty rich program since we apply the knowledge we have in the builder, attributes, and conditionals. The first thing that we can notice is the creation of the class in the usual way, then we used the builder to initialize the variable color, doors, and country.

The next step was the creation of our first method, state, which is designed to start the car or not with the help of an if-else.

Finally, two cars were created, the ford_blue and the toyota_red. We did not only create that, but we also turned on the ford_blue since we got access to the state method for which we passed the True parameter, and it caused the car to turn on. Extrapolate the concepts to reach everyday life because that is its objective. Similarly on how the family inherited the grandfather's green eyes is how the Python inheritance works. An object can inherit the attributes and methods of a higher class.

Let's look at the following example:

```

1  class car:
2      def __init__(self,color,dors,country):
3          self.color=color
4          self.dors=dors
5          self.country=country
6
7      def state(self, bol):
8          if bol:
9              print("The car is on")
10             else:
11                 print("The car is off")
12
13  class carElec(car):
14      def __init__(self,color,dors,country, electronicConsume):
15          car.__init__(self,color,dors,country)
16          self.elctronicC=electronicConsume
17
18  Tesla=carElec("black",2, "Italy", "20Kw")
19  Tesla.state(False)
20

```

As users, we can notice that the class carElec does not have methods but has similar behaviors, not to say identical. Also, the builder of a class called car is used.

First, we have to start with the definition of the class car. This explanation has already been said before, but it is never excessive to repeat it because this is one of the roughest chapters as readers may find object-oriented programming very complicated.

After the state method, another class will be created, the one for electric cars, in the program called carElec. But here, we get certain differences regarding the car class since it has some parentheses within, which has the car class. What does this mean?

It means that the carElec class inherited everything from the car class, and with this, we mean that it inherited both attributes and methods.

Finally, when creating the Tesla variable, we create it in a totally simple way since it is created as an object of type carElec. The necessary parameters are passed to it and it is done, so we can already use the methods within its class and everything it inherits from the "father" class.

And right now, you may be wondering about its functionality. What will it be used for? Is it really useful?

Of course, it is since one of the great features of python is object-oriented programming for a whole series of benefits like reusability.

With the implementation of classes and this programming model as objects with a class, you can create endless objects without the need to make thousands of lines of code.

As the practice makes the master, we will make one more example, a program that will be responsible for creating a number of objects. This amount will depend on the user, plus it will have several methods which will be within the builder.

```
1 class person:
2     def __init__(self, name, age, sport):
3         self.name=name
4         self.sport=sport
5         self.age=age
6         self.activity()
7     def activity(self):
8         if self.sport=="Soccer":
9             print(self.name+" practice "+self.sport+" and the favorite team is the Real
10                Madrid")
11        elif self.sport=="Baseball":
12            print(self.name+" practice "+self.sport+" and the favorite team is the
13                Yankees")
14        elif self.sport=="Basketball":
15            print(self.name+" practice "+self.sport+" and the favorite team is Cleveland
16                ")
17        else:
18            print(self.name+" practice "+self.sport)
19
20 number=input("Number of objects:")
21 for x in range(int(number)):
22     name=input("name:")
23     age=input("age:")
24     sport=input("sport:")
25     item=person(name, age, sport)
```

As you can see, activity is a method of person which uses a combination of if, elif and else, which allow us to filter the activity performed by each person and thus print on console what does each object or person, better said.

Finally, the variable number is created, which is an input in which the user will decide how many objects to create, then number becomes an integer and will allow us to use it for cycle, since x is going to iterate in the range from 0 to 4, and in each one of them, the values desired by the user will be assigned to the variables name, age, and sport to finally create the item object that is a person.

Chapter 9:

Files

A Python file is a set of bytes (information) which has the function of storing data, which are organized in a format specifically.

The structure of the files is composed in the following way:

- **Header:** Data about what the file contains (name, size, type)
- **Data:** Body of the file and content written by the editor
- **End of file:** Sentence that indicates the end of the file

Our file would look like this:



Programs are made with input and output in mind. You input data to the program, the program processes the input, and it ultimately provides you with an output. For example, a calculator will take in the numbers and operations you want. It will then process the operation you wanted. And then, it will

display the result to you as its output. There are multiple ways for a program to receive input and to produce output. One of those ways is to read and write data on files. To start learning how to work with files, you need to learn the `open()` function.

The `open()` function has one *required* parameter and two *optional* parameters. The first and required parameter is the file name. The second parameter is the access mode. And the third parameter is buffering or buffer size. To practice using the `open()` function, create a file with the name `sampleFile.txt` inside your Python directory.

Try this sample code:

```
>>> file1 = open("sampleFile.txt")
>>> _
```

Note that the file function returns a file object. The statement in the example assigns the file object to variable `file1`.

The file object has multiple attributes, and three of them are:

- **Name:** This contains the name of the file.
- **Mode:** This contains the access mode you used to access the file.
- **Closed:** This returns `False` if the file has been opened and `True` if the file is closed. When you use the `open()` function, the file is set to open.

Now, access those attributes.

```
>>> file1 = open("sampleFile.txt")
>>> file1.name
'sampleFile.txt'
>>> file1.mode
'r'
>>> file1.closed
False
```



```
>>> _
```

Whenever you are finished with a file, close them using the `close()` method.

```
>>> file1 = open("sampleFile.txt")
```

```
>>> file1.closed
```

```
False
```

```
>>> file1.close()
```

```
>>> file1.closed
```

```
True
```

```
>>> _
```

Remember that closing the file does not delete the variable or object. To reopen the file, just open and reassign the file object. For example:

```
>>> file1 = open("sampleFile.txt")
```

```
>>> file1.close()
```

```
>>> file1 = open(file1.name)
```

```
>>> file1.closed
```

```
False
```

```
>>> _
```

Opening, Reading, Writing Text Files

Reading From a File

Before proceeding, open the `sampleFile.txt` in your text editor. Type "Hello World" in it and save. Go back to Python. To read the contents of the file, use the `read()` method. For example:

```
>>> file1 = open("sampleFile.txt")
```

```
>>> file1.read()
```

```
'Hello World'
```

```
>>> _
```

File Pointer

Whenever you access a file, Python sets the file pointer. The file pointer is like your word processor's cursor. Any operation on the file starts at where the file pointer is. When you open a file and when it is set to the default access mode, which is "r" (read-only), the file pointer is set at the beginning of the file. To know the current position of the file pointer, you can use the `tell()` method. For example:

```
>>> file1 = open("sampleFile.txt")
```

```
>>> file1.tell()
```

```
0
```

```
>>> _
```

Most of the actions you perform on the file move the file pointer. For example:

```
>>> file1 = open("sampleFile.txt")
```

```
>>> file1.tell()
```

```
0
```

```
>>> file1.read()
```

```
'Hello World'
```

```
>>> file1.tell()
```

```
11
```

```
>>> file1.read()
```

```
"
```

```
>>> _
```

To move the file pointer to a position you desire, you can use the `seek()` function. For example:

```
>>> file1 = open("sampleFile.txt")
```

```
>>> file1.tell()
```

```
0
>>> file1.read()
'Hello World'
>>> file1.tell()
11
>>> file1.seek(0)
0
>>> file1.read()
'Hello World'
>>> file1.seek(1)
1
>>> file1.read()
'Hello World'

>>> _
```

The `seek()` method has two parameters. The first is `offset`, which sets the pointer's position depending on the second parameter. Also, argument for this parameter is required.

The second parameter is optional. It is `whence`, which dictates where the “seek” will start. It is set to 0 by default.

If set to 0, Python will set the pointer's position to the offset argument.

If set to 1, Python will set the pointer's position relative or in addition to the current position of the pointer.

If set to 2, Python will set the pointer's position relative or in addition to the file's end.

Note that the last two options require the access mode to have binary access. If the access mode does not have binary access, the last two options will be

useful to determine the current position of the pointer [seek(0, 1)] and the position at the end of the file [seek(0, 2)]. For example:

```
>>> file1 = open("sampleFile.txt")
```

```
>>> file1.tell()
```

```
0
```

```
>>> file1.seek(1)
```

```
1
```

```
>>> file1.seek(0, 1)
```

```
0
```

```
>>> file1.seek(0, 2)
```

```
11
```

```
>>> _
```

Writing to a File

When writing to a file, you must always remember that Python overwrites and not insert file. For example:

```
>>> x = open("sampleFile.txt", "r+")
```

```
>>> x.read()
```

```
'Hello World'
```

```
>>> x.tell(0)
```

```
0
```

```
>>> x.write("text")
```

```
4
```

```
>>> x.tell()
```

```
4
```

```
>>> x.read()
```

```
'o World'  
>>> x.seek(0)  
  
0  
>>> x.read()  
  
'texto World'  
  
>>> _
```

You might have expected that the resulting text will be “textHello World”. The write method of the file object replaces each character one by one starting from the current position of the pointer.

Buffer

A buffer is a temporary storage area in the ram (or just a simple file given some use) that is responsible for containing a data fragment of the sequence of files in the operating system until it is possible to use it.

The storage in a buffer is very useful when we do not know the file size with which we will work. If, in a hypothetical case, the file size exceeds the memory of the computer, the processing unit will not be able to work properly. The size of the buffer is responsible for indicating the amount of data that can be stored at the same time until the moment it is used.

The function `io.DEFAULT_BUFFER_SIZE` indicates the size of the platform by default. Example:

```
1 import io  
2 print("Default buffer size:",io.DEFAULT_BUFFER_SIZE)  
3 file=open("thisismynewfile.txt",mode="r",buffering=4)  
4 print(file.line_buffering)  
5 file_contents=file.buffer  
6 for line in file_contents:  
7     print(line)  
8
```

Binary Files

A binary system has only two symbols, 1 and 0, which are referred to as bits.

All the numbers are represented by combining these two symbols. Binary systems are ideal for electronic devices because they also have only two states, on or off. In fact, all electronic devices are based on the binary number system. The number system is positional, which means the position of symbols determines the final value. Since there are two symbols in this system, the system has a base of 2.

The sole purpose of input and output systems is to convert data to and from binary system to a form that makes better sense to the user. The first bit from the left side is called Most Significant Bit (MSB), while the first bit from the right is called the Least Significant Bit (LSB).

Deleting and Renaming

For practice, you need to perform the following tasks:

Create a new file named test.txt.

Write the entire practice exercise instructions on the file.

Close the file and reopen it.

Read the file and set the cursor back to 0.

Close the file and open it using append access mode.

Add a rewritten version of these instructions at the end of the file.

Create a new file and put similar content to it by copying the contents of the test.txt file

Chapter 10:

Intermediate and Advanced Concepts

Lambda Functions

A lambda type function is a special type of function (it is considered part of the functions that are already predefined in Python).

This type of function is characterized by a syntax that is considered "exclusive" or different from the others and is also characterized by allowing you to create functions more quickly.

These functions are capable of executing an expression and returning its result.

In their structure, they can take parameters of optional form (those that we saw when we spoke of the statement def); nevertheless, these have their own restrictions.

Syntax

The syntax of the lambda function in Python is considered very simple and is as follows: the reserved word "lambda" is used followed by the respective arguments and finally separated from the expression by a colon ":"

Example: We need to do the addition of the variable x with the variable y.

```
1  sum = lambda x, y: x + y
2  #This example sums variables.
3
```

To put it more clearly, this is equivalent to writing a "def" function as

follows:

```
1 def sum(x,y):  
2     return x + y  
3     #This example sums variables.  
4  
5
```

We use the function type lambda when we need to use a function (which has no name) only for a short period in our program. Lambda functions are used together with integrated functions such as: map(), filter()

Dictionary Handling

As we have seen with the last two data structures, here we also find a structure; in this case, each word has a key, working like a dictionary, but to understand this a little better, it will be important to see a simple example.

```
variable.py ×  
  
1  dicc={"zero":0, "one":1, "two":2}  
2  a=dicc["one"] # a=1  
3  print(a)  
4  dicc["try"]="ok" #{"zero":0, "one":1, "two":2, "try":"ok"}  
5  print(dicc)  
6  del dicc["one"] #{"zero":0, "two":2, "try":"ok"}  
7  print(dicc)  
8
```

As we could see in the previous example, the first thing we do is to declare our variable as a dictionary, and for it, we use the curly brackets; as we can see, each key has its pair, for example "one" is the key, and the integer 1 goes with it. Then we print the value of a, on screen, then we proceed to add an item more in the dictionary, and the only thing that is done is to place a value to a new key, then print the dictionary and verify that no error has been instructed and knowing which is the key you want to remove. An important detail of the dictionaries is that they do not admit repeated keys inside their items.

Threading

This is a very helpful tool in a lot of the applications that you are going to use C++ for. It allows processes to run at the same time, rather than having them sit in a queue and wait their turn. This can speed up the process in which things get done and are useful with a lot of the gaming that you may want to do with your program. Some of the best libraries that are out there that can help C++ do better with some of the multithreading that you want to do will include:

- **Standard library for C++ 11 and 14:** Multithreading is one of the areas where the standard library with C++ has been updated to include support for this. Some of the basic functionality of this was added to C++ 11, and some more of the classes were then added to the C++14 option. The other options are good ones to work with if you have not started working with the C++11 version.
- **Boost Thread:** This library is going to be implementation that comes in a different name, that makes the functionality of multithreading possible for those who have an older compiler of C++.
- **POCO:** This library is going to come with a few different classes that can help us to work with this process. This is going to include some of the higher-level abstractions for managing tasks.

Packages and the Pip Package Manager

Modules are files with codes in them, and similarly, packages are directories, or folders, that contain multiple files within them. We use packages to ensure that we organize our files accordingly. If you were to create hundreds of files and a good chunk of them belonged to calculations, you can create a package directory within PyCharm and Python with an appropriate name and move all such files within this new folder.

To create a package file, right-click on the project name and choose Python package. Instantly, you will notice that it has one special file within it by default. It is the `__init__` file that basically initializes the package and allows Python to know what this is.

Now, let us add a module to this newly created package. We shall call this 'test1' and give it a function within it. Remember, this file needs to be created in the new package we created.

```
def test():  
    print("This is just a test")
```

We gave this as a function for the module. Now, let's browse into one of our old files within the previous folder. If you like, you can create an empty file as well. Now, since we are in a different folder, and we need to import this, things will be slightly different.

If you try and use the previous method of "import test1", it will not work as the directories are different. Instead, we will do this:

```
import test.test1
```

This lets Python know that you are importing a module called test1 from a Python package called test. Now, you will be able to use the function easily.

Now, we have effectively learned everything that a beginner should know about. You are all set to set sail and seek out your own true calling and create programs that are ready to take the world by storm. All it needs is practice and patience.

Libraries

There are many different features that are going to set Pandas apart from some of the other libraries that are out there. Some of the benefits that you are going to enjoy the most will include:

- There are some data frames or data structures that are going to be high level compared to some of the others that you can use.
- There is going to be a streamlined process in place to handle the tabular data, and then there is also a functionality that is rich for the time series that you want to work with.
- There is going to be the benefit of data alignment, missing data-friendly statistics, merge, join, and groupby methods to help you handle the data that you have.

Before we move on from here, we also need to have a good look at what some of the types of data are when it comes to Pandas. Pandas is going to be well suited when it comes to a large amount of data and will be able to help you sort through almost any kind of data that you would like. Some of the data types that are going to be the most suitable for working with the Pandas library with Python will include:

- Any kind of tabular data that is going to have columns that are heterogeneously typed.
- Arbitrary matrix data that has labels for the columns and rows.
- Unordered and ordered time-series data.
- Any other kinds of sets of data that are statistical and observational.

Working with the Pandas library is one of the best ways to handle some of the Python codings that you want to do with the help of data analysis. As a company, it is so important to be able to go through and not just collect data, but also to be able to read through that information and learn some of the trends and the information that is available there. Being able to do this can provide your company with some of the insights that it needs to do better and really grow while providing customer service. There are a lot of different methods that you can use when it comes to performing data analysis. And some of them are going to work in a different way than we may see with Python or with the Pandas library. But when it comes to efficiently and quickly working through a lot of data and having a multitude of algorithms and more that can sort through all of this information, working with the Python Pandas is the best option.

SQLite

The first thing that you need to ask is what SQL is. SQL stands for Structured Query Language, and it is a pretty basic language that you can use in order to interact with the different databases that are on your system. The original version did come out in the 70s, but it really started to see some changes when IBM released a new prototype that released SQL to the world.

This first particular tool was called ORACLE, and it was so successful that

the part of the company that worked with ORACLE was able to break off from IBM and started off on their own. ORACLE is still one of the leaders in the programming language field because it works with SQL and continuously makes it easier for people to learn how to work with the database. When you are working with SQL, you are going to see that the databases are basically groups of information. Some will consider these databases to organize mechanisms that are able to store the information that the user will be able to access at a later time, effectively and efficiently, helping the business to get the information that they need without having to worry about issues coming up. There are a lot of times that you are already using databases like the ones you find in SQL without even realizing that you are using them. For example, if you have ever used a phonebook, you are using a type of database because it is going to contain a lot of information about people that can help you to sort them out, such as the names, phone numbers, and even their addresses. It is even in alphabetical order, so it is easier for everyone to find out the information, also known as the pieces of data, that you need in no time. The SQL database is going to work similarly to help you keep track of the information that your business needs.

JSON Data

There may be times when you would also like to work with the JSON features of C++. This is not as common as some of the other parts that we have talked about above, but it is still important for us to learn how to make this work for our needs. Some of the best C++ libraries that will also support JSON will include:

- **JsonCPP:** This is going to just be a simple JSON reader and writer for the C++ language.
- **JSON Spirit:** This is going to be another reader and writer of JSON. It is going to be built up with the help of the Boost Spirit parser generator, so you will need to work with the Boost library as well because there is already a dependency on this one.
- **POCO::JSON:** POCO is also going to get this one covered and ready to go as well. It has a namespace for JSON if you would like to stay with this family.

Unit Testing

Exploratory testing is going to be a great way to learn more about a program and how it works. It is basically a type of testing that is going to be done without having a good solid plan in place. In this kind of test, you are not looking to get anything out of it or looking to see something specific happen. But you are just taking some time to open up your chosen application and poke around to see how things work.

To have a complete set of manual tests, you have to make a list of all the different features that are available in the application, the different inputs that it is able to accept, and the results that you are expecting. Now, every time that you go through and try to make some changes to your code, it is time for you to go through every single item that is on that list, and double-check that it is working and doing what you want. Of course, this doesn't sound like a lot of fun and can make coding even more complicated for a beginner, which is why you may not use manual testing that much.

This is the place where we need to start talking more about automated testing. This kind of testing is going to be the execution of your test plan, which is the parts of the application that you wish to get tested, the order you want them tested in, and the expected responses you plan to get out of them by a script rather than you or another programmer having to go through and do the work. Python is already going to provide us with some tools, as well as some libraries, to create the automated tests for your applications.

Conclusion

Learning how to get started with computer programming may seem like a huge challenge. You can go with many distinct programming alternatives, but many of them are difficult to learn, will take some time to figure out, and will not always do all the things you need. Many people are afraid they need to be smart or have a lot of education and coding experience before they can make it to the level of coding they want. But even a beginner may get into programming with Python.

The best way to learn programming is really just to challenge yourself. Start putting yourself out there and seeking programming challenges. Think of something that you want to do and then start looking into whether other people have done it. If they have, look into the way that they did it and try to find a way that you could do it yourself. If they haven't, then this gives you a perfect opportunity to be the first to do something. Get your hands dirty and start looking into and learning as much about things having to do with programming as you possibly can.

Programming is not easy. You have a long and arduous journey ahead of you, and the truth about programming is that to be a programmer means to submit yourself to constantly learning. At no point in programming do you ever consider yourself “done with learning.”

Whether you're a beginner or have been in this field for some time, Python has made it so easy to code. The language is English-based, so it's simple to read, and it's got rid of many of the other symbols that make coding difficult for others to read. And since it's a user domain, to make things easier, anyone can make changes and see other codes.

Keep in mind that if you have any questions that may not have been answered in this book, you can always visit the Python website! The Python website contains a lot of material that will help you work with Python and ensure that you are entering your code properly. You can also find any updates that you may need for your Python program in the event that your program is not updating properly, or you need another version of it.

You can work with the program and teach it what you want it to do, and you may even be able to help someone else out if they are not able to get the program to do what they want it to do!

Just remember that you do not need to worry if your Python code doesn't work the first time because using Python takes a lot of practice. The more you practice, the better your code will look, and the better it will be executed. Not only that, but you will get to see Machine Learning in action each time you enter your code!

Thank you for downloading this book.