DEEP LEARNING WITH PYTHON

The ultimate beginners guide to Learn Deep Learning with Python Step by Step

ETHAN WILLIAMS

DEEP LEARNING WITH PYTHON

The ultimate beginners guide to Learn Deep Learning with Python Step by Step

ETHAN WILLIAMS

Deep Learning with Python

The ultimate beginners guide to Learn Deep Learning with Python Step by Step

Table of Contents

CHAPTER ONE: BASIC INTRODUCTION TO THE PYTHON PROGRAMMING LANGUAGE

Introduction

Getting started in writing lines of codes can be a very tough experience. Once a beginner looks at the lines of codes consisting of a group of expressions and terms, he or she is likely to become scared. However, Python programming language will make your experience become sweeter in writing of codes and become a professional within a short period of time.

I believe that this is not your first interaction with the computer. If you are a daily user of a computer then you must have interacted with very many programs that assist you in performing various tasks on the computer. The programs can either be simulating an environment or even automation of tasks on the computer. For instance, if you want to perform tasks like counting the number of occurrence of words in a file or a word document you will require a small program to do so. You might also want to find a specific word in a document and either delete or replace it. All these operations require a suitable program in order to perform any specific action that you need.

In order to perform all these actions in a simple way, Python programming language is the best choice for you to learn and implement. This is a language that is very easy to use and easily available on the accessible operating systems like Mac OS, Windows and UNIX. It is an actual programming language that will help you to program with a lot of ease, develop support structures for larger programs and also eliminate the errors quickly as compared to other programming languages. It is important also to note that it is among the leading very high-level languages having several features like;

- High-level built-in data types
- Very flexible arrays
- Very flexible dictionaries

Before we go further into this programming language, many people get to ask why the program was named after reptile.

Why is it named after a Reptile?

It is important to note that it was never named after python, the reptile animal as many of us assume. It was named after the show “Monty Python’s Flying Circus”. The short history about this programming language is that it was implemented when the inventor Guido van Rossum was reading the published script from that comedy series. Guido was looking for a mysterious, short and a unique name and hence he decided to name the programming language as “Python”.

Short History of Python Programming Language

The program was developed in December 1989 by Guido van Rossum. Guido’s passion and hobby was to write and learn new codes that were available during his time. It is documented that he developed the python programming language while interacting and learning the ABC programming language.

The program was developed during the era of the development of personal computers. The main objective of Guido was to make complex things to become simple to use. In addition, he focused on making many platforms supported as well as to communicate various libraries and multiple file formats. As a result of developing these features, the Python programming language became very popular in the market because people wanted to understand and use the computer in a simple manner.

Over the years, Guido decided to make the programming language open sourced. This became a greater opportunity for the public to assist in the development of the language. In the year 2000, Python 2.0 version was launched and released which became very community-based and transparent in terms of development. In the year 2008, Python 3.0 version was also released which portrayed a unique change to the programming language. These changes were accompanied with great features and compatibility to the new advanced operating systems. However, it limited the user to make choices between the 2.0 version and the 3.0 versions.

In summary, since the programming language was open-sourced, we expect a lot of advancements and developments on the language that will make it simpler and easier to use over the coming years.

Features of Python Programming Language

The Python programming language has endless advanced features that make your programming experience more splendid. For instance, it allows you to divide your larger problem into smaller tasks referred to as “Modules” in order to solve them quickly . In addition, the modules are universal in the language that they can also be used to solve other problems in the language. On the other end, the programming language also offers standard modules that you can use as basic skeletons of developing your own modules. Examples of such standard modules include; sockets, GUI interfaces, system calls and file Input/output.

This programming language <u>was designed to be interpretative</u> . This means that it does not require any compilation and linking of codes as compared to other programming languages. This feature helps the programmer to save a lot of time during the program development process. The design of the programming language also helps the programmer to interact freely with the programs in a way that he or she can get to write simple codes, test the functions of the codes and run the program efficiently. This feature makes this programming language to be compared to a "handy desk calculator".

It makes <u>your program simple to write and proofread</u> . As compared to other programming languages, writing and reading of the programs in Python is easy and very compact. The following are simple reasons that allow us to achieve this feature;

- The written programs do not require any variables, arguments and declarations.
- The built-in high-level data types enable us to write very complex operations in a single and a simple statement.
- There are no uses of opening and closing brackets since the grouping of statements is achieved by indenting of the statements.

The Python programming language <u>is very extensible</u> . This means that it is very easy and cheap to link programs in Python. In other words, linking a module or a built-in function to the interpreter can be done at a very high speed. This is usually achieved by critical operations that operate in the binary formats. On the other end, once you become a professional in this language you will be able to link programs to the interpreter and use it as an extension to other programming languages such as C programming language. The linked programs can also be used as commands when they are extended to other programming languages.

The Python programming languages <u>is equipped with advanced libraries</u> . Since it has been in existence for more than two decades, the language is open-sourced for purposes of further development by the public. In other words, the Python program is freely available for the purposes of personal use and development. For this case, the Python libraries are also easily available and can help you to code or reuse the code as an extension in other programming languages. This makes the program easily available for public manipulation.

Lastly, the python programming language is a community-based language . As a result of being a popular programming language, the language has become a part of the community. In addition, the community using this language is very big. This makes it easier for the users of the language to interact and learn from each other by use of workshops, conferences and even networks. This helps to spread and disseminate new ideas and advancements that do occur in the programming language. There are also other platforms such as websites that do exist in the internet world that can help you learn a lot from experienced personnel.

Advantages of the Python Programming Language

This is one of the best languages that you can choose to begin learning and at the end have a successful career in it. I know that you are going to have a very nice experience in this programming language because it has the following advantages;

i) Easy and Simple to Use, Write and Read

As mentioned before, the lines of codes in Python utilize the style indentation as compared to tons of openings and closing of brackets. As a beginner, this has nothing to scare you but to encourage you because already we are going to deal with simple programming activities not as other languages. In other words, the special characters used in Python are so minimal that you can look at a page of code and not feel intimidated or overwhelmed.

In addition, the programming language uses the white spaces that make it easy for the programmer to proofread the codes that he or she has written. This makes the programming space to be very neat and easy to follow. In summary, it is the best programming language to use and start with as a beginner.

ii) English is the Main Language

This programming language uses English for writing codes. It not only uses English as the language of operation but also uses simple English for expressions and modules. This is not a program that you will go ahead and start using the dictionary to interpret but it’s a program that has that basic structure of simple English commands. In other words, Guido decided to write the program in English so that it can spread all over the world just to make the computer world to become very simple. The program also accommodates simple tenses and phrases that should give you a better idea on what each line of code does. I bet that you will really enjoy using this programming language.

iii) Built-In in Some Operating Systems

This simply means that when you buy and install operating systems like Ubuntu and macOS, the Python platform comes as an already pre-loaded version. This requires the programmer to download or install the text interpreter in order to get started on the Python programming process. In other operating systems like Windows, it is pretty easy to download and install in just a few steps. In summary, python is a very compatible programming language that connects with almost all of the existing operating systems. However, when the python program has not been installed correctly it will not work as it is expected. Therefore, the only caution to be undertaken it to download and install the program strictly as it is recommended in its manual.

iv) Highly Compatible with Other Programming Languages

As mentioned in the introductory part, a module in the Python programming language can be used as an extension in another programming language. In addition, as you grow in python language, you may decide to link your Python program with a function from another programming language and get to achieve complex operations. This feature is highly possible in the python programming language and hence enables you to develop very complex programs in the modern world.

v) Ability to Test by Using Text Interpreter

The interpreter is the one that we download or install during the installation of the Python program. The main objective of the interpreter is to enable Python to read the lines of codes easily. After equipping yourself with an appropriate text interpreter, it means that you are ready to start writing lines of codes. For the case of Python programming language, it is very easy to start coding since it will take in the words and test for you while you still continue to code. This enables the programmer to be able to test the pieces of codes as he or she continues to formulate other codes.

vi) Requires Little Time to Learn

Since it is a very simple language, it becomes relatively simple for a beginner to learn. I can recommend it to be the first language that a person should learn when it comes to programming skills. this is because it builds a skeleton concept in which other languages can be attached as flesh. In other words, it maximizes the use of easy syntax and relatively shorter lines of codes as compared to other programming languages.

In summary, there are other many benefits of using this language that you will find out on your own as you begin to study it carefully. It is an impressive kind of programming language that will fascinate you as you have an encounter with it.

The Negative side of Python Programming Language

While there are many reasons to love and adore the Python, the programming language does have a few weaker sides to watch out for. The limitations of Python include;

i) Relatively Slower Than Other Programming Languages

If you are considering working with a program that contains a lot of speed then I would not recommend python programming language for you. I would like to remind you that python is an interpretative language that does not require the use of a compiler. It is very evident that programming languages that have compilers always have higher speed as compared to those that do not have. On the other end, it is always dependent on what you are actually compiling or translating. In some other cases, python can act faster using benchmarks such as PyPy.

Despite being slow, there is a new word of hope that this problem is being solved by a series of developers. The Python developers are working on a criterion that will make the interpreter of the programming language faster and more efficient as compared to other programming languages having the compilers. The developers have compared what they are developing to be very efficient, effective and faster than the popular C and C++ programming languages.

ii) Unavailable in Majority Mobile Browsers

The programming language works best on personal computers. It is most efficient and available on regular computers, desktops, servers and laptops. However, the program is not efficient in the mobile browsers used by smartphones and tablets. The sad thing about this defect is that most computers are increasing in size and have reached the stage of smartphones and yet the programming language has not. As a result, the programming language is getting to a point where it is going to get unpopular in terms of usage. This is because most organizations are working in the direction of developing mobile phone websites.

Perhaps we hope that the python developers will soon advance the language to be available on mobile browsers.

iii) Limited Design Options

The Python programming language is limited in terms of designing options. We can say that it has restrictions on design options. This program entails actions of typing and more of observation on what you are typing. The errors that have not been seen by the tests of the interpreter will only show up during the last stage of running the lines of codes.

The limited designs in this program include that you can only access the Python interpreter once at a time. This will limit you in terms of testing since you will have to divide the task to different processes and hence cost more resources and time wastage. The only way to save this situation is to apply the feature of indentation. As a result, we will have avoided the emergence of issues and errors that would have occurred in the program.

In summary, Python is one of the best programming languages if you become careful while doing operations in it. As we have seen above, we cannot compare the ratio of advantages to the disadvantages. This shows that there are a lot of advantages in using this programming language to write your own lines of codes. However, we should be keen to avoid meeting the negative aspects of the language by becoming very careful with the language itself.

CHAPTER TWO: STARTING THE PYTHON PROGRAMMING JOURNEY

Introduction

Before we get into deeper aspects of the python programming language, we are going to learn and understand some of the few terminologies that are used in python. Since this book is a beginner's guide, are going to make sure we understand most of the terms so that we can avoid confusion and getting lost in the process of writing lines of codes. I hope you will take notes from the same chapter and try to distinguish each and every term.

Common Terms in Python Programming Language

- **Class** - This is a blueprint or a template or a structure that is usually used in the creation of user-defined objects.
- **Function** - This entails a group or batch of codes that is used in invoking most especially with when a calling program is used. Other uses include calculation or the use of autonomous services.
- **Docstring** - This is a type of string usually appearing as the first expression in a module, defined class or a function. It also used for documentation purposes.
- **Immutable** - This is an example of an object existing inside a code that usually contains a fixed value. Example of such are numbers, tuples and strings. Once the object contains the fixed value then it cannot be changed and hence you have to create a new one with different values. Another perfect example where it is very applicable is like the case of keys existing in a dictionary.
- **IDLE (Integrated Development Environment for Python)** - This can be defined as the basic environment for interpretation and editing of Python programs. As a beginner, it can help and guide you on how to write basic codes. Its main advantage is that it saves both space and time.
- **Mutable** - These are types of objects that do not have fixed have anc hence they have the liberty to change their values within the python program. It is important to note that they maintain their original or initial values. It is simply the opposite of Immutable.
- **Interactive** - This is a feature of the interpreter that allows the beginner to try out new things and test them on how they will work or behave. It is very helpful to the beginner since it gives you the freedom to learn and experience a lot of concepts that you may have

learned before. It is also a technique that boosts the development of new skills and new ideas that may come on board.

- **List** - This is a built-in data type in python that has sorted values of both mutable and immutable type. It can also accommodate the numbers and strings of the immutable type.
- **Python 3000** - As discussed before, you are not able to shift back to older databases of python 2 if you have installed python3. It is evident that most people have remained as users of python 2 because they fear this development. As a result, python 3000 comes as an intermediate option that allows the python 3 users to shift back to older databases of python 2 versions.
- **Object** - In terms of python programming language, we can define a object as a type of data containing a value or a characteristic which is much defined such as that of a method.
- **String** - This is a basic feature of python programming language whose main objective is the storage of texts. For instance, in python 2, the strings are used to store the texts in a way that the string type bears the data that are in binary format only.
- **Triple "quoted" string** - This is a type of string that contains instances like single quotes or instance of double-quotes. They have very many uses in the python programming languages. Some of the uses include simplification of lines of codes by use of the single and double-quotes. As a result, going through the line of codes becomes very easy.
- **Type** - This defined a category of some data objects that occur in several programming languages. They are separated and distinguished from each other by the use of properties and functions such as mutable and immutable types. Examples of these types in python programming language include; string, long, tuple, dictionary types, integer and floating points.
- **Tuple** - This is an in-built data type that includes immutable values of data but in an ordered sequence. In other occasions, the tuple can contain the mutable values such as the dictionaries that are within it.

Beginning the Journey

At this point, we have analyzed and known the background knowledge and history about the python programming language. We have discussed its benefits as well as the negative side of the programming language. We are going to set up the environment that will allow us to interact with the python programming language.

Installation of the Python Platform or Program

If you are using Ubuntu or macOS, then the Python environment is already installed on your computer. This just requires you to search the platform or the program and hence get started on the same.

For the case of Windows computers, we are going to download and install the program. This should not be a worry unto you because it works perfectly well in this environment after it is installed properly. It is important to note that it is very compatible for windows from windows 7 up to windows 10. We are going to follow the following steps in order to install the python into our windows computer;

- The first step is to download Python either version 2 or version 3. Both of the versions are nice choices it just depends on which one will suit you best in terms of accessing and acquiring it. Grab them at Python's official download page at https://www.python.org/downloads/
- After downloading either of the versions to your Windows computer, you click to run the installer. At this point be keen to select the option "Customized Installation".
- A dialog box should appear. On the dialog box, click on every checkbox below the "Optional Features" and then click "Continue".
- On the next page, you are supposed to check for the "Advanced Options" tab in order to choose the appropriate storage location where you are going to install the python 2 or 3.

At this point, you are going to follow another series of steps in order to set up an appropriate PATH variable. In simple terms, you are going to select folders or directories which are required to store and link all the appropriate packages. In order to set up the PATH VARIABLE we are going to follow the steps below;

- Go to the Control Panel of your computer and search for the "Environment".
- After finding the System Environment Variable, click on the Edit option and then the Environment Variables.
- Click on the User variables to either create a new path or use an existing path.
- Open the command prompt of the Windows operating system and type the word "Python". When you find it, it will lead you

to the python interpreter.

Points to note;

i. When you want to create a new path in the USER VARIABLES option, click on PATH and add it to the folders that are existing there. The only caution to take is that the VARIABLE VALUES are usually separated by the use of a SEMICOLON.
ii. If you choose to edit an existing path, you have to be careful so that the values are located in different lines. In order to achieve this you select the NEW option and put the folders or directories in a different line too.

Setting up the Text Editor in Windows and macOS

A text editor is a basic tool that brings to the platform of programming Python language. In other words, you cannot do anything without the presence of the text editor. The common confusion among Windows users is that beginners think that Word is a text editor for programming. From now on, you should understand that the only Windows built-in text editor that is safe for editing source code is the NOTEPAD.

The best version of Notepad that is used on Windows for Python programming is NOTEPAD++. For macOS, TEXTWRANGLER is a great code editor worth trying. In order to set up on Windows we will use the following procedure;

- The first step is to acquire the Notepad++ by downloading the latest version from its official site - https://notepad-plus-plus.org/ .
- After downloading it, go to the settings and select the “Language Menu and Tab settings”.
- Click the checkbox beside the “Expand Tab” in order to activate it. Please be keen to check the value in it to be 4
- Close the tab.

For the case of the macOS, we are going to follow the procedure below;

- The first step is to download TextWrangler from the Mac App Store. This will install the program as well.
- A point to note is that you will not be required to register the software but to install it freely. So feel free to cancel the pop-up

registration box if it appears.

- Follow the simple instructions that come up on the screen and then save and close.

Once the program is fully installed, then you are very ready to start learning the Python programming skills.

Acquiring the IDLE

Before getting far with the Python installation, I would advise you to also download the IDLE (Integrated Development and Learning Environment). This is because they are supposed to be installed together throughout the installation process. In other words, this is the environment that we are going to interact with once we start learning of the Python programming language hence it becomes necessary to be installed together.

The following are features of a standard IDLE that will make your programming with python language very easy to work with;

- Python shell- the function of this shell is to highlight or select the syntax of the programming language
- Integrated-debugger- this feature is unique since it contains strong and persistent breakpoints, stepping and visible call stacks that make things become easy to work with.
- Multi-window editor (Text) - this feature assists the programmer to highlight, indent and finish writing the code.

If you find an environment that has the above features then you can install the package but you have to make sure that it works perfectly with the python programming language. However, in the modern world, the websites that offer the different versions of python whether 2 or 3, they usually attach the environment package unto it so that it can be downloaded and installed together. This shows that it is not easy to get lost in the installation process.

On the other end, if you do not install the IDLE package properly, you will face problems like focusing problems, unable to copy things and even problems with the graphics interface. In order to avoid these problems, be careful in terms of choice of environment package. Please choose a package that is very compatible with not only Python programming language but also your operating system.

In summary, setting up python in order to start coding is pretty easy and available even for a beginner. If you decide to use the macOS or Ubuntu then you have everything on your computer. If you are a Windows user then it is very simple to download it. It is now a good time that we are supposed to start learning of the concepts of how to program using the Python programming language.

Our First Interaction with Python Programming Language

From the previous chapter, Python is an interactive, flexible, dynamic and interpretative programming language. All these features of the Python programming language make it susceptible to be used or manipulated in all sorts of different ways. As a beginner, you may decide to perform tests on codes or explore its features in different ways and still get everything done attributing to its interactive feature.

It is important to note that you can manipulate Python effectively by the use of the following platforms;

1. The Command-line platform
2. The IDLE platform.

We are going to learn of the two platforms in order to understand how we can freely interact with the python programming language.

The Command-line platform

This is the most direct way to interact with the Python programming language. It deals with the user seeing the lines of codes and the results after each and every line of code. This means that once the programmer finishes writing the piece of code and hits the enter button, he or she can see the immediate action of the piece of code. The surface where the line of code is written is referred to as the "prompt".

However, this is not the most preferred way of interacting with the python programming language. This is the simplest way to do some explorations on how the codes of python programming language work.

Interacting with the Command Line Platform

There are numerous ways in which we can access and interact with this platform. The numerous ways and manners depend on the type of operating system that has been installed on your computer. For windows users we are going to start python in the following ways;

I. Click on the menu item from the Start menu.
II. Another alternative is to locate the folders or directories that contain the installed files and then select the "Python Command Line".

For the users of UNIX, Linux or MAC OS operating systems, you will have to search and run the “Terminal Tool” and run the python command in order to start your programming session. It is important to remember that we give the computer commands so that they can be our servants and do what we want them to. Therefore, Python programming language simply works in the same way. You have to type in the commands that they are familiar with so that the computer can perform and action for you. Python simply acts as the platform which converts your thoughts to a language that the computer understands. In order to understand this concept, we are going to have an example below that will illustrate how python works;

i. Step number 1: Open the command line.
ii. Step number 2: Type the following words after the >>>prompt: print (HELLO THERE! I AM LEARNING PYTHON!)
iii. Step number 3: Hit the enter button so that the computer can know that you have finished entering the command.
iv. Step number 4: Observe what the computer does. Once you have hit the enter button, the computer immediately displays the message “HELLO THERE! I AM LEARNING PYTHON!” within a fraction of a second.

The reason why the program has responded correctly is that we have entered the correct format of command in the prompt. If we could have given an incorrect format of command then the following would have occurred;

Input: Print (“HELLO THERE! I AM LEARNING PYTHON!”)

The python program would respond in the following manner

Output: Syntax error: invalid syntax

From the above example, the computer will display a syntax error message because the format cannot be recognized by the command line prompt. This situation also happens when the programmer has typed and run half the command. The main reason for the syntax error above is typing the print command using an uppercase letter P. It is important to note that Python programming language is very case sensitive and can print an error message when one command is inappropriately written.

How to Exit in Python

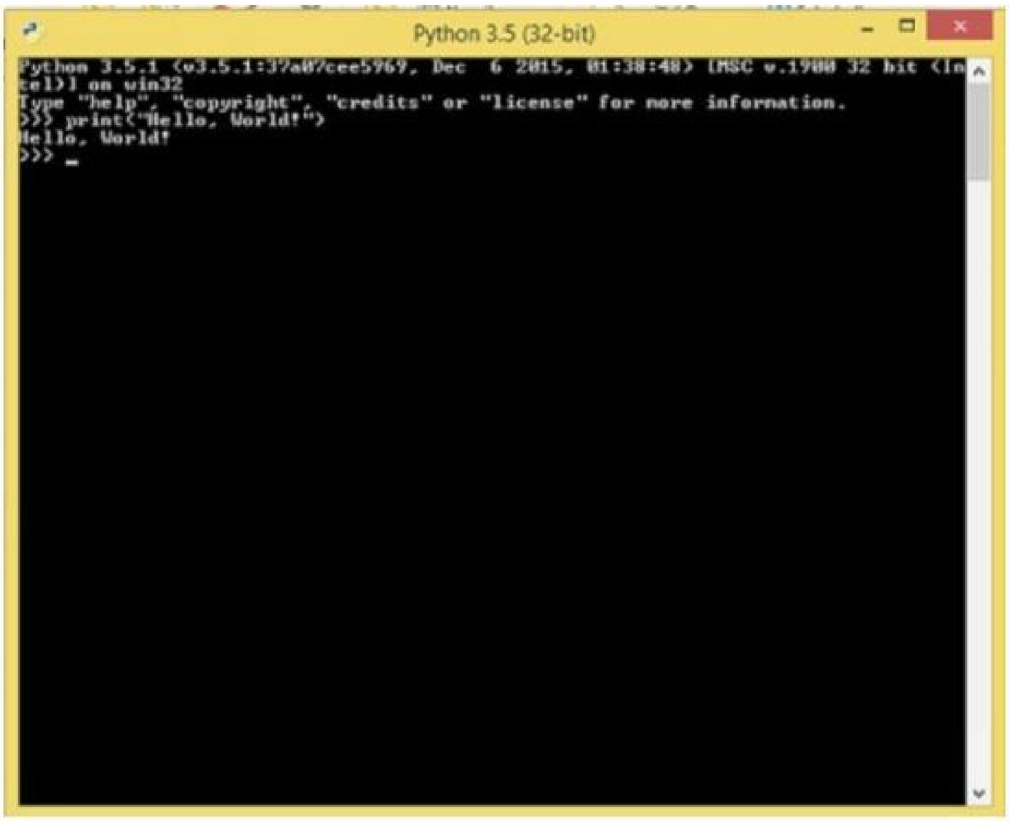
In order for you to quit or exit from python then you can choose to type the following commands;

exit() or

quit() or

Control+Z+Enter

The following is an example of a command-line interface;



```
Python 3.5.1 (v3.5.1:37a07cee5969, Dec  6 2015, 01:38:48) [MSC v.1900 32 bit (In
tel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> print("Hello, World!")
Hello, World!
>>> _
```

The IDLE Platform

As mentioned above, this tool or package comes with the python package but can also be downloaded from other sites that are available on the internet. This is an environment that is developed in order to create an efficient interactive platform that you can write your lines of codes neatly and effectively.

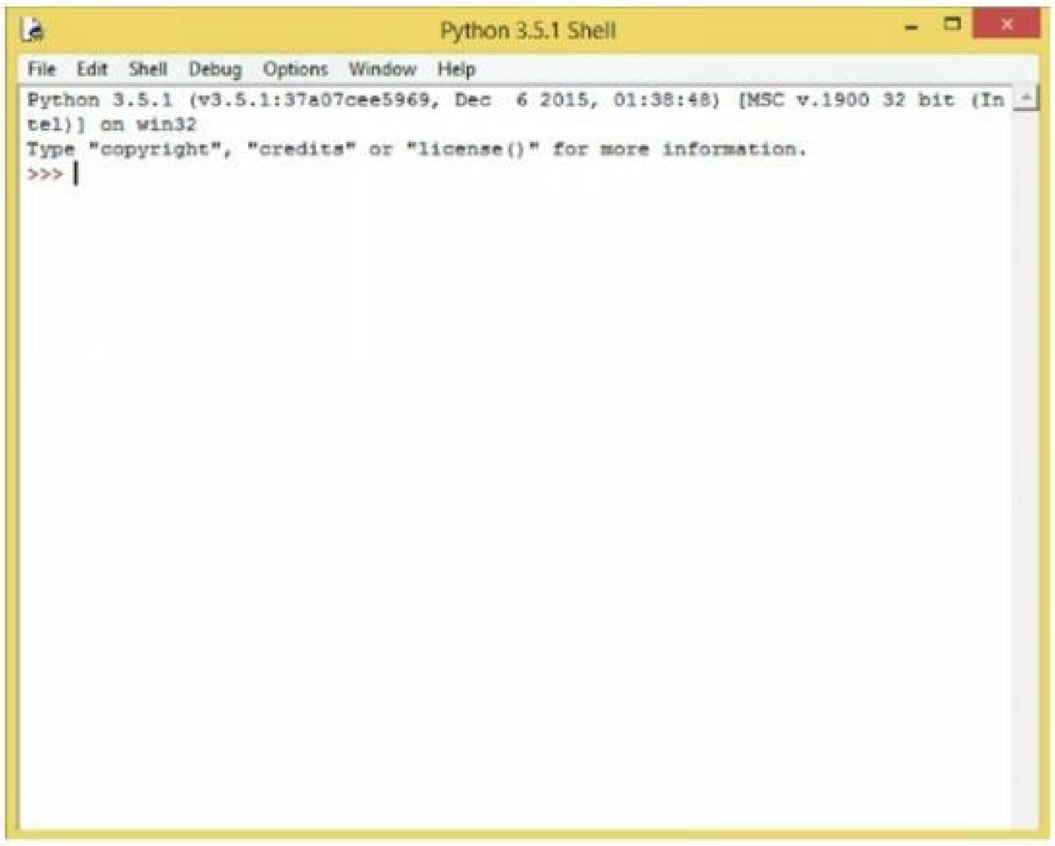
For you to locate the IDLE platform, you may go to the same folders or directories that you located the Command line platform and then click to launch the environment. After clicking on it, it will open the python shell window. Let us discuss the python shell window in detail.

The Python Shell platform

This is a platform that shares properties with the command line platform or window. The only thing that differs with the command line window is that it has drop-down menus. The operations and the >>>prompt are still similar.

The other noticeable differences in the two platform in terms of operations are that in the command line you cannot go back to a line of code that has already been run but in IDLE you can go back and even edit it. The IDLE platform allows you to modify your line of codes and also copy or delete what was not necessary. In a quick summary, the IDLE appears to be an improved version of the command line platform.

The diagram below shows an example of an IDLE platform;



```
Python 3.5.1 (v3.5.1:37a07cee5969, Dec  6 2015, 01:38:48) [MSC v.1900 32 bit (In
tel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> |
```

From the diagram we can see and observe several items belonging to the menu such as FILE, EDIT, SHELL, DEBUG, OPTIONS, WINDOWS and HELP.

The main function of the Shell and Debug menus is to provide an environment for creations and coding of larger programs by the programmer. They simply give a provision for the enlargement of programs. On the other end, the shell menu is for the purpose of resetting or restarting of shells or log of shells. It provides an ample capability so that you can reset the shell or locate the last reset once a problem occurs during coding.

The components of the debug menu include options of tracing and highlighting sources of file exceptions and lines with errors. It also gives you the provision of interacting with a running program. For instance the “Stack viewer” will output a new window that shows the current ongoing python stacking.

The Options menu displays a compatible and suitable working area that is relative to the working environment of the python preferences. On the other end, the Help option leads you to the help website or the python documentation.

The File Menu contains options that will allow you to create, open or save a file. The new option will lead you to a new window that is blank. Initially, a new file is named “Untitled” but you can save it in the way that is suitable according to the program. The difference between the File and Shell windows is that it introduces us to other different options such as RUN and FORMAT menus. The two windows work in coordination since when you run the program while on file window it will be displayed on the Shell window.

The Script Mode

Lastly, in this mode it is not as interactive as we have seen in the other platforms above. In other words, you are not able to see the result and actions of your codes just as in the command line platform. In this case, in order to run your piece of code and see its results you will have to run the entire written script or perform an invoking using a code.

In summary, we have already started the Python programming journey!

CHAPTER THREE: BASIC SYNTAX OF PYTHON PROGRAMMING LANGUAGE

Introduction

This refers to the defined rules and procedures that are required in order to achieve interpretable lines of codes. These rules will enable you to write, edit and interpret any line of codes that are written in the Python programming language. In addition, it will assist you in familiarizing with the Python structure of coding.

Reserved words in Python Programming Language

The reserved keywords can also be referred to as Keywords. These are words that are not supposed to be used as either functions, variables, identifiers or constants. When the reserved words are used wrongly, they result in errors during the execution of the line of codes. The following are some of the main keywords that are majorly used in Python;

And, assert, break, class, continue, def, del, elif, else, except, exec, finally, for, from, global, if, import, in, is, lambda, not, or, pass, print, raise, return, try, while, with, yield.

Identifiers in Python Programming Language

They can be defined as the names or the initials that are assigned to distinguish the variables, functions, classes, modules and objects. They are necessary for this programming language so that you can easily differentiate entities that have been used. It is important to note that you should name each entity appropriately so that you can identify them easily in the future.

The following are some of the guidelines that you should observe in order to name an entity in an appropriate way;

- An identifier can be composed of lowercase letters, underscores, digits and uppercase letters. For example you can create an identifier like myFunction, my_class and many others.
- You are not allowed to include the special characters when creating an identifier. Some of the special characters that are not allowed include; @, $, %, and many others.
- You are not allowed to name an entity with an identifier that starts with a number or any digit for example 2Class. On the other end, you are allowed to use the numbers or digits at other places but not the beginning.
- You are required to know that python programming language is very

case sensitive. For instance, the words Lazy and lazy are not the same in this programming language. You should be keen on choosing an identifier for an entity.

- Another caution that should be understood is that you cannot use a reserved word as an identifier. It is not allowed at all.
- The names that are assigned to distinguish Classes should always begin with a capital or uppercase letter. However, all other entities should be named with identifiers starting with lowercase letters.
- The Underscores symbols are preferred when you want to differentiate between multiple words.

In summary, it is very important to note that appropriate use of identifier help in making references in the future. You should select names for entities that can make you remember easily.

Use of Quotation marks in Python Programming language

Depending on the type of quotation marks, Python allows you to use them in order to make an indication of string literals. The only caution that is to be undertaken is that once you use a single or double quotation mark then you must end with the type that you started with. For instance, a situation that requires you to use triple quotes is when your strings exist in many lines.

Use of Statements in Python Programming language

These are set of instructions that the interpreter can run or execute. There are different types of statements that exist in this programming language. For instance, if a value is assigned to any variable like the_variable = “cat”, then this becomes an “assignment statement”. It is also allowed that you can write the assignment statement in a short format such as v=16. Other examples of statements in this programming language include; for statement, if statement and while statement.

The Multi-line type of statements

Just as the triple-quotation string, a statement can spill over a number of lines. This is what is referred to as the multi-line type of statement. In order for you to break such long statements to fit the number of lines required you can use the brackets, parentheses and the braces to wrap the lines. In addition, you can also use the element of the backslash (\) at the end of every line so that you can indicate that the line has not ended. The concept of the backslash is majorly for the purpose of indicating that the line continues in multiple lines.

How indentation is used in Python Programming language

In other programming languages, their structures are defined by the use of braces and brackets in writing of codes. However, python uses indentation as a way of defining its structure in writing the lines of codes. It is important to note that this is not a style of writing to make it impressive but a requirement while writing lines of codes in this programming language. The main objective of indentation is to make the block of codes in Python more understandable and readable.

The following example illustrates the use of indentation. It is an example of lodging room rental in a small town;

```
def room_rental_cost(hours):
    cost = 360 * hours
   if hours >= 10:
          cost -= 700
      elif hours >= 5:
          cost -= 400
      return cost
```

The most important thing in the application of indentation in the Python programming language is maintaining consistent indentation spacing in your lines of codes. On the other end, some of the IDLEs that assist you in typing of codes will automatically help you in maintaining the indentation space. The standard indentation space is always four to the right.

Use of Comments in Python

This is a common feature among all the programming languages. The comments can be defined as the explanations of code that are placed in between the lines of codes in order to remind you or inform another programmer of what the written code does. It is very important to write comments so that it can be used as a source of reference to other programmers.

In Python, to write any comment you must start with a hash (#) sign or symbol. The hash symbol is mainly to inform the interpreter that this is not part of the codes to be executed in the program. On the other end, the comments can spill over a number of lines and hence the programmer is required to assign a hash sign at the beginning of each and every comment line.

The following are examples of how we are supposed to write the comments;

```
#the game is over

#shows the print action

#John Parks-12/2/2020
```

In addition, it is also important to note that comments can be written in any other understandable language. In some of the IDLEs, the comments are usually indicated in red so that you can be able to identify them by the use of your eyes. I would like to encourage you that you begin your line of codes with short and simple comments that will inform other programmers of the title, name and date when the program was written. This can help when another programmer wants to modify your program in the future.

Use of Blank lines

Blank lines are used in programming to make the codes easier to read and locate them. They are also used to separate different sections of codes that play different roles in a program. The computer always ignores the blank lines because they are not part of the program to be executed. They are majorly used for neatness and tidiness of the space of programming in the IDLEs.

Use of the "waiting line"

This is always the last line of your program that will display the command "Press the enter key to exit". When you press the Enter button, the program will shut down automatically. I would advise you to keep this window open until you are sure that you are done with your coding. The code is illustrated below;

```
raw_input("\n\nPress the enter key to exit.")
```

Summary

In this chapter we have interacted with some of the features that will assist us in coding in the Python programming language. I believe that you have installed the python program as well as a suitable IDLE that will help you in a place to write the codes. We have also tested on writing a simple program in the interactive mode of the computer. In addition, we have seen the script mode and some of the menus that exist in that mode. I believe we have laid a basic foundation for this programming language and hence we are prepared to learn other parts of the programming language too.

Practical Exercises

1. Generate a syntax error in the interactive mode by inputting your favorite television program show. Correct the error by

entering a statement that prints the name of your favorite television show.

2. Create a simple program that prints your name and current city that you live in.
3. Create a program that writes a famous quote and its author in two different lines. (Hint: feel free to manipulate multiple print statements).

CHAPTER FOUR: VARIABLES, AND DATA TYPES AND DATA STRUCTURES

Introduction

We have been introduced to the Python programming language and now we are able to write and save a simple program. It is time to dig into concepts that will build our knowledge in the Python programming language and generate more complex codes for other programs.

In this chapter, we are going to study about different ways in which you can manipulate data, categorize and store them at the same time. We are also going to study in detail on how to use the different categories of data in our python programming language. On the other end, we will focus on creating an interactive program that will prompt the user to enter some of the data in order for the program to run. The main objective of this chapter is to help you to do the following task;

- Create programs that can be able to perform some arithmetic.
- Use the different variables in the manipulation of data.
- Store different categories of data in computer memory.
- Create very interactive programs that prompt the user for some information in order to be executed by the program.
- Use the different features like the quotations in the manipulation of strings and texts.

The Variables in Python

In a simple definition, a variable can be likened to a container or a room that acts as a store of data values. The values that are stored in the variables can be accessed and manipulated at any time. In other words, it can be referred to as pointed memory that can instruct the computer to retrieve or save data to. Variables can also be used to organize information in computer memory.

It is important to note that there is a big difference in how python utilizes the variables as compared to other programming languages. In the other programming languages, the variables are very specific to the storage of the data type. For instance, a variable can be meant to story only integer type of data values. On the other end, Python is very flexible in terms of dealing with the variables. For instance, if a variable is required in Python, the only thing that is required is to assign a value to it. You are flexible to change the data type and assigned value before executing the python program. Let us study the example below;

Example 1

```
Our_variable=20
```

Understanding our code

It is very important to note that when declaring a variable, it does not mean that “our_variable” is equivalent to 20 but it means that the variable is set to 20. You should be careful not to mix the two to prevent yourself from getting the wrong interpretation of your code. In order for you to add to the variable you can do the following step;

```
>>>our_variable = our_variable + 10
```

When you want to know how Python reacted to your line of code then you can print as shown below;

```
>>>print (our_variable)
```

The result will be:

```
30
```

In this example, the two values have been combined to a new value by virtue of simple addition. The results have been printed after the addition.

Example 2

We are going to manipulate the “our_variable” to store a string(literal) as shown below;

```
>>>our_variable = “Banana”
```

Let us print the string;

```
>>>print (our_variable)
```

The result will be:

```
Banana
```

Practical Assignment

A. Identify the different parts of the lines of codes below;

```
#The greeting program
 #shows the uses of a variable
 #John Parks 2/02/2020
 name= “John”
 print name
 print “Hello, “+ name
 raw_input (“\n\nPress the enter key to exit.”)
```

How should we name the variables?

There are a few rules in regard to naming variables in Python. This will enable you to create a required name of a variable because of failure to which it will create an error message. The following rules are important in assigning a variable name;

i. The name should contain ONLY letters, underscores and numbers.
ii. The variable name is NOT supposed to begin or start with any number.

Besides the rules, the following are some ideas that have led other programmers in creating very good variable names.

i. Be careful to choose on very descriptive names that give other programmers the insight of what is taking place at any point of the program. This can be very important for future reference and modification of your program.
ii. Be very consistent and simple in naming your variables to avoid confusion and ambiguous names. Do not use the names that will confuse or bring different ideas in the program.
iii. Be careful so that while naming you can follow the rules of python programming language. Do not come up with names that can resemble or copy the keywords of the programming language. In other words, you are also supposed to remember that python is very case sensitive. Other rules like what to avoid at the start of a name should also not be violated.
iv. Lastly, you should check the length of the name because very lengthy names create a problem usually in this programming language. Hence you should choose a simple, descriptive name that is not too long to be a variable name. The required length of a variable name should be below fifteen characters.

In summary, the trick of choosing a good variable name is the combination of the guidelines that have been shown above. You should be able to choose a name that can guide another programmer in your lines of codes apart from the comments in the same program. This can be a good start to programming in python.

The Data types in Python

When it comes to handling data types, python deals with data types so as to help programmers exploit the data in an efficient manner. Some of the data types that we are going to study include; numbers, lists, Booleans, strings, time, date etc.

Strings

They can be defined as the Unicode type of characters in the Python programming language. The strings can contain special symbols, letters and combination of numbers. Strings are defined by the use of quotes learned in the previous chapter. For example;

```
>>>string one = “I am defined by single quotes.”
>>>string two = “I am defined by double quotes.”
```

In special cases where they exist literal strings defined by single quotes, a backlash is supposed to be added in order to escape that specific character as shown below;

```
>>> string3 = ‘It does not look impressive at all.’

>>> print (string3)
```

The results will be: It does not look impressive at all

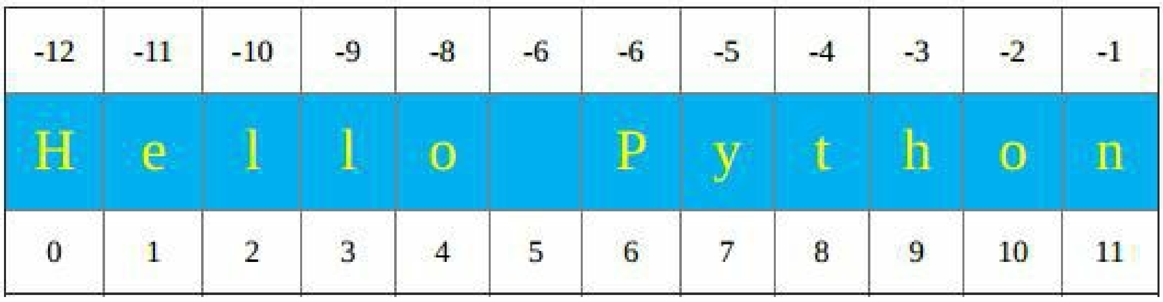
You will obtain the same results if you choose to use double-quotes.

On the other end, in the Python programming language, the strings can either be subscripted or indexed so that its first character is not one but zero.

Let us observe the example below on how we can get to index in Python.

```
>>>s = “Hello Python”
```

The python will index the string above in the following manner;



-12	-11	-10	-9	-8	-6	-6	-5	-4	-3	-2	-1
H	e	l	l	o		P	y	t	h	o	n
0	1	2	3	4	5	6	7	8	9	10	11

In order for you to access any of the letters that you have typed in the string you would use the criteria of entering the variable name and the index enclosed in square brackets as shown below;

```
>>>s [6]
```

The output will be ‘P’

The simple formula of obtaining other character is

```
>>>s [len(s)-1]
```

In summary, besides indexing of strings you can perform various arithmetic operations and functions on the strings.

Concatenated strings

We have studied this concept in variables but we are going to illustrate how we can join strings to each other in python programming language. For instance, let us study the simple strings below;

```
>>> “Hi” + “Python programmer”
The result will become: ‘HiPython programmer’
```

From the example above, we can recognize that the strings are joined by the operator +. It simply joins then like the simple addition mathematics. It is important to note that when the strings are joined, they are joined exactly as they were. There is no inclusion of spaces just as shown in the example above.

Repeated strings

We can repeat the number of strings to the number of times we want by the use of * symbol or operator. For instance let us repeat a string five times as shown below;

```
>>>s = “**^**”
>>>s * 5
‘**^****^****^****^****^**’
```

Another example;

```
Print “Cookie” *10
```

The output will be “Cookie Cookie Cookie Cookie Cookie Cookie Cookie Cookie Cookie Cookie”

Sizing of Strings

The size of the string can be obtained by the use of the “len() function”. For example if we want to obtain the size of the string “Parker” then we would run the following function;

```
>>>len("Parker")
The output will be 6
```

Sliced Strings

In this section, we can split a string into substrings by the use of slicing action or notation. This is achieved by placing the index of the strings and splitting them using semicolons. The following is an example of sliced strings;

```
>>>"Python"[3:5]
will result in:
'ho'
>>>"Python"[3:6]
will yield:
'hon'
```

Other simpler ways to achieve slicing of strings into substrings include;

i. Use of variable

In this way we can store the string in a variable and go direct in manipulating the variable in order to create the substring as shown below;

```
>>>p = "Python"
>>>p [3:6]
the output will be: 'hon'
```

ii. Omission

This is done when we want specific parts of the substring and hence we omit the second part of the index in order to obtain the targeted substring as shown below;

```
>>>p = "Pythons"
>>>p [4:]
the output will be: 'ons'
```

iii. Omission from the first character

This is a special case where you only want the first character from the original string. In order to achieve this then you are going to omit the first character of the index and indicate the last character that you want to appear in the string as shown below;

```
>>>p = "Pythons"
>>>p [:4]
the output will be: 'Pyth'
```

String Functions

In this section, there are two types of functions which are the upper and the lower functions. This function works hand in hand in the conversion of uppercase and lowercase strings respectively. For example, if we want to convert the string “John Parker” into lowercase, we will do the following;

```
>>>c = “John Parker”
>>>print (c.lower())
the output will be: john parker
```

If we want to convert it to uppercase, we will apply the upper function as shown below;

```
>>>print (c.upper())
JOHN PARKER
```

On the other hand, there is the str () function. This is a special function that creates a string from “non-string” characters. This function makes the programmer perform actions such as printing characters that are not strings as though they are strings.

Let us keenly look at the example below;

```
>>>pi =3.14
>>>str(pi)
‘3.142’
>>>print(“This is a very good formula: ” + str(pi))
This is a very good formula: 3.142
```

In summary, you should do more examples on strings so as you can grasp the different concepts existing under this section.

Numbers

The major advantage that you can obtain while working with Python programming language is that you do not need to declare the numeric data type in order to start manipulating the values. It has the ability to distinguish the data type before executing the line of codes. Some of the numeric data types that are supported in the latest version of Python (Python 3) include complex numbers, integers, and floating-point numbers.

The Integer numeric data type

These are whole numbers that do not contain any decimal points amongst them. They can either be positive or negative as long as they still do not have any decimal points. The best thing about this integers is that they have an unlimited size in this version of Python. The following are types of integers and literals that are accepted and recognized in the Python programming language;

Integers: 234, -456, 10001

Octal literals: These are numbers to the base of 8. They are indicated by starting with number zero then letter O in upper case or lower case. For example:

```
>>>a = 0O9
>>>print(a)
the resultant output will be: 9
```

Hexadecimal Literals: These are numbers to the base of 16. They are indicated by starting with zero then letter X either in upper case or lower case. For example;

```
>>>hex_lit = 0xA0C
>>>print(hex_lit)
the resultant output will be: 2572
```

Binary literals: These are numbers to the base of 2. They are usually indicated by starting with a zero then a letter be either in upper case or lower case as shown in the example below;

```
>>> c = 0b1100
>>> print(c)
the resultant output will be: 12
```

How to convert from integers to string data type

It is important to note that this is a feature that is only allowed in the python programming language. It allows you to convert the integer type to literal strings during the printing process. The functions that are used in accomplishing this process include; Oct (), bin () and hex ().

The following are examples of how to convert to different string literals.

Examples

Convert integer 8 to its octal literal equivalent.

```
>>>oct(8)
the output will be: ‘0o8’
```

Convert the integer 2572 into a hexadecimal literal string equivalent.

```
>>>hex(2572)
the resultant output will be:‘0xa0c’
```

Convert the integer 12 into its binary literal string equivalent.

```
>>>bin(12)
```

```
‘0b1100’
```

Note: In order for you to check in the variable about the type of value that has been stored there then you can type the following command;

```
>>>type(x)
The output will be <class ‘str’>
```

The Floating-point integers (numbers)

They are commonly referred to as floats. They represent the real numbers. This means that they also contain the decimal points that define the fractional part of integer numbers. In addition, you can also write these float-type integers in scientific notation which is usually to the power of ten. (10^x) the following are examples of floats;

```
>>>5.2e3
the output will be: 5200.0
>>>5.2e2
the resultant output will be: 520
```

The complex integers (Numbers)

These are the integers which represent the real and imaginary part of numbers. They are usually written in the format of “a+bJ”. The first part “a” represents the real part of the complex integer whereas the “b” represents the imaginary part. However, the complex numbers are not widely used in Python programming language. But you can still see in the example below what complex numbers look like in Python;

```
>>>a = 3 + 5j
>>>b = 5 – 2j
>>>c = a + b
>>>print(c)
(8 + 3j)
```

Date/Time

It is very evident that almost all the applications and programs require time in order to work effectively and efficiently. In this programming language, you obtain the time and the date by the use of functions. A good example of a function that brings the current date and time is “datetime.now()”. This function basically calls the built-in functions that exist in python in order to obtain the current date and time.

The following is an example of the code in Python;

```
>>> from datetime import datetime
>>> datetime.now()
datetime.datetime(2019, 4, 9, 1, 15, 19, 851318)
```

From the example above, the date is in the order in which you can only identify the year. In order for you to obtain the readable format that is easy to recognize you will have to refer to the python library using the "strftime".

The following is an example of how the function is used from the standard library in order to obtain the correct time;

```
>>>from time import strftime
>>> strftime("%Y-%m-%d %H:%M:%S")
the resultant output will be: '2019-02-09 01:19:02'
```

The Boolean Data type

These are the data types that are mostly used in comparisons that will end up with only two sides which are true or false and yes or no among other options. Let us study the example below and see how the Boolean data type is used in Python programming language;

```
bool_one = 5 == 1*3
bool_two = 9 < 3 * 2**3
bool_three = 7 > 3* 4 + 8
print(bool_one)
print(bool_two)
print(bool_three)
```

The resultant outputs will be:

```
False
True
False
```

The List

In comparison with a variable, a list can be likened to a bigger container that can store a number of variables as well as other data types. It is mainly used for storage. In order to assign and define components to a list, we will observe the following;

```
Our_list = [Item_one, Item_two, Item_three]
```

In python programming language you can also create a blank list as shown below;

```
Our_list =[]
```

Since the list is always indexed, the first value in the list always has the zero indexes. In order to print the first value in the list below then you will enter the following code;

```
Cars= [ “Mercedes”, “Toyota”, “Bugatti”, “Subaru”]

>>> print(cars[0])
the resultant output will be: Mercedes
```

In order for you to count the number of cars on the list, you will have to use the lens function as shown below;

```
>>> len(cars)
the resultant output will be: 4
```

In order to remove a car from the list then you will execute the following command;

```
>>> cars.remove(“toyota”)

The resultant output will become; Mercedes, Bugatti, Subaru
```

In order to add a car into the list, you will use the append function as shown below;

```
>>> cars.append(“BMW”)

The resultant output will be: Mercedes Bugatti Subaru BMW
```

Sliced lists

The list can be sliced the same way the strings are also sliced. This happens the same way we did in the strings when we need to select a few items from the list. The example below is an illustration;

```
>>> Cars [3:4]
the resultant output will be: [‘BMW’]
```

The dictionary

The dictionary and the list are similar data types. The only difference between the two data types is the mode of access to values that are stored inside them. For the list, we have seen in the section above that you access and manipulate the data using indices. In the dictionary, you access the values by the use of a unique key. The unique key can either be a number, a tuple or a string.

We have to ensure that the key value is an immutable data type. The dictionary structure can be defined by writing the key and separating it from the other values by the use of a colon as shown below;

d = {key_1 : b, key_2 : 3, key_3 : bc}

In python programming language you are allowed to create an empty dictionary as shown in the format below;

d = {}

One major application of the dictionaries is the storage and manipulation of menus, phonebooks and other directories. This platform allows the programmer to add, delete or modify any entries in the storage component.

In summary, the same functions that were used in updating the lists are the ones that are used in updating the dictionaries such as the lens functions and others. In order to remove, use del function.

DATA STRUCTURES

In this section we are going to consider things that we have learned before in the data types but now we will be integrating them with the Python programming language. Do not see it as a repetition but an advancement of the topics.

The List

Some of the methods that exist as list objects include;

- List. Append (x) - This are used to add new items to the end of a list
- List.extend () - They are used in the extension of the list by appending the items that are iterable.
- List.insert() - They are used in the insertion of any item at any position of the list.
- List.remove() - They are used in the removal of items from the top of the list whose value is equal to x. If the value is missing then an error occurs.
- List.pop() - They are used for two functions; removing and returning an item at a point of the list.
- List.clear() - They are used in the removal of all the items in the list.
- List.count() - They are used to count and return the number of appearances of x in the selected list.
- List.sort() - They are used in sorting the items that exist in the list.
- List.reverse() - They are used in reversal of elements existing in the list.
- List.copy() - They are used in returning a copy of the list.

I believe the above information is going to help you in working with the list. As a result, the example below tries to show us how the above functions are used;

```
>>> fruits = ['orange', 'apple', 'pear', 'banana', 'kiwi', 'apple', 'banana']
>>> fruits.count('apple')
2
>>> fruits.count('tangerine')
0
>>> fruits.index('banana')
3
>>> fruits.index('banana', 4) # Find next banana starting a position 4
6
>>> fruits.reverse()
>>> fruits
['banana', 'apple', 'kiwi', 'banana', 'pear', 'apple', 'orange']
>>> fruits.append('grape')
>>> fruits
['banana', 'apple', 'kiwi', 'banana', 'pear', 'apple', 'orange', 'grape']
>>> fruits.sort()
>>> fruits
['apple', 'apple', 'banana', 'banana', 'grape', 'kiwi', 'orange', 'pear']
>>> fruits.pop()
'pear'
```

In this programming language we are going to focus more on the list as compared to other data structures. We are going to see how the stacks can be used as lists in python.

How can we use the list as the stacks?

From the methods that we have studied above, it is very simple to apply the lists as the stacks. The stacks work by accessing the last added element first. It uses the LIFO technique of access to data (Last In Last Out). The following is an example of the same;

```
>>> stack = [3, 4, 5]
>>> stack.append(6)
>>> stack.append(7)
>>> stack
[3, 4, 5, 6, 7]
>>> stack.pop()
7
>>> stack
```

```
[3, 4, 5, 6]
>>> stack.pop()
6
>>> stack.pop()
5
>>> stack
[3, 4]
```

How can we use the lists as the queues?

In Python, the queues can be used as lists too. What determines this conversion is the mode in which the data is accessed. In terms of queues, it uses the FIFO technique of access to data where the first element to be accessed was the one that was first stored.

However, the lists are not suitable to convert to queues. This is because it becomes a slow process since the elements have to be shifted one after the other. The following example shows how we can implement the queues as the lists;

```
>>> from collections import deque
>>> queue = deque(["Eric", "John", "Michael"])
>>> queue.append("Terry") # Terry arrives
>>> queue.append("Graham") # Graham arrives
>>> queue.popleft() # The first to arrive now leaves
'Eric'
>>> queue.popleft() # The second to arrive now leaves
'John'
>>> queue # Remaining queue in order of arrival
deque(['Michael', 'Terry', 'Graham'])
```

Assignment

1. The Comprehending Lists

These are concise ways in which the lists can be created altogether. The following is an example of such a list and hence you are required to analyze the piece of codes and generate the other list;

```
>>> vec = [-4, -2, 0, 2, 4]
>>> # create a new list with the values doubled
>>> [x*2 for x in vec]
[-8, -4, 0, 4, 8]
>>> # filter the list to exclude negative numbers
>>> [x for x in vec if x >= 0]
```

```
[0, 2, 4]
>>> # apply a function to all the elements
>>> [abs(x) for x in vec]
[4, 2, 0, 2, 4]
>>> # call a method on each element
>>> freshfruit = [' banana', ' loganberry ', 'passion fruit ']
>>> [weapon.strip() for weapon in freshfruit]
['banana', 'loganberry', 'passion fruit']
>>> # create a list of 2-tuples like (number, square)
>>> [(x, x**2) for x in range(6)]
[(0, 0), (1, 1), (2, 4), (3, 9), (4, 16), (5, 25)]
>>> # the tuple must be parenthesized, otherwise an error is raised
>>> [x, x**2 for x in range(6)]
File "<stdin>", line 1, in <module>
[x, x**2 for x in range(6)]
^
SyntaxError: invalid syntax
>>> # flatten a list using a listcomp with two 'for'
>>> vec = [[1,2,3], [4,5,6], [7,8,9]]
>>> [num for elem in vec for num in elem]
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

2. Study about the tuples and sequences in the lists and analyze the simple piece of code shown below; Hint (the tuples below consist of values separated by use of commas

```
>>> t = 12345, 54321, 'hello!'
>>> t[0]
12345
>>> t
(12345, 54321, 'hello!')
>>> # Tuples may be nested:
... u = t, (1, 2, 3, 4, 5)
>>> u
((12345, 54321, 'hello!'), (1, 2, 3, 4, 5))
>>> # Tuples are immutable:
... t[0] = 88888
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support the item assignment
>>> # but they can contain mutable objects:
```

```
... v = ([1, 2, 3], [3, 2, 1])
>>> v
([1, 2, 3], [3, 2, 1])
```

Sets in Python programming language

Sets can be defined as a group of items that has no particular order as well as no duplicate items. In the programming world, the sets are used to test for the presence of data items and also the elimination of repeated items. They can also help in mathematical calculations such as differences, union, symmetric differences and intersections.

Sets are created by the use of set () functions. The caution to be taken here is that in the creation of an empty set you should not use the curly brackets but use the round brackets. The following is an illustration of how sets are created;

```
>>> basket = {'apple', 'orange', 'apple', 'pear', 'orange', 'banana'}
>>> print(basket) # show that duplicates have been removed
{'orange', 'banana', 'pear', 'apple'}
>>> 'orange' in basket # fast membership testing
True
>>> 'crabgrass' in basket
False
>>> # Demonstrate set operations on unique letters from two words
...
>>> a = set('abracadabra')
>>> b = set('alacazam')
>>> a # unique letters in a
{'a', 'r', 'b', 'c', 'd'}
>>> a - b # letters in a but not in b
{'r', 'd', 'b'}
>>> a | b # letters in a or b or both
{'a', 'c', 'r', 'd', 'b', 'm', 'z', 'l'}
>>> a & b # letters in both a and b
{'a', 'c'}
>>> a ^ b # letters in a or b but not both
{'r', 'd', 'b', 'm', 'z', 'l'}
```

In summary, the python programming language supports the list and set comprehensions as shown below;

```
>>> a = {x for x in 'abracadabra' if x not in 'abc'}
>>> a
```

```
{'r', 'd'}
```

The Dictionaries

This is one of the useful data types that exist in the Python programming language. In other programming languages, the dictionaries are used as associative arrays and memories. They can be viewed as unorganized or unordered group of unique keys.

Unlike sets, empty dictionaries are created with curly braces instead of round brackets. The data values inside the dictionaries are also separated by the use of the commas.

The best way to describe the work of the dictionary is to see it as a container that stores some data values using unique keys and accessing the values using the same keys. In cases where you want to delete the key, you will be required to apply the del-clause. When you store a value with the same key, the old data values will be updated and hence replaced. In addition, an error occurs if you want to retrieve a data value using a key that is not existing.

The following is an example of how the dictionary is used;

```
>>> tel = {'jack': 4098, 'sape': 4139}
>>> tel['guido'] = 4127
>>> tel
{'sape': 4139, 'guido': 4127, 'jack': 4098}
>>> tel['jack']
4098
>>> del tel['sape']
>>> tel['irv'] = 4127
>>> tel
{'guido': 4127, 'irv': 4127, 'jack': 4098}
>>> list(tel.keys())
['irv', 'guido', 'jack']
>>> sorted(tel.keys())
['guido', 'irv', 'jack']
>>> 'guido' in tel
True
>>> 'jack' not in tel
False
```

From the above example, the dict () function constructor creates the dictionary directly from direct sequences of value pairs as shown below;

```
>>> dict([('sape', 4139), ('guido', 4127), ('jack', 4098)])
{'sape': 4139, 'jack': 4098, 'guido': 4127}
```

In summary, just as in sets and lists, the python programming languages supports the dictionary comprehension as shown below;

```
>>> {x: x**2 for x in (2, 4, 6)}
{2: 4, 4: 16, 6: 36}
```

BASIC OPERATORS IN PYTHON PROGRAMMING LANGUAGE

These are types of functions that enable the programmer to modify and manipulate data efficiently and effectively. The following are types of basic operators that are supported by python programming language;

1. Assignment operators
2. Comparison operators
3. Arithmetic operators
4. Membership operators
5. Logical operators
6. Bitwise operators
7. Identity operators

The Arithmetic operators

These are types of operators that enable the programmer to process and perform mathematical calculations such as computing expressions and calculations of discounts. The table below shows the different types of arithmetic operators, its functions and examples.

Symbol	Name	Function	Example
+	Addition	It adds value to either the left or right	>>>1 + 3 4
-	Subtraction	It subtracts value either to the right or left	>>>10 – 4 6
*	Multiplication	It multiplies values either to the right or left	>>>4 * 2 8
/	Division	It divides the value either to the right or left	>>>10 / 2 5.0
**	Exponent	It usually performs the	>>>2**3

		exponent operations or calculations	2 raised to the power of 3 8
%	Modulus	Returns the remainder after a division calculation	>>>17 % 5 2
//	Floor Division	It performs division where the remainder of the quotient is truncated its decimal numbers.	>>>17 // 5 3

The Assignment operators

These operators enable the programmer to assign values to variables in a more effective and efficient manner.

Operator	Function	Example

=	It assigns a value from located on the right side to the operand in the left side.	x = c
		x = b + c
		x = 8
		x = 8 + 6
		s = “I adore Python.”

+= add and	It performs addition of left and right data and then assigns the output to the left operand	x = x + a
		x+=a

-= subtract and	It subtracts the value of the right operand from the left operand and assigns it to the left operand.	x = x - a
		x-=a

*= multiply and	It multiplies both the left and the right and assigned the result to the left data	x = x * a x*=a
/= divide and	Divides the right operand with left operand and assigns the value to the left	x = x / a x/=a

	operand	
**= exponent	Performs the exponential operations on the left operand	x = x % x%=a
//= floor division and	Performs the floor division calculation on the left operand.	x = x // x//=a

The Comparison operators

These operators work like the Boolean data types where they compare two values and return a true or a false answer. They are also referred to as the relational operators. They majorly compare values on the left and right and return an output of either true or false.

Operator	Meaning	Example
==	Means it is equal to	>>> 8 == 6+2 True
<	Means it is less than	>>>2<5 True
>	Means it is greater than	>>> -1 > 0 False
<=	Means it is less than or equal to	>>>-2<=-5 False
>=	Means it is greater than or equal to	>>> 7 >= 5 True
!=	Means it is not equal to	>>> 6 != 6 False

The Logical Operators

The python programming language only supports three logical operators as shown in the table below;

Operator	Function	Example

or	It evaluates the first argument and if it is false it evaluates the other argument	>>> (2==2) or (9<20)
		True
		>>> (3!=3) or (9>20)
		False

And	It evaluates the first argument and if	>>> (8>9) and (2<9)

	it becomes true it moves to the second argument but if the first argument is false then it is evaluated.	False >>> (2>1) and (2>9) False

not	If the first argument is false, then it returns a true value and vice versa	>>> not (8 > 2)
		False
		>>> not (2 > 10)
		True

In summary, the following is a standard table of precedence that is used in the python programming language. Study it carefully and master it.

No	Description	Operators
1	Exponentiation	**
2	Multiplication, division, modulo, and floor division	*, /, %, //
3	Right and left bitwise shift	>>, <<
4	Regular `OR' and Bitwise exclusive 'OR'	|, ^
5	Equality operators	== !=
6	Identity operators	is, is not
7	Membership operators	in, not in
8	Logical operators	or, and, not
9	Assignment operators	=, +=, -=, *-, /=, %= //= **=
10	Comparison operators	<= < > >=
11	Bitwise 'AND'	&
12	addition and subtraction	+ -
13	Complement, unary plus, and minus	~, +, -

CHAPTER FIVE: CONTROL AND USER DEFINED FUNCTIONS

Introduction

From the previous chapter, we have gone through a series of instructions and if we could have combined all the codes together then our IDLE would have been full. In this way, we can split our programs into smaller problems that are very easy to work with.

In this chapter, we are going to learn how to break down large chunks of programs into smaller bits by the use of functions. We are going to create the functions and hence solve bigger problems through smaller problems. The following are the main objectives of this chapter;

a) Develop our own functions.
b) Insert values to our functions by the use of parameters.
c) Obtain information from our functions by the use of return values
d) Manipulate the global forms of variables as well as the constants
e) Develop simple programs.

In other words, it is important to note that functions provide structure, reliability and efficiency to the Python programming language. Before we create our functions we are going to study about the in-built functions that exist in the Python programming language.

Input () Function

There are so many ways that you can input content into the program and some of them include; from computer memory, mouse-clicking, typing on the keyboard, database and even downloading from the internet. The most used type of input during python programming is the input from the typing of the keyboard. As a result, python developed a user input function for the same that is usually referred to as the “prompt string”.

This function creates an interactive platform because when the input function has been called it requires the user to key into the program. The prompt string provides the platform on the display screen and returns the value as a string to the computer. The following is an example of a program that illustrates the concept above;

```
name = input(“Can you please tell me your first name? “)
print(“It is nice to know your ” + name + “!”)
age = input(“Will you mind sharing your age, please? “)
print(“I am glad to know that you are ” + age + ” years old, ” + name
+ “!”)
```

Understanding our code

When you closely look at our code above, you will notice that there is some space in the second line immediately after the word you. The main purpose of the space is to ensure that the sentence is rightly spaced when it is printed. The same applies even in the fourth line of the code. When we save and run our code it will print the following;

```
Can you please tell me your name?
```

The program won't proceed until you key in your name. After typing in your name, the program will proceed and respond to you that;

```
It is nice to know you John.

Will you mind sharing your age?
```

The program will proceed to the next input function as it waits for you to input your age. After you have inserted your age the program will proceed and print out;

```
I am glad to know that you are 23 years, John.
```

In summary, the input function of the program will aid in the proper development of the lines of codes because it makes it more fun and user-friendly.

Range () function

This function is used in the Python programming language to efficiently handle arithmetic calculations and progressions while working with a series of numbers. Another major application of this function is while working with loops in this programming language. Let us study the example below and how this function is used;

```
>>> range(6)
The resultant output will be: range(0, 6)
```

From the simple code above we can see that the expression ranges the integers from zero to five. In order for you to view the whole list of integers in this list then you will use the following command;

```
>>>list(range(6))
The resultant output will be: [0, 1, 2, 3, 4, 5]
```

Let us study another different way in which we can use the range function but still achieve what we need;

```
>>>range (begin, end)

>>>range (6, 10)
```

```
    The resultant output will be; range (6, 10)
```

If we want to display the list of the range then we will type in the command;

```
>>> list (range(6, 10))
[ 6, 7, 8, 9, 10 ]
```

In the next example, we are going to introduce a new style of ranging where there is a change in the incrementing of numbers by using the step as shown below;

```
  >>>range(begin, end, step)

>>> range(20, 81, 5)
the resultant output will be: range(20, 81, 5)
```

When we call the function, the following sequence of numbers will appear;

```
>>> list (range (20, 81, 5))
[20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70, 75, 80]
```

Print () Function

Initially, this was not a function in the other versions of python (Python 1 and Python 2). It was upgraded into a function in the latest version of python that is the python 3. As a result, you are supposed to enclose the parameters to be printed using the round brackets or parentheses. The following are examples of how the function is used;

```
print(“This is an illustration of Python 3 print function)
print(d)
print(8)
```

The caution to take while printing a big number of values is to separate them using the commas just as shown below;

```
x = 3.1423
y = “year”
z = 46

print (“x = “, x, y, z)
The resultant output will be:
x = 3.1423 year 46
```

Other functions

The abs () function

This is a type of function that returns an absolute result of a single number.

It works by evaluating an integer as an argument and then outputting a positive value from it. The following are examples of how it is used;

```
>>> abs (-11)
the resultant output will be; 11
>>> abs(5)
10
```

In a situation where complex numbers are used, the result will be the magnitude of the numbers as shown below;

```
>>> abs (3 + 4j)
5.0
```

The max () function

This function is used to compare two or more expression or arguments and hence output the largest of the expression or arguments as shown below;

```
>>> max(10, 11, 7, 15)
the resultant output will be: 15
>>> max(-2, -5, -45, -42)
the resultant output will be: -2
```

The min () function

This function works in the opposite way of the max () function. It samples out the least in the argument or expressions then it prints it out as shown below;

```
>>> min(23, -109, 5, 2)
the resultant output will be: -109
>>> min(7, 26, 0, 4)
the resultant output will be: 0
```

The type () function

This is one of the special types of function that is used in classifying the data type that exists in the argument. The following are examples of the same;

```
>>> type("This is a string")
the output will be:<class 'str'>
>>> type(12)
the output will be: <class 'int'>
>>> type(2 +3j)
the output will be:<class 'complex'>
>>> type(215.65)
```

```
the output will be: <class ‘float’>
```

The lens () function

This function is used to evaluate the size of an object or count the number of items that exist in a given list of an argument or expression as shown below;

```
>>> len(“pneumonouloscopicsilicovolcanokoniosis”)
the output will be: 37
>>> s = (“winter”, “summer”, “fall”)
>>> len(s)
the output will be: 3
```

In summary, there is a lot of other built-in functions of Python that play different roles. We are going to list them below so that you can be aware of their existence;

Abs	Classmethod	Filter	Id	Map	Print	
All	Compile	Float	_import_	Max	Repr	
Any	Complex	Format	Input	Memoryview	Reversed	
Ascii	Delattr	Frozenset	Int	min	Round	
Bin	Dict	Getattr	Isinstance	Next	Set	Zip
Bool	Dir	Globals	Issubclass	Object	Setattr	Var
Bytearray	Divmod	Hasattr	Iter	Oct	Slice	Typ
Bytes	Enumerate	Hash	Len	Open	Sorted	Tup
Callable	Eval	Help	List	Ord	Staticmethod	Sup
Chr	Exec	Hex	Locals	Pow	Str	Sun

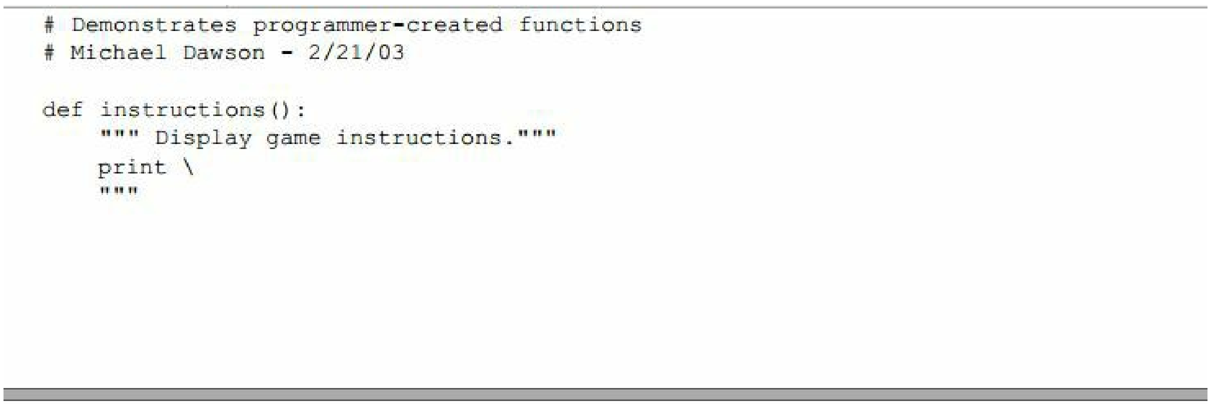
Note: It is advised that while writing the functions that you do not forget to write the round bracket so that it can be recognized as a function and not a word.

Creating functions

Functions can be defined as a block of statements that work to achieve a specific goal and can be used repeatedly in the program. They can also be called, procedures, routines, subprograms and even subroutines.

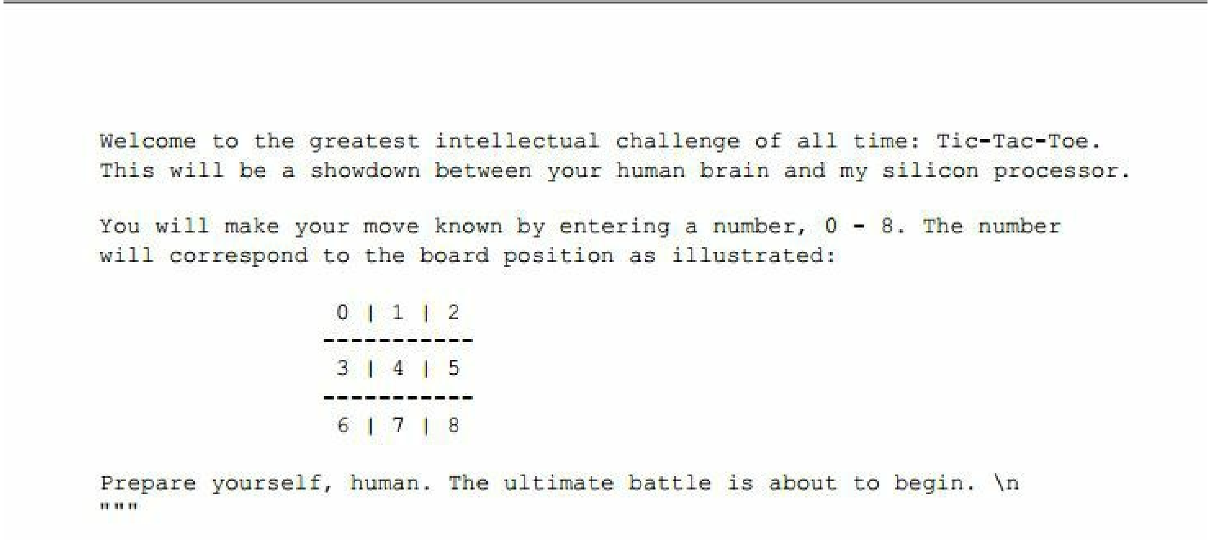
We have interacted with various in-built functions in the Python programming language in the above section. However, sometimes these functions are not adequate for us and we need to create new ones for complex operations. Python allows us to create new functions that can suit our needs. These functions work efficiently as those that are inside the library of the python programming language. In addition, there are many advantages that accompany user-defined functions. The main advantage is the flexibility to break large piece of code to a small manageable piece of code. This makes the programming process to be very easy and enjoyable to work on.

The following is an example of a program of a tic TAC game with user-defined functions. Study it carefully;



```
# Demonstrates programmer-created functions
# Michael Dawson - 2/21/03

def instructions():
    """ Display game instructions."""
    print \
    """
```



```
    Welcome to the greatest intellectual challenge of all time: Tic-Tac-Toe.
    This will be a showdown between your human brain and my silicon processor.

    You will make your move known by entering a number, 0 - 8. The number
    will correspond to the board position as illustrated:

                    0 | 1 | 2
                    -----------
                    3 | 4 | 5
                    -----------
                    6 | 7 | 8

    Prepare yourself, human. The ultimate battle is about to begin. \n
    """
```

How to define a created function

From the example above, the function is simply defined as; “def instructions():”

The main function of this line is that it informs the computer that the following functions are used as functions immediately after the word “Instructions”. In other words, when the block of statement is invoked then the function “instructions ()” is always executed. The whole lines of codes and the instruction function is referred to as “Function definition”.

It is important to note that the above illustrated defined function does not execute the code but it gives us the insight of what the function will do. On the other end, when the computer sees the defined function it marks it and schedules it so that it can be used later. It only executes the function once a call has been invoked or the function has been called.

In summary, this is the way that you are supposed to follow when you have created your own function and you need to define it so that the computer can recognize it and run it when you invoke the function. Let us make a brief summary of the process so that you can get to understand further.

Begin with "def", then type in the suitable name of your function. After the name you insert the round parentheses () immediately followed by a colon and then you can write the indented lines of codes. You should be cautious in naming the function by using the guidelines in naming the variables. The most important thing is to name it in the direction of what it is meant to accomplish in the program.

The following are examples of created functions;

Example 1

```
def love_cakes():
print "I love cakes!"

the output is: I love cakes!
```

Example 2

In this example, we used the number as a parameter existing in the function absolute_integer. Hence it will be the variable name as well as it will hold the value evaluated in the expression.

```
def absolute_integer(number):
if number >= 0:
return number
else:
return -number
print(absolute_integer(4))
print(absolute_integer(-6))
```

The resultant output will be

```
4

6
```

Example 3

The example below shows an illustration of a function that utilizes the looping expression.

```
def shutdown(yn):
if yn.lower() == “y”:
return(“Closing files and shutting down”)
elif yn.lower() == (“n”):
return(“Shutdown cancelled”)
else:
return(“Please check your response.”)
print(shutdown(“y”))
print(shutdown(“n”))
print(shutdown(“x”))
```

The resultant output will become;

```
Closing files and shutting down
Shutdown canceled
Please check your response.
```

NOTE: USER-DEFINED FUNCTIONS ARE ABLE TO ACCOMMODATE MORE THAN ONE PARAMETER.

The example below shows a function that takes more than one parameter in order to complete some calculations.

```
def calculation(a, b):
return a * b + 2
print(calculator(2,6))
print(calculator(3,7))
```

The resultant output of the function becomes;

```
14

23
```

Can functions invoke other functions?

It is very true that apart from functions performing many other functions, they can call other functions too. Let us study the examples below in order to see how this can be possible.

```
def mem_total(n):
return n * 4
def org_total(m):
return mem_total(m) + 5
```

In order for us to monitor the different activities done by the codes above, we will key in the following commands;

```
print(org_total(2))
print(org_total(5))
print(org_total(10))
```

The resultant output will be:

```
11

20

35
```

How to document your function

In this section we will learn of a special way of how to comment or document our own defined functions. We shall use a documentation string well known as Docstring for documenting the user-defined functions.

The Docstring is a “triple-quoted-string” that is supposed to be used in your first line during the creating of your own function. In the case of simple functions, you can be at liberty of explaining what it does from the first line. However, the function can just work efficiently without the placement of the Docstring but I would like to encourage you always use it so that it can be perfect for documenting and storage for future reference. In addition it can pop up during invoking of the function as compared to an interactive session.

How to invoke/call a user-defined function

In this section, it is very similar to how we call the in-built python functions. You are required to use the appropriate name of the function and the set of round brackets as shown below;

```
Instructions ()
```

In summary, the computer is able to identify the function and hence execute the function each time it is invoked. The process of writing and defining your own functions is known as Abstraction.

Using the Namespaces in Global Variables and Constants

In a simple definition, namespaces can be related to components that distinguish or separate different parts of the program from each other. For instance, in order to distinguish different functions you will use the namespaces. It is true to conclude that each function contains its own namespace.

Another simple illustration of how namespaces work is to see them as cars. It is also important to bring in the concept of encapsulation as the tinted windows of the car for privacy purposes. In this scenario, it is possible to see things inside the car while you are inside the car and not outside the car because of the tinted windows. In simple terms, when you inside the function, you can access and manipulate all the variables that are inside that particular function. When you are outside the function, you can access nor manipulate any of the variables that exist in the same function. You are reminded that when you are outside the function then you must be inside the global namespace.

In summary, the global variables do form the platform on where to exercise encapsulation.

How to read a global variable from the function side

It is important to remind ourselves that we can read the values of any global variable from any point of a global namespace. In the same case, we can read the global variables from the inside of the functions just as we saw in the illustration of a car with tinted windows. The only disadvantage that exists at this point is that you cannot change the values in the variables from within the function. In other words, you are able to see the global variables but you can neither access nor manipulate the variables.

The Concepts of Shadowing

This is the process of giving a global variable the same name as that of the function. When this is done, we say that shadowing of the global variable is done by the function. This does not allow you to change anything on the function but you can only change in the local variable.

NOTE: it is not a good idea to perform shadowing because it often leads to confusion. The sharing of names can be distractive because you might mistake a function for a global variable. It is good to be creative so that you give the global variables different names from the one that is assigned to the function.

How do we manipulate global variables from within the function?

In order for the programmer to gain access to the global variable, he or she must use the keyword “global”. As a result, the function can have complete access to the global variable and hence perform the specific changes that are necessary.

In conclusion, the scope and objective of the local variables is usually the place where the program sections are verified. The life of a local variable is usually in computer memory. It is usually defined by the length of time that it is sustained inside the memory. When the function is returned, it coincides with the length of the local variable.

CHAPTER SIX: LOOPS AND CONDITIONAL STATEMENTS

Introduction

In the practical aspect of the common world, there are times in which we have to make tough decisions as well as easy ones. You can decide to make a decision depending on either past experience or totally a newer experience. The same situations also occur in programming fields.

The conditional statements and the loops are common parts of the programming languages that exist in the modern world. They are commonly used to perform calculations and actions that are based on conditions of true or false values. Some of the conditional statements include the if-then-else, while and others. In this section, we are going to work on conditions of either truth or false outcomes. It is important to note that in the Python programming language, any answer that is zero or null is always a false answer; otherwise it is true.

On the other end, the looping part creates the repetitive aspects of conditional statements. In Python programming language, there is only two provisions of the loops which include “the while” and “the for-loop”. They are used to allow the block of codes to repeat several times as they execute until a condition is satisfied.

The Conditional Statements

The conditional statements can also be referred to as decision structures. We are going to consider some of the decisive structures that are used in making the decisions in Python.

The IF statement

This type of conditional statement consists of a single Boolean argument or expression which is accompanied by a statement that will be executed if the result of the Boolean is true. The following is an example of the same;

Example 1

```
age = 23
If (age == 23):
print(“The age is 23”)
print(“Have a good day!”)
```

The output of the code on execution will be;

```
The age is 23
Have a good day!
```

Understanding our code

From the example above, the keyword “if” is going to tell the computer that the following piece of code that is written is a decision or conditional statement. It works by evaluating the condition if it is true or not. If the condition becomes true then it prints the code but if it is false then the computer ignores and moves on to the next piece of code.

The If-Else statement

This type of conditional statement is used when there is a range of conditions to be analyzed. For instance, when the first condition is true and there are other steps or conditions to be evaluated then this conditional statement is appropriate. In order to understand this conditional statement well then we will study the structure below;

```
if condition1:
block1_statement
elif condition2:
block2_statament
else:
block3_statement
```

Understanding our structure

If the first condition becomes true then its corresponding statement will be executed instantly. On the other end, if it is false then the computer will shift to execute the next condition. The same pattern continues for the next block of code. If it turns out that the next block was also false then the next block of code will be executed until the options are over.

The following is an example to illustrate the conditional statement;(the example has a user-defined function)

```
def your_choice(answer):
if answer > 5:
print(“You are overaged.”)
elif answer <= 5 and answer >1:
print(“Welcome to the Toddler’s Club!”)
else:
print(“You are too young for Toddler’s Club.”)
print(your_choice(6))
print(your_choice(3))
print(your_choice(1))
print(your_choice(0))
```

The resultant output for the piece of code above will be;

```
You are overaged.
None
Welcome to the Toddler’s Club!
None
You are too young for Toddler’s Club.
None
You are too young for Toddler’s Club.
None
```

Inserting the Elif Statement

The decision control structures that are used as conditional statements may branch out to have many elif branches. The special thing about this statement is that it allows you to confirm if a few of the statements is actually true. The following is the structure of how the elif statement is supposed to be used;

```
if expression1:
statement(s)
elif expression2:
statement(s)
elif expression3:
statement(s)
else:
```

```
statement(s)
```

The following is a direct application of the syntax above. Study it carefully and mark out the elif statements;

```
print("Let's enjoy a Pizza! Ok, let's go inside Pizzahut!")
print("Waiter, Please select Pizza of your choice from the menu")
pizzachoice = int(input("Please enter your choice of Pizza:"))
if pizzachoice == 1:
print('I want to enjoy a pizza napoletana')
elif pizzachoice == 2:
print('I want to enjoy a pizza rustica')
elif pizzachoice == 3:
print('I want to enjoy a pizza capricciosa')
else:
print("Sorry, I do not want any of the listed pizza's, please bring a
Coca Cola
for me.")
```

The elif statements are inserted to bring more platforms for the addition of conditions as shown in the example below;

```
def your_choice(answer):
if answer > 5:
print("You are overaged.")
elif answer <= 5 and answer >2:
print("Welcome to the Toddler's Club!")
elif answer == 2:
print("Welcome! You are a star member of the Toddler's Club!")
else:
print("You are too young for Toddler's Club.")
print(your_choice(6))
print(your_choice(3))
print(your_choice(1))
print(your_choice(0))
print(your_choice(2))
```

The resultant output of the block of codes above will be:

```
You are overaged.
None
Welcome to the Toddler's Club!
None
```

```
You are too young for Toddler’s Club.
None
You are too young for Toddler’s Club.
None
Welcome! You are a star member of the Toddler’s Club!
None
```

It is very important to note that some other books and resource centers call this feature THE NESTED IF. They are majorly used to make more opportunity to use the” if-else” functions. They create more platforms where you can introduce more conditions to the program.

Before we conclude on decisive control structures we are going to study about ways in which we can determine if our conditional statements are true or false;

Expressions	The Condition will be true if;
A=B	A is equal to B
A!=B	A is not equal to B
A<B	A is less than
A>B	A is greater than B
A<=B	A is less than or equal to B
A>=B	A is greater than or equal to B

The following is an example that illustrates the information in the table above;

```
age = int(input(“Enter your age:”))
if (age <=18):
print(“You are not eligible for voting, try next election!”)
print(“Program ends”)
```

The resultant output will become;

```
Enter your age :18
You are not eligible for voting, try next election!
Program ends
Enter your age :35
Program ends
```

Looping

At this point in the book, we have seen how we can expand our program in various ways in order to manipulate and access information. As a result, the number of variables and functions continue to grow each and every step. In this section we are going to consider a new technique of working with a sequence of information that is known as looping.

Looping statements are used in a situation where you want to rerun or rewrite a block of statements in a repeated number of times. You are usually required to tie the statement into a loop so that it can appear the number of times that you need in the system. In the Python programming language, you are allowed to apply two to three types of loops that will make you repeat your codes to the number of your satisfaction. The three types of loops in python language include;

- The Nesting loops statements
- The for-loop statements and
- The while-loop statements

The While loop

This type of control loop is only appropriate for situations where you want to run some block of code for a fixed amount of time before it comes to a stop. In other words, the while loop works on a target statement and in most times, it works when the condition is true. The block of statement continues to run if the statement stays true. When it becomes false, it goes back to the first block of the statement. The following is the basic structure of a while loop;

```
While condition
     statement
statement
```

Let us study the example below of the while loop and see how the looping statement works;

Example 1

```
counter = 0
while (counter < 10):
print('The count is:' , counter)
counter = counter + 1
print("Done!")
```

The output will be:

```
l.py
```

```
The count is: 0
The count is: 1
The count is: 2
The count is: 3
The count is: 4
The count is: 5
The count is: 6
The count is: 7
The count is: 8
The count is: 9
Done!
```

Example 2

```
counter = 1
while(counter <= 3):
principal = int(input("Enter the principal amount:"))
numberofyears = int(input("Enter the number of years:"))
rateofinterest = float(input("Enter the rate of interest:"))
simpleinterest = principal * numberofyears * rateofinterest/100
print("Simple interest = %.2f" %simpleinterest)
#increase the counter by 1
counter = counter + 1
print("You have calculated simple interest for the 3rd time!")
```

Assignment

Follow up on example 2 and write down the expected output of the while loops when the code is executed.

When do we use the pass statement?

This is a very good question that is asked by most people who learn the python looping statements. This type of statement simply informs the computer to completely do nothing at the point where it has been written. However, many people tend to confuse this part as the end of the program. This is not the end of the program because the interpreter will continue to execute other statements after the pass statement. This statement is used in places where the Python programmer needs to insert a line and the function or the program does not need to act on the line.

The following is the syntax of the pass statement;

```
def function_name(x):
pass
```

The For-Loop

The looping criterion is used in a situation where you want the loop to repeat for a certain number of times before it terminates. The main keyword that introduces this type of loop is “for” which is followed by a variable name having the value to be evaluated. The structure or the syntax of the for-loop is shown below;

```
for variable in list:
    statements
else:
    statements
```

The following is an example of the application of for-loop in Python;

```
pizza = [“New York Style Pizza”, “Pan Pizza”, “Thin n Crispy
Pizza”, “Stuffed Crust
Pizza”]
for choice in pizza:
if choice == “Pan Pizza”:
print(“Please pay $16. Thank you!”)
print(“Delicious, cheesy ” + choice)
else:
print(“Cheesy pan pizza is my all-time favorite!”)
print(“Finally, I’m full!”)
```

The resultant output when the code is executed will be:

```
Delicious, cheesy New York Style Pizza
Please pay $16. Thank you!
Delicious, cheesy Pan Pizza
Delicious, cheesy Thin n Crispy Pizza
Delicious, cheesy Stuffed Crust Pizza
Cheesy pan pizza is my all-time favorite!
Finally, I’m full!
```

Assignment

In the example below, analyze and study the output of the code when it is executed and hence develop a similar program;

```
#write a multiplication table from 1 to 10
For x in xrange(1, 11):
For y in xrange(1, 11):
Print ‘%d = %d’ % (x, y, x*x)
```

The expected output will be:

```
1*1 = 1
1*2 = 2
1*3 = 3
1*4 = 4
```

When do we use the break statement?

In Python, this type of statement is used to end the presently running loop and shift to the next block of statements. It informs the interpreter that time for running the current loop is over and hence it needs to terminate and move to the next block of statement immediately after the loop. On the other end, it can also be used to stop the running of the else statements. It is also important to note that it is placed after the print function.

The following example shows how the break statement is used in Python programming language;

```
pizza = [“New York Style Pizza”, “Pan Pizza”, “Thin n Crispy
Pizza”, “Stuffed Crust
Pizza”]
for choice in pizza:
if choice == “Pan Pizza”:
print(“Please pay $16. Thank you!”)
break
print(“Delicious, cheezy ” + choice)
else:
print(“Cheezy pan pizza is my all-time favorite!”)
print(“Finally, I’m full!”)
The Python Shell will now show:
Delicious, cheezy New York Style Pizza
Please pay $16. Thank you!
Finally, I’m full!
```

When do we use the continue statement?

This is also another special type of statement that is used to restore the control of the program back to where it started. It is applicable in both the while and the for-loops. We will use the example below to illustrate how the continue statement is used in Python;

```
pizza = [“New York Style Pizza”, “Pan Pizza”, “Thin n Crispy
Pizza”, “Stuffed Crust
Pizza”]
for choice in pizza:
if choice == “Pan Pizza”:
```

```
print(“Please pay $16. Thank you!”)
continue
print(“Delicious, cheesy ” + choice)
else:
print(“Cheesy pan pizza is my all-time favorite!”)
print(“Finally, I’m full!”)
```

The resultant output for the code will be:

```
Delicious, cheesy New York Style Pizza
Please pay $16. Thank you!
Delicious, cheesy Thin n Crispy Pizza
Delicious, cheesy Stuffed Crust Pizza
Cheesy pan pizza is my all-time favorite!
Finally, I’m full!
```

Integration of the For-loop with the Range function

This is a special combination that occurs between a looping statement and function. The combination of the two is majorly done in order to obtain values that are suitable to be evaluated most especially by the loop.

The following is a simple example of how the combination is applied in Python;

```
x = 50
total = 0
for number in range(1, x+1):
total = total + number
print(“Sum of 1 until %d: %d” % (x, total))
```

The output of the lines of codes above when executed will be;

```
l.py
Sum of 1 until 50: 1275
```

In summary, the looping statements are used to help us to achieve various tasks in almost all the programming languages. It is necessary to know the type of looping statement you require in order to achieve what you want.

Practical Exercise

Determine the output of the following piece of code and identify the parts and type of looping statement that has been used to achieve the lines of codes;

```
# Measure some strings:
words = [‘apple’, ‘mango’, ‘banana’, ‘orange’]
```

```
for w in words:
print(w, len(w))
```

CHAPTER SEVEN: MODULES, INPUT AND OUTPUT

INTRODUCTION

When you define the functions and variables in python programming language and you close the python interpreter then the definitions become lost. This will be a great disadvantage to you if you are writing very long programs. In order for you to write the long program, you will be required to use the text editor and transfer it to the interpreter upon finishing. This concept is referred to as Scripting. You may want to break down your long program into short manageable codes that will be easier for you to edit and maintain.

In Python, the scripting and definition of variables and functions are done in a special way and stored in a file so that it can be used in the interpreter as a script. The file that is used as a script is referred to as a Module. Modules can be manipulated by other modules. There is always a slot in python for the main module. The main module can be defined as a group of variables that can be manipulated from the script.

In other words, a module can be described as a file that contains the python definitions as well as a block of statements. For instance we are going to create a program in the text editor as shown below in order to generate a script from it.

```
# Fibonacci numbers module
def fib(n): # write Fibonacci series up to n
a, b = 0, 1
while b < n:
print(b, end=' ')
a, b = b, a+b
print()
def fib2(n): # return Fibonacci series up to n
result = []
a, b = 0, 1
while b < n:
result.append(b)
a, b = b, a+b
return result
```

We will import this from the text editor by the use of the import command

```
>>> import fibo
```

The module that will be generated is the one that is entered from the text editor directly having the names of functions. The module name will become fibo. As a result you will obtain the functions shown below;

```
>>> fibo.fib(1000)
1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987
>>> fibo.fib2(100)
[1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]
>>> fibo.__name__
'fibo'
```

We can also take the option of having a local name on the function as shown below;

```
>>> fib = fibo.fib
>>> fib(500)
1 1 2 3 5 8 13 21 34 55 89 144 233 377
```

The module can also be likened to the function definitions. This is the platform where it can contain many executable statements within the module. The main objective of the statements contained in the module is to initialize the module so that when the module name is encountered then it is referred to as an imported statement. In other words, the executable statements can be run if the file is imported as a script.

It is very true that the modules have their own symbols table. The symbols tables are private to each and every module. The main advantage that accompanies this type of private symbol tables is that the programmer is free to use any global variable without interfering with the same global variable properties. On the other end, modules can actually acquire other module properties or parameters by simply importing the child module. In addition, while importing other modules the programmer is required to place the name of the child module at its private symbol table.

Let us study the table below and examine the importation of modules and the allocation of names to the symbol table.

```
>>> from fibo import fib, fib2
>>> fib(500)
1 1 2 3 5 8 13 21 34 55 89 144 233 377
```

Understanding our code

From the above example, fibo has not yet been defined. I want to believe that you have observed the importation before we move to the next example. I would like to advise that if you haven’t understood then you should not rush to the next example.

Let us also observe the following example to deepen our understanding in importing of modules.

```
>>> from fibo import *
>>> fib(500)
1 1 2 3 5 8 13 21 34 55 89 144 233 377
```

From the above example, the action that is expected is the importation of all the names with the exception of those that begin with the underscore symbol

However, most of the python programmers do not fancy this type of processes because it requires keen usage of commands and exploitation of function definitions. It also leads to poor reading and organization of codes in the interpreter and hence most programmers tend to avoid it.

On the other end, I would recommend you try and utilize this feature because it helps us during creation and application of interactive sessions during programming.

How do we execute modules as scripts?

Python modules can be executed and imported at the same time in order to be used in another module of a different file. In order to achieve this operation we will have to type in the following code;

```
if __name__ == "__main__":
import sys
fib(int(sys.argv[1]))
```

The above code will enable the file to be executed as a script because the code is saved as an imported module and hence the command line is executed when the file is read as the “main” file just as shown below;

```
$ python fibo.py 50
1 1 2 3 5 8 13 21 34
```

On the other side, if the code is not executed then it means that the code was not imported correctly and hence it will appear as shown below;

```
>>> import fibo
>>>
```

The major application of this feature is when the programmer wants to interface the module for the purposes of testing the code.

The Standard Module

The python programming language is equipped with a list of standard modules that are stored in the python library. The python library is often referred to as the “Python Library Reference”. We can confidently say that some modules are also encrypted in the interpreter in order to provide access to other operations not belonging to the primary language. Some of these operations include system calls and others. they can be in-built or user-defined operations. For instance, such encrypted modules can be labeled as “sys”. The example below shows the application of some of the standard modules;

```
>>> import sys
>>> sys.ps1
'>>> '
>>> sys.ps2
'... '
>>> sys.ps1 = 'C> '
C> print('Yuck!')
Yuck!
C>
```

From the above example we can see the variables sys.ps1 as well as the sys.ps2 which are used as primary and the secondary prompts respectively. The variables above become defined once the interpreter is only in the interactive mode.

THE INPUT AND THE OUTPUT

Introduction

In this sub-chapter, we are going to study some of the ways in which the output of the codes can be stored, presented in a human-readable format and even kept for future use and modification.

There are a number of ways in which we can output and express values during programming. Some of the ways that we can use to achieve this are by;

- using the expression statements,
- using the write () function together with the methods having the objects and finally
- The use of the print () function.

In order to present your lines of codes in a formatted way that can also be understood by other programmers you can use the following two formatting procedures;

- Organization of the strings. This is also referred to as “String handling”. It achieved by slicing and concatenation operations in order to create a formatted layout.
- The use of literals in string arrangements. In this type of formatting, we do not have to be worried because the python programming language offers us a template in which will help us in substitution of the strings.

The main concern that will rise above our heads is the conversion of values into strings so that it can fit the above formatting styles. Luckily enough, the python programming language offers simpler ways to convert values to strings. We can either use the repr () function or the str () function to convert the values into strings.

The Str () function works by returning the human-readable format of the values whereas the repr () function works by a generation of value representations that can only be recognized by the interpreter. However, there are situations in which the STR () function will act as the repr (). Such situations include when the objects in place cannot be recognized in a human-readable format.

The following are some examples;

```
>>> s = 'Hello, world.'
>>> str(s)
'Hello, world.'
>>> repr(s)
"'Hello, world.'"
>>> str(1/7)
'0.14285714285714285'
>>> x = 10 * 3.25
>>> y = 200 * 200
>>> s = 'The value of x is ' + repr(x) + ', and y is ' + repr(y) + '...'
>>> print(s)
The value of x is 32.5, and y is 40000...
>>> # The repr() of a string adds string quotes and backslashes:
... hello = 'hello, world\n'
>>> hellos = repr(hello)
>>> print(hellos)
'hello, world\n'
>>> # The argument to repr() may be any Python object:
... repr((x, y, ('spam', 'eggs')))
```

```
"(32.5, 40000, ('spam', 'eggs'))"
```

In the above example, we can recognize the spacing between the columns as an illustration of how the print () function works. We can also see the Str () functions such as the str.rjust, str.ljust and str.center ().

Let us also study other features in the example below;

```
>>> for x in range(1, 11):
... print(repr(x).rjust(2), repr(x*x).rjust(3), end=' ')
... # Note the use of 'end' on the previous line
... print(repr(x*x*x).rjust(4))
...
1 1 1
2 4 8
3 9 27
4 16 64
5 25 125
6 36 216
7 49 343
8 64 512
9 81 729
10 100 1000
>>> for x in range(1, 11):
... print('{0:2d} {1:3d} {2:4d}'.format(x, x*x, x*x*x))
...
1 1 1
2 4 8
3 9 27
4 16 64
5 25 125
6 36 216
7 49 343
8 64 512
9 81 729
10 100 1000
```

The example above is about different ways in which we can define a table containing the squares and cubical values.

The methods that we have studied and observed in the examples do not have any other effect apart from creating and returning a new string. It does not matter if the input string is longer than usual because it will not be truncated or shortened but returned the way it is.

The basic format or syntax of str. Format () is shown below;

```
>>> print('We are the {} who say "{}!"'.format('knights', 'Ni'))
We are the knights who say "Ni!"
```

The brackets that contain the characters in the syntax above are referred to as "Format fields". They can be replaced over by objects which should be passing through the str.format method as shown below;

```
>>> print('{0} and {1}'.format('spam', 'eggs'))
spam and eggs
>>> print('{1} and {0}'.format('spam', 'eggs'))
eggs and spam
```

The name of the arguments is used in situations where the keywords of the str.format() method is often used

```
>>> print('This {food} is {adjective}.'.format(... food='spam',
adjective='absolutely horrible'))
This spam is absolutely horrible.
```

In python programming language, it is recommended that we use the keyword and positional arguments together just as shown below;

```
>>> print('The story of {0}, {1}, and {other}.'.format('Bill',
'Manfred', other='Georg'))
The story of Bill, Manfred, and Georg
```

On the other end, the ascii () function which is usually written as "!a" and the repr() function written as "!r" can also be used extensively to convert values into strings for formatting purposes;

```
>>> contents = 'eels'
>>> print('My hovercraft is full of {}.'.format(contents))
My hovercraft is full of eels.
>>> print('My hovercraft is full of {!r}.'.format(contents))
My hovercraft is full of 'eels'.
```

The field names are immediately followed by the symbols ":" in order to allow control of the technique of conversion of values just as shown below;

```
>>> import math
>>> print('The value of PI is approximately
{0:.3f}.'.format(math.pi))
```

```
The value of PI is approximately 3.142.
```

From the example above, we have seen that we are able to convert the summation of values above. In other instances, we can decide to truncate some values from the table so as to make it look presentable. This is achieved by passing the integer values through the semicolon symbol as shown below;

```
>>> table = {'Sjoerd': 4127, 'Jack': 4098, 'Dcab': 7678}
>>> for name, phone in table.items():
... print('{0:10} ==> {1:10d}'.format(name, phone))
...
Jack ==> 4098
Dcab ==> 7678
Sjoerd ==> 4127
```

Sometimes the strings that we have in our programs can end up being too long but also necessary. The python programming language allows us to sustain these long strings by referencing the variables considering the name but not the value positioning. This is basically to avoid truncating of the strings. This is achieved by;

a. The use of square brackets and dictionary;

```
>>> table = {'Sjoerd': 4127, 'Jack': 4098, 'Dcab': 8637678}
>>> print('Jack: {0[Jack]:d}; Sjoerd: {0[Sjoerd]:d}; '
... 'Dcab: {0[Dcab]:d}'.format(table))
Jack: 4098; Sjoerd: 4127; Dcab: 8637678
```

b. The use of the “**” notation

```
>>> table = {'Sjoerd': 4127, 'Jack': 4098, 'Dcab': 8637678}
>>> print('Jack: {Jack:d}; Sjoerd: {Sjoerd:d}; Dcab:
{Dcab:d}'.format(**table))
Jack: 4098; Sjoerd: 4127; Dcab: 8637678
```

In summary, the above methods are extensively useful because they return values to the dictionaries that contain the local variables

The Old formatting style

In the recent past, Python used the % operator in the formatting of strings. It worked by interpreting the left side of the argument compared to a sprint () function style in the reflection of the right side. The following example illustrates how the string would be returned in the old formatting style;

```
>>> import math
>>> print('The value of PI is approximately %5.3f.' % math.pi)
The value of PI is approximately 3.142.
```

The File methods and Objects

In most of the examples in this chapter, we are likely to have assumed that we had a file that was existing and was called “f”. in order for us to read and access the contents of file f then we invoke the f.read (size). The contents that have been accessed and read will be returned as strings in either bytes or text mode. The only caution to take is that the file to be accessed is not as big as the program or occupies a large space because it can end up hanging. On the other side, when the program reached the end of the file then it will have to output the string as empty. (“”)

```
>>> f.read()
'This is the entire file.\n'
>>> f.read()
''
```

In summary, the arrangement and neatness of a program are required. This is the reason why the formatting of the strings is recommended. This is to make the programs more presentable and appealing to the eyes of other programmers.

I believe that you have learned a lot in this chapter and it will assist you in the formatting and creation of interactive programs. I understand that most of the people do not like to use the modules but I believe that this book has covered it well and will assist you in applying it. You can also refer to other books so that you can know and understand the working of different input and output formatting procedures.

Practical exercises

1. Study and analyze the reading and writing of files
2. Do simple research and make short comprehensive notes on saving structures using the json () function.

CHAPTER EIGHT: EXCEPTIONS AND ERRORS

Introduction

In any programming language, errors are bound to occur since we make various mistakes while typing or running the blocks of statements. If you have been keen on running some of the examples then you must have encountered errors or mistakes due to spacing or otherwise. It is very important as a beginner to learn to manage and correct errors as they occur. This skill is known as “Interception and Error handling skills”

In this chapter we are going to study the two major types of errors which include;

i. The Syntax error
ii. The Exceptions

The Syntax Errors

They are also referred to as the “Parse errors”. This is a common error most especially encountered by beginner Python coders. Below is an example;

```
>>> while True print('Hello world')
File "<stdin>", line 1
while True print('Hello world')
^
SyntaxError: invalid syntax
```

Understanding our Syntax error

The error above is detected after the repetition of the line above. The error was detected by the interpreter at the print () function. The error that the programmer did was that he or she forgot to add the colon symbol immediately after the function.

The Exceptions

These are errors that are made during the execution of the program. It is very funny that your program may have the correct syntax but still have an error during execution. This is not strange because these errors are not very fatal to the program but can be easily controlled. In this chapter, we will try to interact with such situations so that we can be confident when solving the problem whilst it occurs. The following is an example of such errors;

```
>>> 10 * (1/0)
```

```
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
ZeroDivisionError: division by zero
>>> 4 + spam*3
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
NameError: name 'spam' is not defined
>>> '2' + 2
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
TypeError: Can't convert 'int' object to str implicitly
```

Understanding our exceptions

The type of exception that is encountered is usually indicated in the error message that pops up in the last line of the block of statements. Examples of exception errors include; NameError, ZeroDivisionError and TypeError. It is also important to note that the exception type can be derived from the string which was affected. However, this only occurs for in-built exceptions and not the user-defined exceptions. There are also in-built identifiers which are referred to as "standard exceptions".

How do we handle Exceptions?

Since we are programmers, we can decide to design programs that can solve the occurrence of exceptions. For instance, the example below illustrates how a program can prompt a user to enter only valid answers.

```
>>> while True:
... try:
... x = int(input("Please enter a number: "))
... break
... except ValueError:
... print("Oops! That was no valid number. Try again..."
```

The use of a try statement

The following is the procedure on how the try statement helps the program to achieve an exception-free error;

- The try-statements are executed.
- In a case where there is no exception error, the except-clause will be skipped and the action of the try statement will be terminated.
- In the case that the exception error occurs, the try clause would

have been executed and hence the next block of statements will be skipped. The except-statement will be executed if it matched the exception error.

- There are cases where the exception error is totally different from the except-statement. In such cases, it is passed to the next try-statements until a matching except clause is located. In extreme cases where the exception-statement is not found, the error is referred to as “Unhandled exception” and hence the execution is terminated.

It is important to note that a try-clause can contain many corresponding exception-statements. This is to increase the probability of having a handler clause for each and every exception error. The handlers are very specific to the type of exception that occurs. There is no time where the handler can execute an exception of a different type. in addition, a group of exception errors that have occurred can be classified by the except statement as shown below;

```
... except (RuntimeError, TypeError, NameError):
... pass
```

On the other end, classes can be used in except clauses to classify the type of exception just as shown below;

```
class B(Exception):
pass
class C(B):
pass
class D(C):
pass
for cls in [B, C, D]:
try:
raise cls()
except D:
print("D")
except C:
print("C")
except B:
print("B")
```

The wildcards are used in last exception errors. This is done with a lot of caution because it may lead to a bigger error as compared to the exception error. However, it can be positively used in the printing of error messages.

Let us examine the example below;

```
import sys
try:
f = open('myfile.txt')
s = f.readline()
i = int(s.strip())
except OSError as err:
print("OS error: {0}".format(err))
except ValueError:
print("Could not convert data to an integer.")
except:
print("Unexpected error:", sys.exc_info()[0])
raise
```

In addition, the try-except clauses can also have else-statements. If they happen to exist in your try statement, then you will have to use them in all the exception statements too as shown in the example below;

```
for arg in sys.argv[1:]:
try:
f = open(arg, 'r')
except OSError:
print('cannot open', arg)
else:
print(arg, 'has', len(f.readlines()), 'lines')
f.close()
```

The use of the else statement is very beneficial because it will prevent you from picking a wrong exception statement that will cause misappropriation of codes to be corrected. In addition, the exception statement is free to specify the type of variable most especially after obtaining its name. The example below shows an illustration of an exception that has been instantiated;

```
>>> try:
... raise Exception('spam', 'eggs')
... except Exception as inst:
... print(type(inst)) # the exception instance
... print(inst.args) # arguments stored in .args
... print(inst) # __str__ allows args to be printed directly,
... # but may be overridden in exception subclasses
... x, y = inst.args # unpack args
... print('x =', x)
... print('y =', y)
...
<class 'Exception'>
('spam', 'eggs')
('spam', 'eggs')
x = spam
y = eggs
```

In other occurrences, the exception comes with various arguments. For instance, the unhandled exceptions are always accompanied by the descriptions of the arguments usually when they are printed. In the try statement, the exception statements solve the occurred exception errors in consideration of the functions that exist in the codes inside the try clause as shown below;

```
>>> def this_fails():
... x = 1/0
...
>>> try:
... this_fails()
... except ZeroDivisionError as err:
... print('Handling run-time error:', err)
...
Handling run-time error: division by zero
```

The Raise Clause Exception handling

This is a type of special statement that is used by the programmer to force an exception statement to be applied abruptly. Let us study the following example;

```
>>> raise NameError('HiThere')
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
NameError: HiThere
```

The exact place that the raise statement has been applied is indicated in the exception that is forced. In this situation, there must be a class that has been derived from the Exception. If the exception class has been passed it will have to be instantiated by invoking the class constructor but lacking the arguments.

In order to verify if the exception really occurred, you will have to type in the following command lines to re-raise the raise clause;

```
>>> try:
... raise NameError('HiThere')
... except NameError:
... print('An exception flew by!')
... raise
...
An exception flew by!
Traceback (most recent call last):
File "<stdin>", line 2, in <module>
NameError: HiThere
```

Creating your Own Exceptions

In Python, you are allowed to create and name your own exceptions. This section will be defined while learning about the classes that exist in the python programming language. However, it is important to know that exceptions are usually directly or indirectly derived from the Exception classes.

The Exception classes are usually very simple; having clearly defined attributes and works like any other class. The reason for the clearly defined attributes is to make it easy for the handlers to extract information on the exception errors. When a module is generated for raising different errors, a base class is generated that will hold the exception derived from the module.

```
class Error(Exception):
"""Base class for exceptions in this module."""
pass
class InputError(Error):
"""Exception raised for errors in the input.
Attributes:
expression -- input expression in which the error occurred
message -- explanation of the error
"""
def __init__(self, expression, message):
self.expression = expression
self.message = message
class TransitionError(Error):
"""Raised when an operation attempts a state transition that's not
```

```
allowed.
Attributes:
previous -- state at beginning of transition
next -- attempted new state
message -- explanation of why the specific transition is not allowed
"""
def __init__(self, previous, next, message):
self.previous = previous
self.next = next
self.message = message
```

Establishing Cleaning-up Actions

The try clause has an extension of an optional clause that is majorly used for cleaning-up functions which should be immediately executed as shown below;

```
>>> try:
... raise KeyboardInterrupt
... finally:
... print('Good morning, world!')
...
Good morning, world!
KeyboardInterrupt
Traceback (most recent call last):
File "<stdin>", line 2, in <module>
```

The final statement is usually run before the try clause is terminated regardless of an occurrence of exception or not. It is usually referred to as the “Way out” statement as shown in the example below;

```
>>> def divide(x, y):
... try:
... result = x / y
... except ZeroDivisionError:
... print("division by zero!")
... else:
... print("the resultant is", result)
... finally:
... print("executing finally statement")
...
>>> divide(2, 1)
the resultant output is 2.0
```

```
executing finally statement
>>> divide(2, 0)
division by zero!
executing finally clause
>>> divide("2", "1")
executing finally clause
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
File "<stdin>", line 3, in divide
TypeError: unsupported operand type(s) for /: 'str' and 'str'
```

Assignment

1. Distinguish between exceptions and errors
2. Create a program that will be able to test and analyze a beginner’s knowledge of Python files and how to handle an exception.
3. Create a program that will have multiple try and raise statements before the execution of the commands.
4. Which statement is commonly referred to as the “On the way out” statement?
5. Create a simple program that handles any exception that may occur.
6. Mention two instances that exception errors may occur.
7. What are the possible causes of errors in any programming language?
8. Which of the error messages have you encountered with during your programming sessions? Explain the cause and the remedy that you took to correct it.

CHAPTER NINE: CLASSES AND OBJECT-ORIENTED PROGRAMMING

Introduction

In the programming world, the classes have always acted as a house of data and their specific functions. In other words, the classes that are created come along with other attributes such as the new objects and new instances. The instances also give birth to methods which define the class's characteristics.

In Python, the classes that are created have limited semantic and syntaxes. The classes are integrated with the characteristics of object-oriented programming. For instance, the class inheritance criterion is supported by the existence of many base classes, derived classes and methods. It is important to note that the structures of classes are very dynamic since the methods and objects are fitted with multiple kinds of arbitrary data. In addition, the classes are generated on a runtime basis and can be edited or modified later after creation.

Python Objects and Names

Objects can be defined as having individualism. They have multiple names contained in multiple scopes which are linked to a single object. In other programming languages, this property is usually referred to as "aliasing". However, in Python, this property is not well embraced and hence it is often ignored.

On the other end, the mutable objects in python have an added advantage of aliasing. The objects become like the pointers in programming languages.

The Syntax of a Class in Python

The reserved word that is used in the definition of classes is the word "class". The keyword is followed by the name of the class then a colon. You should note that the classes begin with capital or uppercase letters. The following is an example;

```
class Pupils:
    pass
```

In situations where the class being created contains an object then it will be as shown below;

```
class Pupil(object)
```

The_init_ () method

When the instance of a class has been generated, this function is usually invoked. The purpose of invoking this function is to initialize the object of the created class. The object can be identified by the argument called "self". The following are two examples to illustrate the same.

Example 1

```
class Pupils:
  def __init__(self) :
```

Example 2

```
class Employers(object):
  def __init__(self, name, rate, hours) :
```

It is important to note that any function that is used inside a class is referred to as "Method". From the above examples, the function that has been used can be rightfully referred to as a method that initializes the new classes.

The Instant Variable

As we continue to code in the classes that we have built, the number of arguments also increases. As a result, the number of instant variables needs to be increased so that they can be parallel to the existing number of objects. Let us study the example below;

```
class Employees(object):
 def __init__(self, name, rate, hours) :
 name.self = name
 rate.self = rate
 hours.self =hours
```

From the example above, the components that have been placed in the parenthesis are the instant variables. In other words, when the instance of the class Employees has been created, all the components of that class are allowed by the function to access the available variables. The following is an example;

```
staff = Employees("Cayne", 20, 8)
supervisor = Employees("Swight", 35, 8)
manager = Employees("Belinda", 100, 8)
```

When we run the codes and verify the above-written codes below then we will expect the following output;

```
print(staff.name, staff.rate, staff.hours)
print(supervisor.name, supervisor.rate, supervisor.hours)
print(manager.name, manager.rate, manager.hours)
```

The output on display will be:

```
Cayne 20 8
Swight 35 8
Belinda 100 8
```

The following is another example which is very descriptive;

Example 1

```
class Lion:
kind = 'canine' #the class variable that will be shared by all instances
def __init__(self, name):
self.name = name # instance variable unique to each instance
>>> c = Lion('Lido')
>>> d = Lion('Duddy')
>>> f.kind # this is shared by all lions
'canine'
>>> c.kind # this is shared by all lions
'canine'
>>> c.name # unique to d
'Lido'
>>> d.name # unique to e
'Duddy'
```

Example 2

```
class Cat:
def __init__(self, name):
self.name = name
self.tricks = [] # here there is a creation of a new empty list for each
of the cats
def add_trick(self, trick):
self.tricks.append(trick)
>>> c = Cat('Lido')
>>> d = Cat('Cuddy')
>>> d.add_trick('rolling your body over')
>>> f.add_trick('play like you are dead')
>>> d.tricks
['roll over']
>>> e.tricks
['play like you are dead']
```

What are python's Namespaces and Scopes?

A namespace is a transform of names to objects. In other words, we can describe it as mapping the names to get an object. They are usually located in the python dictionary. They are generated or created at different times as well as have different lifespans and duration of stay in the classes. For instance, a global namespace that serves a module is usually created during the definition of the same module. On the other hand, the local namespace that serves in the function is created during the invoking of the same function. Its lifespan is determined when the function is returned or when an exception has been raised.

A scope can be defined as a textual platform in Python whence the user can access the namespace directly. The scopes are defined statically and applied dynamically. The following are types of scopes that can be accessed directly by the namespaces during the execution of a program;

- Innermost scope: it has the local names.
- Enclosing function scope: it has non-global and non-local names.
- Next to the last scope: it has the existing module's global naming.
- Outermost scope: it has the in-built names.

The following is an example of namespaces and scopes in python programming language;

```
def scope_test():
def do_local():
spam = "local spam"
def do_nonlocal():
nonlocal spam
spam = "nonlocal spam"
def do_global():
global spam
spam = "global spam"
spam = "test spam"
do_local()
print("After local assignment:", spam)
do_nonlocal()
print("After nonlocal assignment:", spam)
do_global()
print("After global assignment:", spam)
```

```
scope_test()
print("In global scope:", spam)
```

The resultant output will be:

```
After local assignment: test spam
After nonlocal assignment: nonlocal spam
After global assignment: nonlocal spam
In global scope: global spam
```

The Private Variables

These are instant variables that cannot be directly accessed not until you use objects that exist in other programming languages. It is very hard to access them from python and hence there are a lot of procedures to be followed. For beginning classes, I would prefer that we do not go beyond this point of explanation but just have the following example;

```
class Mapping:
def __init__(self, iterable):
self.items_list = []
self.__update(iterable)
def update(self, iterable):
for item in iterable:
self.items_list.append(item)
__update = update # private copy of original update() method
class MappingSubclass(Mapping):
def update(self, keys, values):
# here there is a provision of a new signature for the update()
# however it does not break __init__()
for item in zip(keys, values):
self.items_list.append(item)
```

Iterators

The Iterators are generally used to loop just as shown below;

```
for element in [1, 2, 3]:
print(element)
for element in (1, 2, 3):
print(element)
for key in {'one':1, 'two':2}:
print(key)
for char in "123":
print(char)
```

```
for line in open("This is myfile.txt"):
print(line, end='')
```

In other words, the Iterators make it pervasive and unified because iterating creates a unique style of accessing objects. The unique style of accessing can be described as clear, convenient and concise. The for-loop is used both in looping and invoking of container objects. The following is a simple illustration of how it works.

```
>>> s = 'xyz'
>>> it = iter(s)
>>> it
<iterator object at 0x00A1DB50>
>>> next(it)
'x'
>>> next(it)
'z'
>>> next(it)
'c'
>>> next(it)
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
next(it)
StopIteration
```

The iterator looping creates a unique behavior in the python programming language. It is also important that we note that the iterator uses the _iter_() method for returning of objects together with the _next_() method. Let us observe the example below;

```
class Reverse:
"""This Iterator is used for looping over a sequence backwards."""
def __init__(self, data):
self.data = data
self.index = len(data)
def __iter__(self):
return self
def __next__(self):
if self.index == 0:
raise StopIteration
self.index = self.index - 1
return self.data[self.index]
```

```
>>> rev = Reverse('spam')
>>> iter(rev)
<__main__.Reverse object at 0x00A1DB50>
>>> for char in rev:
... print(char)
...
maps
```

The Ends and the Odds

These are special types of data types that are used to bundle together the data items that exist in a class. These special data types will be used in advanced levels of python programming language to define empty classes as shown below;

```
class Employer:
pass
john = Employer() #here, there is Creation of an empty employer
record
# here there is Filling of the fields of the empty employer record
john.name = 'John Park'
john.dept = 'computer lab'
john.salary = 10000
```

This topic should not confuse you as a beginner but it is a nice way to prepare for the future.

The Generators

These are special tools that are used in the creation of iterators. They work almost exactly like the regular functions incorporating the yield statement during returning of data. In the Python programming language, when the next () function is invoked, the generators are also activated just as shown below;

```
def reverse(data):
for index in range(len(data)-1, -1, -1):
yield data[index]
>>> for char in reverse('golf'):
... print(char)
...
flog
```

In some other times, people get to confuse the generators and the iterators because they also most play the same roles. The best ways to differentiate the two is to check for the functions that are used by the two or the functions that are created in the two.

What are generator expressions?

The expressions of a generator can be coded using different syntax or arguments but maintaining the round brackets instead of the square brackets. The generators expressions are compact in nature and are only used when the generators have been used by the enclosing function. The following are examples;

```
>>> sum(i*i for i in range(10)) # the summation of squares
the output will be :285
>>> xvec = [20, 30, 40]
>>> yvec = [8, 6, 4]
>>> sum(x*y for x,y in zip(xvec, yvec)) # dot product
the output will  be: 260
>>> from math import pi, sin
>>> sine_table = {x: sin(x*pi/180) for x in range(0, 91)}
>>> unique_words = set(word for line in page for word in
line.split())
>>> valedictorian = max((student.gpa, student.name) for student in
graduates)
>>> data = 'golf'
>>> list(data[i] for i in range(len(data)-1, -1, -1))
['f', 'l', 'o', 'g']
```

Inheritance

In the Python programming language, the classes are allowed to inherit the methods, characteristics and attributes of other classes. This feature is very beneficial because the programmer can create very complex classes that bear the attributes of other classes. The class which is inheriting the attributes is referred to as the child class while the one which is being inherited from is called the parent class. The following is the structure of inheritance from a parent class;

```
 class ChildClass(ParentClass):
```

The syntax of the derived class can also be written in the following format;

```
class DerivedClassName(BaseClassName):
<statement-1>
...<
statement-N>
```

For instance let us study the example below and see how things are done while inheriting from a parent class;

```
class Employers(object):
def __init__(self, firstname, workrate,numberof hours):
self.name = name
self.rate = rate
self.hours = hours
staff = Employers(“Wayne”, 20, 8)
supervisor = Employees(“Dwight”, 35, 8)
manager = Employers(“Melinda”, 100, 8)
print(staff.name, staff.rate, staff.hours)
print(supervisor.name, supervisor.rate, supervisor.hours)
print(manager.name, manager.rate, manager.hours)
class Resigned(Employers):
def __init__ (self, Firstname,work rate,numberof hours, status):
self.name = name
self.rate = rate
self.hours = hours
self.status = status
exemp_1 = Resigned(“Dorothy”, 32, 8, “retired”)
exemp_2 = Resigned(“Malcolm”, 48, 8, “resigned”)
print(exemp_1.name, exemp_1.rate, exemp_1.hours,
exemp_1.status)
print(exemp_2.name, exemp_2.rate, exemp_2.hours,
exemp_2.status)
```

The resultant output of the above code will be:

```
Wayne 20    8
Dwight    35   8
Melinda 100   8
Dorothy 32   8   retired
Malcolm   48   8     resigned
>>>
```

Before we summarize on the inheritance, there are two in-built functions that are used with inheritance in order for it to function well;

- Isinstance() function: this function is used to check the type of instance
- Issubclass () function: this function is used to confirm if the

class inheritance has occurred.

In addition, Python allows for multiple inheritances to take place. The following is the syntax that can be used to achieve multiple inheritances;

```
class DerivedClassName(Base1, Base2, Base3):
<statement-1>
...<
statement-N>
```

CHAPTER TEN: GRAPHIC DEVELOPMENT

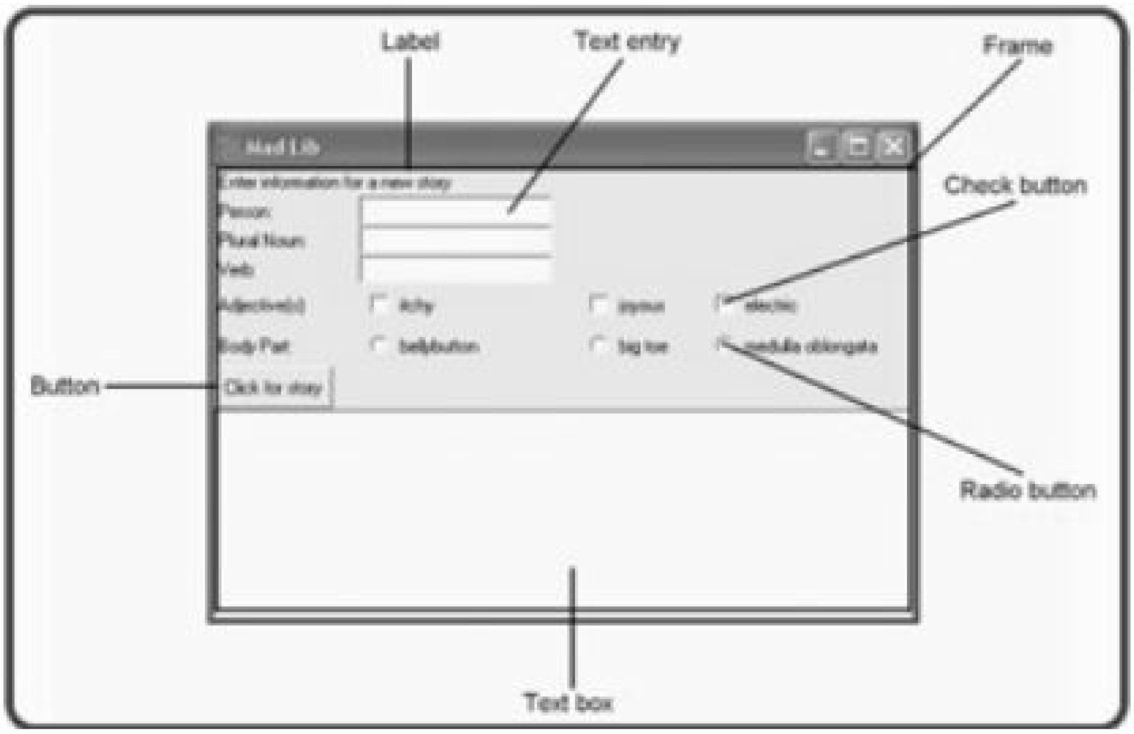
Introduction

From the programs we have created and interacted with, they were based on textual language which was not appealing to the human eye. In this chapter, we are going to interact with sophisticated techniques on how to present and interact with information in the program. The graphical interaction with the computer is referred to as the Graphical User Interface.

The following are the objectives of this chapter;

- Familiarizing with the GUI kits
- Creating and filling frames
- Creating and using buttons in the GUI
- Creations and the use of text entries and boxes
- Creations and the use of checkboxes and radio buttons

The following is an example of the graphical interface and some of the elements have been labeled and named;



We are going to study and learn how to create the above-shown components of the graphical user interface.

Before we get further, we will need to use the GUI TOOL kit for the development of this chapter. We can pick any toolkit from the internet but my preference is the popular Tkinter Toolkit which works best with Python.

The Tkinter toolkit is equipped with a module which we have to instantiate in order to create the GUI components. The following is a table that shows the different GUI components;

Component	Description	The Tkinter class they belong to
Frame	They are used in holding the GUI components	Frame
Button	Used to perform the designated function when it is activated by the user.	Button
Label	Used in the display of texts or icons that cannot be edited	Label
Text entry	It accepts a line that has been entered and therefore displays it.	Entry
Check button	It is used to allow the user to select any option on the display.	Checkbutton
Text box	It accepts multiple lines of texts and hence displays them.	Text
Radio button	Allow the user to select an option from a group of options.	Radiobutton

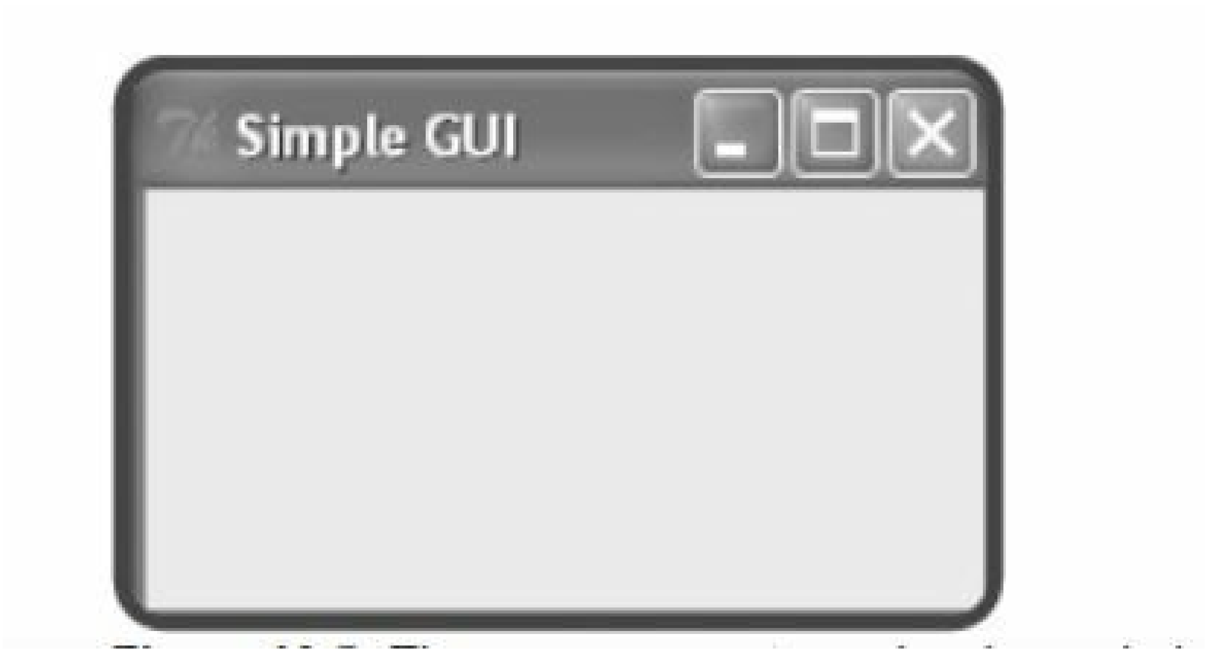
What is Event-Driven Programming?

The programs that are based in GUI can be said to be event-driven. This means that the program quickly responds to the commands which they have been given regardless of the time in which they have been executed. This procedure is very different from coding.

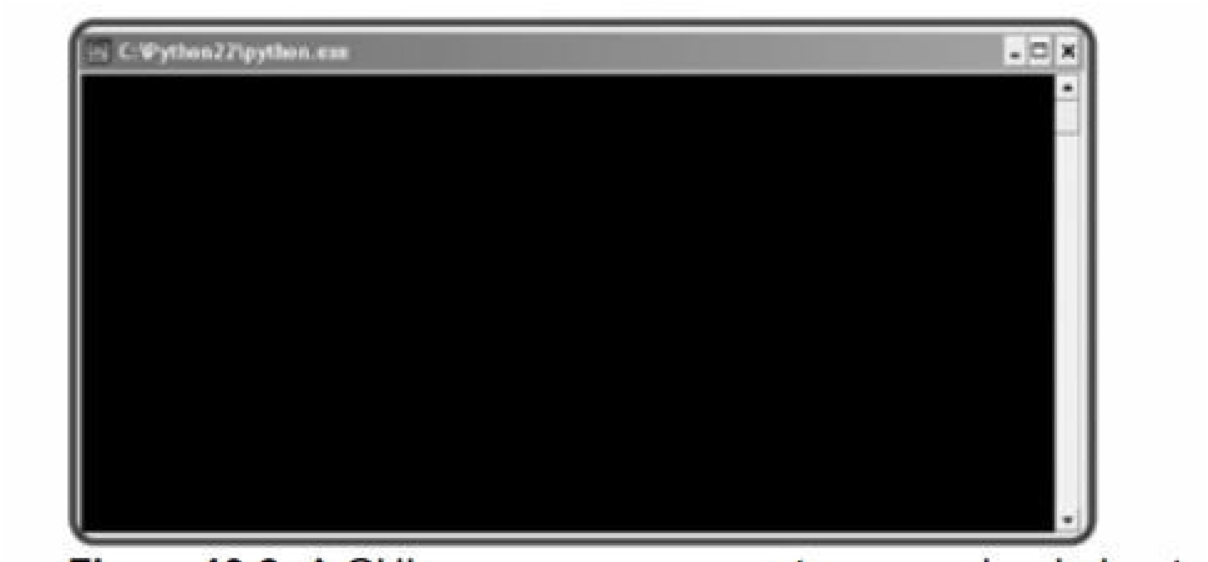
Programs that are written in an event-driven manner can input information in any manner regardless of the order. The program is more user-defined as compared to the real coding process. It involves bounding of event handlers with events so that it can merge the program object to work in an efficient way. This means that the program objects have to be defined together with the events and their handlers. The creation of the event-driven program should be started by the inclusion of an event loop. In this way, the program will have to wait for the described events to occur and hence work on them once they occur.

The Root Window

This can be described as the foundation of the GUI program. This is the platform unto which you add the GUI components during the design of the program. The diagram below shows a simple GUI window;



Depending on the type of operating system that you are using, the GUI windows can be generated in different ways apart from the diagram shown above. The other simple GUI window can be a black console screen as shown below;



Tips: Sometimes you might be running your code and the console window pops up almost every time which creates discomfort. The easiest way in which you can remove this disturbance is to shift your program extension from py to pyw.

How do we import the Tkinter's module

At this point we are going to type in some block of code in order to import the Tkinter module. Let us type in the following block of code;

```
#a very simple GUI
#illustrates the creation of a window
#John Parks-3/3/2019
from Tkinter import*
```

After running the code above, all the Tkinter's modules will be directly imported into the global namespace of the program.

How do we create the root window?

The creation of the root window needs an instantiation of the program object belonging to the Tkinter class as shown below;

```
#Creation of the root window
root =Tk ()
```

From the above example, we can directly access any part of the module without having to type in the name of the module. This is one of the benefits of using the Tkinter tool kit where you do not have to type and it is generally easy to read.

It is important to note that in one Tkinter program you can create only a single root window. When you make a mistake of creating multiple windows then the program becomes frozen and bogus.

In order to modify or edit a root window, we will use the following methods:

```
#modify the window
root.title (“Simplest GUI”)
root.geometry (“300x50”)
```

The title () - this method is used to create or set the heading or the title of the root window. The words in the quotation marks in the round brackets are the exact words that will appear as the title.

The Geometry () – this is the size of the window. The methods understand the integers as strings which are the width and height of the root window.

After gathering all those information about the Root window, we will have to create the event loop. We will have to call the root's mainloop ()

method as shown below;

```
#establishing the root window's event loop.
root.Mainloop().
```

When we run the method shown above, the root window will be activated to open and wait for events to occur so that they can be handled. The window will carry out the defined actions that have been specified in the program. In addition, the window can be enlarged, closed and even minimized depending on your visual needs.

The Labels

From the previous table in the chapter, we have already known about the function of the labels in GUI programs. This is one of the simplest GUI components. Before we continue, the GUI components can also be referred to as the Widgets. The widgets name was derived from two names which are windows and gadgets. Therefore we can say that the label is used to indicate other widgets in the GUI window. The user is not able to change the label during interaction with the GUI window.

The following is an example of a program;

```
#creating a label
#Illustration of how to create a label
#John Park -3/3/2019

From Tkinter import*

#creation of a window
root = Tk()
root. Title ("show a root labeler")
root. Geometry ("300x100")
```

How do we create the frame?

The frame is the widget that encloses all the other widgets. It acts as the fence of the GUI window. In order to create a new frame we will type in the following piece of code;

```
# Creation of a frame that will carry other GUI widgets
app = Frame (root)
```

We have to invoke the grid () function so that it can pass it through the master constructor of any object that is new as shown below;

```
App.grid()
```

You should take caution so that you cannot get to confuse between a method and a function. In the above example we have used the method to arrange the widgets on the window.

How do we create the label?

In order to create a label, we need to instantiate the object of the class containing the label (Label Class);

```
#creation of the label inside a frame
lbl = Label (app, text = “This is a label”)
```

As a result of running the above code, the label is created inside the frame. The options that are available in setting up a widget are mainly for the appearance of the widget in the frame. The next appropriate action is to call the grid method as shown below;

```
Lbl.grid()
```

The method above ensures that the label that has been created is very visible.

The last step is to initiate the event loop. You are required to call the event loop in order to kick off the showing of the label;

```
# Starting the root windows’ event loop
root.mainloop()
```

The Buttons

This is the type of widget that needs to be activated in order to start functioning. The concept of creating the buttons resembles the procedure of creating the labels. Therefore, this section will be easy.

The procedure is easy; we start by instantiating the object of the class containing the button (Button class) as shown below;

```
# Creation of a button inside the frame
bttn1= Button (app, text = “This is a button”)
bttn1.grid ()
```

When the codes above are run, they create a new button having the initials “this is a button”. You can follow the same procedure and create more and more buttons as shown above with different initials. The other important step is the configuration of the button as shown below;

```
Bttn2.configure (text = “another button”)
```

After following the whole procedure then you can enter the event loop just as you did with the labels and frames.

Implementation of the GUI through a Class

In Python programming language, the more you organize your block of codes into classes, the more organized and easier it becomes for you. This is also advantageous in the creation of bigger GUI programs and codes. Therefore, in this section we are going to learn how to organize our GUI codes into classes.

We will have to create a new class in order to define the "application" class. Let us observe the example below;

```
Class Application (Frame):
""""A GUI application that has two buttons"""".
```

The next step if the definition of the "Constructor" method as shown below;

```
Def_init_ (self, master):
""""Initialization of the frame""""
Frame ._init_ (self, master)
self.grid ()
self.create_widgets ()
```

The next step is the definition of a method that will create the other widgets as shown below. (the other widgets are the two buttons that were mentioned in the program);

```
Def create_widgets (self):
""""Creation of two functionless buttons""""
#the creation of the first button
self.bttn1 = Button (self, text = "This is the first button with no
function")
self.bttn1.grid ()

#Creation of the second functionless button
self.bttn2 = Button (self)
self.bttn2.grid ()
self.bttn2.configure (text = "This is the second functionless button")
```

The piece of coding above looks similar to you because we have gone through the same procedure in the other widgets. The last step is the creation of the "application" object as shown below;

```
Root = Tk ()
root.title ("Buttons Maniac"
root.geometry ("300x 75")
```

After the code above, we will instantiate the object as follows;

```
App = Application (root)
```

Finally you must invoke or call the event loop as shown below;

```
Root.Mainloop()
```

In summary, we have come to the end of our chapter and kindly attempt the challenges shown below;

Exercise

1. Create a GUI program that has all the widgets that have been discussed in this chapter.
2. Create a GUI program that can be used in a restaurant.

CHAPTER ELEVEN: STANDARD LIBRARY AND VIRTUAL ENVIRONMENT

Introduction

In this chapter, we are going to take see what is inside the standard library of the Python programming language. I urge you to familiarize yourself with the different components of the library because you shall be using them someday.

The OS Interface Model

In this area, the operating system gives us the platform of interaction with many functions running inside the OS as shown below;

```
>>> import os
>>> os.getcwd() # Return the current working directory
'C:\\Python36'
>>> os.chdir('/server/accesslogs') # Change current working
directory
>>> os.system('mkdir today') # Run the command mkdir in the
system shell
0
```

The import is used in the OS as os.open () which performs various functions too. The in-built function open () is kept from shadowing by the os.open (). On the other end, the in-built dir () work together with the help () function for interactive purposes of big modules;

```
>>> import os
>>> dir(os)
<returns a list of all module functions>
>>> help(os)
<returns an extensive manual page created from the module's
docstrings>
```

A higher interface level is often used in situations where the shutil module handles the file and directory tasks in terms of management;

```
>>> import shutil
>>> shutil.copyfile('data.db', 'archive.db')
'archive.db'
>>> shutil.move('/build/executables', 'installdir')
'installdir'
```

The Wildcards (file)

In most cases, wildcards are used for searching purposes. As a result, the python programming language offers the file wildcards from the glob module which returns the lists from a directory search as shown below;

```
>>> import glob
>>> glob.glob('*.py')
['primes.py', 'random.py', 'quote.py']
```

The Commanding arguments (Line)

These are the type of line arguments that are usually stored in the system module as a list. They can be described as common utility scripts that process the command arguments. The following is an example;

```
>>> import sys
>>> print(sys.argv)
['demo.py', 'one', 'two', 'three']
```

In summary, there are other powerful commanding line arguments such as the argparse module. They are also known to be very flexible

The Error Output Redirector and Program Terminator

The sys module contains various attributes such as the stderr, stdin and stdout. They are very useful in terms of signaling of warnings and the error messages most especially during redirection of stdout as shown below;

```
>>> sys.stderr.write('Warning, log file not found starting a new
one\n')
Warning, log file not found starting a new one
```

In summary, the most efficient and effective way of ending a script is usually the use of sys.exit ().

Marching of String Patterns

The “re” module always contains some arguments or expressions that are very advanced in processing of strings. They can also be used for complex processing, matching and the manipulation of regular arguments. All these are done for the purpose of optimizing the solutions as shown below;

```
>>> import re
>>> re.findall(r'\bf[a-z]*', 'which foot or hand fell fastest')
['foot', 'fell', 'fastest']
>>> re.sub(r'(\b[a-z]+) \1', r'\1', 'cat in the hat')
'cat in the hat'
```

On the other end, some programmers prefer the string methods because they are simple to read, analyze and debug;

```
>>> 'tea for too'.replace('too', 'two')
'tea for two
```

Performing Mathematical Calculations

The module that is used in mathematics allows the C library functions to utilize the float pointing mathematical calculations;

```
>>> import math
>>> math.cos(math.pi / 4)
0.70710678118654757
>>> math.log(1024, 2)
10.0
```

The random module is used to provide criteria for random selection mathematics as shown below;

```
>>> import random
>>> random.choice(['apple', 'pear', 'banana'])
'apple'
>>> random.sample(range(100), 10) # sampling without replacement
[30, 83, 16, 4, 8, 81, 41, 50, 18, 33]
>>> random.random() # random float
0.17970987693706186
>>> random.randrange(6) # random integer chosen from range(6)
4
```

In addition, the statistics module provides the basic tools for statistical calculations such as variance, mean, mode and median;

```
>>> import statistics
>>> data = [2.75, 1.75, 1.25, 0.25, 0.5, 1.25, 3.5]
>>> statistics.mean(data)
1.6071428571428572
>>> statistics.median(data)
1.25
>>> statistics.variance(data)
1.3720238095238095
```

The Internet Access

In this section, there are several modules that can be used in accessing the internet as well as manipulating the internet protocol. Let us see the example below;

```
>>> from urllib.request import urlopen
>>> with urlopen('http://tycho.usno.navy.mil/cgi-bin/timer.pl') as
```

```
response:
... for line in response:
... line = line.decode('utf-8') # Decoding the binary data to text.
... if 'EST' in line or 'EDT' in line: # look for Eastern Time
... print(line)
<BR>Nov. 25, 09:43:32 PM EST
>>> import smtplib
>>> server = smtplib.SMTP('localhost')
>>> server.sendmail('soothsayer@example.org',
'jcaesar@example.org',
... """To: jcaesar@example.org
... From: soothsayer@example.org
...
... Beware the Ides of March.
... """)
>>> server.quit()
```

The simplest of the modules are the urllib.request for manipulating the URL and the smtplib for sending the emails.

The Dates and The Times

The modules involved here work both in simple and complex ways depending on the classes that they belong to. The modules work partly as arithmetic and support the objects that are time zone based.

```
>>> #The dates can easily be constructed, modified and formatted
>>> from datetime import date
>>> now = date.today()
>>> now
datetime.date(2013, 02,2 2)
>>> now.strftime("%m-%d-%y. %d %b %Y is a %A on the %d day
of %B.")
'12-02-03. 22 Feb 2013 is a Wednesday on the 03 day of February.'
>>> # dates support calendar arithmetic
>>> birthday = date(1963, 8, 3)
>>> age = now - birthday
>>> age.days
14365
```

The Data Compression

These modules are mostly used for data archiving purposes and formats that compress such as bz2, tarfile, lzma, zipfile, zlib and gzip;

```
>>> import zlib
>>> s = b'witch which has which witches wrist watch'
>>> len(s)
41
>>> t = zlib.compress(s)
>>> len(t)
37
>>> zlib.decompress(t)
b'witch which has which witches wrist watch'
>>> zlib.crc32(s)
226805979
```

The Quality Control Modules

The best measures to take while developing high quality and accurate software is to write and conduct tests for each and every function that has been developed in that program. You are also supposed to run those test frequently during the development stages until you finish the coding part.

The modules such as the doctest are very good at analyzing the Docstring. This measure improves the quality of the software program as well as the documentation part of the running program;

```
def average(values):
"""Performs the Computation of the arithmetic mean of a list of
numbers.
>>> print(average([30, 40, 80]))
40.0
"""
return sum(values) / len(values)
import doctest
doctest.testmod() #This will automatically achieve validation the
embedded tests
```

The other module suitable for quality control is the unittest module. This is a more comprehensive test that does it separately and keeps a copy of the test;

```
import unittest
class TestStatisticalFunctions(unittest.TestCase):
def test_average(self):
self.assertEqual(average([30, 40, 80]), 50.0)
self.assertEqual(round(average([2, 6, 8]), 1), 4.3)
with self.assertRaises(ZeroDivisionError):
```

```
average([])
with self.assertRaises(TypeError):
average(30, 40, 80)
unittest.main() # Calling from the command line invokes all tests
```

The Binary data record Layout

In this section, the struct module is used to provide the pack () and the unpack() functions that work with the variable length binary formats. The following is a good example that shows the ZIP file;

```
import struct
with open('The file.zip', 'rb') as f:
data = f.read()
start = 0
for i in range(3): # This will be showing the first three file headers
start += 15
fields = struct.unpack('<IIIHH', data[start:start+17])
crc32, comp_size, uncomp_size, filenamesize, extra_size = fields
start += 17
filename = data[start:start+filenamesize]
start += filenamesize
extra = data[start:start+extra_size]
print(filename, hex(crc32), comp_size, uncomp_size)
start += extra_size + comp_size # This will be skipping to the next
header
```

Assignments

1. Research and find out about multithreading and hence analyze the following code;

```
import threading, zipfile
class XsyncZip(threading.Thread):
def __init__(self, infile, outfile):
threading.Thread.__init__(self)
self.infile = infile
self.outfile = outfile
def run(self):
f = zipfile.ZipFile(self.outfile, 'w', zipfile.ZIP_DEFLATED)
f.write(self.infile)
f.close()
print('Finished background zip of:', self.infile)
```

```
background = AsyncZip('mydata.txt', 'myarchive.zip')
background.start()
print('The main program continues to run in the foreground.')
background.join() # Waiting for the background task to terminate
print('Main program waited until background was done.')
```

2. Research and find about the weak references in python programming language and analyze the code below;

```
>>> import weakref, gc
>>> class X:
... def __init__(self, value):
... self.value = value
... def __repr__(self):
... return str(self.value)
...
>>> x = X(10) # create a reference
>>> d = weakref.WeakValueDictionary()
>>> d['primary'] = a #THIS WILL not create a reference
>>> d['primary'] # fetching of the object is done if it is still alive
10
>>> del x # removal of the one reference
>>> gc.collect() # running of garbage collection right away
0
>>> d['primary'] # the entry was initially automatically removed
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
d['primary'] # entry was automatically removed
File "C:/python36/lib/weakref.py", line 46, in __getitem__
o = self.data[key]()
KeyError: 'primary'
```

THE VIRTUAL ENVIRONMENT

Introduction

The creation of the virtual environment is essential for situations where the python programming language wants to outsource some help in solving a problem.

A virtual environment can be described as a self-contained platform that contains a Python installation for any version of the python as well as a good number of additional packages. It is important to note that different environments are used for different applications.

Coming up with a Virtual Environment

The module that is involved in the creation and management of the virtual environment is known as “venv”. This module is responsible for the installation of the most updated version of the available python. When creating a virtual environment you are supposed to select an appropriate directory and version of python. After a proper selection, you are supposed to run the venv module as shown below;

```
python3 -m venv tutorial-env
```

Once the virtual environment is created, you will be required to activate it on your operating system. After the activation is done the python shell prompt will change and update itself as shown below;

```
$ source ~/envs/tutorial-env/bin/activate
(tutorial-env) $ python
Python 3.5.1 (default, June 6 2019, 11:59:38)
...
>>> import sys
>>> sys.path
['', '/usr/local/lib/python35.zip', ...,
'~/envs/tutorial-env/lib/python3.5/site-packages']
>>>
```

CHAPTER TWELVE: CONCLUSION AND ACKNOWLEDGEMENT

Congratulations on finishing this book, I hope it was able to equip you with the essential skills and fundamental knowledge to explore and harness the powerful features of Python as a programming language. By the time you finished reading the book, I am confident that you will be prepared to put your basic programming knowledge to practical everyday uses.
The next step is to take up advanced Python programming courses that will help you create more complex programs such as games, web applications, and productivity tools. Finally, if you enjoyed this book, please take the time to share your thoughts and post a positive review on Amazon. It'd be greatly appreciated! Thank you and good luck!

Working in Python can be one of the best programming languages for you to choose. It is simple to use for even the beginner, but it has the right power behind it to make it a great programming language even if you are more of an advanced programmer in the process. There are just so many things that you can do with the Python program, and since you are able to mix it in with some of the other programming languages, there is almost nothing that you can't do with Python on your side. It is not a problem if you are really limited on what you are able to do when using a
programming language. Python is a great way for you to use in order to get familiar and to do some really amazing things without having to get scared at how all the code will look. For some people, half the fear of using a programming language is the fact that it is hard to look at with all the brackets and the other issues. But this is not an issue when it comes to using Python because the language has been cleaned up to help everyone read and look at it together. This guidebook is going to have all the tools that you need to hit the more advanced parts of Python. Whether you are looking at this book because you have a bit of experience using Python and you want to do a few things that are more advanced, or you are starting out as a beginner, you are sure to find the answers that you need in no time. So look through this guidebook and find out everything that you need to know to get some great codes while using the Python programming.

Lastly, I would like to urge you to continue practicing the exercises on the book as well as working out the assignments that have been given in the book.

What next after this book?

You might be stranded on the next step to take after exhausting the study of this book. I want to assure you that you have the best foundation of software engineering hence you are good to develop yourself further. I would also like to prescribe to you the following counsels;

- **Pick a programming language with a corresponding framework or platform.** A good example of such is the one we have studies in the book of C# and Dot Net platform. Other platforms include; Java language with Java EE, PHP with CakePHP, Ruby with Rails and many others. do not feel tied to the C# programming language; you can also explore other programming languages.
- **Find resources and books about databases** . This will enable you to create complex applications using tables and relations. Learn about the queries, Oracle, MySQL and ORM technologies.