A

Technical report on

# PYTHON AND ITS APPLICATIONS

In partial fulfillment of the

**SCA mentorship program**

By

**OMARI SALIMA ONYEBUCHI**

## SUMMARY

This report contains what was learnt during the SCA mentorship program which was python. It also contains the significance of python as a programming language  and describes some tasks that were solved using python and a project that was carry out using python as a major tool. Some essential features of the Python programming language were also discussed.

# CHAPTER ONE

## 1.0 INTRODUCTION

**Python** is a widely used general purpose, high-level programming language which was initiated by Guido van Rossum in the year 1989. The language facilitates a rapid development cycle with its interpreted and interactive nature. An emphasis on code readability makes it easy to learn the language, and the communicative syntax leads to short, clear programs. Python compiles to an intermediary code and this in turn is interpreted by the Python Runtime Environment to the Native Machine Code. Python can be applies in various fields such as web development, game development, scientific and numeric applications, artificial intelligence and machine learning, software development among others.

### 1.01 EDITORS

They are used by programmer to create and modify (edit) programming language source code. Programs are written in the form of human readable text, and then processed by other programs to make them perform the tasks dictated by the programmer. According to the language being used to code, the code editor highlights special keywords, give suggestions for some extent, automatic indentation features and sometimes it has an integrated terminal as well. Examples of some editors are Sublime Text, Visual Studio Code.

### 1.0.2 SOURCE FILE

In many computer languages, code is written into a text file, and then compiled into a form that the computer can execute. The file that contains the text, in human-readable form (or at least semi-human-readable form) is called the source file.

### 1.0.3. COMPILER

A compiler is a software program that transforms high-level source code that is written in a high-level programming language into a low level object code (binary code) in machine language, which can be understood by the processor and the process of converting high-level programming into machine language is known as **compilation.**

### 1.0.4 VIRTUAL MACHINE

A virtual machine is a hypothetical device that is capable of executing a specific instruction set (program). Such an instruction set is commonly called ByteCode. A common implementation strategy for programming languages is to compile code into instructions for a specialized Virtual Machine.

### 1.0.5 LIBRARY MODULES

A library is a collection of non-volatile resources used by computer programs, often for software development. A library is also a collection of implementations of behavior, written in terms of a language, that has a well-defined interface by which the behavior is invoked. For instance, people who want to write a higher level program can use a library to make system calls instead of implementing those system calls over and over again.

### 1.0.6 PYTHON INTERPRETER

The Python interpreter is a virtual machine, meaning that it is software that emulates a physical computer. This particular virtual machine is a stack machine: it manipulates several stacks to perform its operations. The Python interpreter is a bytecode interpreter*:* its input is instruction sets called bytecode.

# CHAPTER TWO

## 2.0 KNOWLEDGE GAINED DURING SCA MENTORSHIP

### 2.0.1 PROGRAMMING PARADIGMS

A programming paradigm is a style, or way of programming. Some languages are easier to write in some paradigms but others are not. A program can contain more than a paradigm so it is not mutually exclusive. There are a number of paradigms used in programming such as;

- **Imperative programming** paradigm executes command in a step-by-step manner in a way that you sort of give some verbal commands. It normally give an approach on how to solve it and it makes a direct change on the state on the program.
- **Procedural Programming** paradigm divides the program into procedures, which are also known as routines or functions, simply containing a series of steps to be carried out. A predefined function, which is usually a major feature of this method of programming, is typically an instruction identified by a name.
- **Functional programming paradigm** is the process of programming by composing pure **function** (which returns same values for the same inputs and also does not change the attributes of the program outside the function). Functional code tends to be more concise and easier to test than imperative or object oriented programming.
- **Object Oriented Programming (OOP)** paradigm relies on the concept of classes and objects. It is used to structure a software program into simple, reusable pieces of code blueprints (usually called classes) which are used to create individual instances of objects. OOP usually contain classes, objects, attributes.
- **Declarative programming** paradigm basically involves instructing a program on *what* needs to be done, instead of telling it *how* to do it or showing how it is being done. They always describe the desired end result rather than outlining all the intermediate work steps.
- **Structured programming uses** flowcharting as a method of documenting the paths that a program would execute. There are three main categories in this control structure which are; **sequence** (which follows one instruction then the next and the next.), **selection** (choosing between two or more paths and this choice is normally influenced by a question that determines which path to follow) and **iteration** (which is known as repetition or looping and it allows a code to run for several times depending on how many times it ought to).

### 2.0.2 LEARNING PYTHON AS A PROGRAMMING LANGUAGE

Python being an expressive and interactive programming language, it has some basic features that must be understood in order to use the language to carry out operations. Some of these features will be discussed below.

- **For loops:** One great thing about computers is that they can repeat things over and over very fast and there are some ways to repeat things in Python, the most common method is the use of **for loop**.

  Example: A user is asked for number three times and the program below prints the squares of the numbers using a for loop.

```
for i in range(3):
    num = eval(input('Enter a number: '))
    print ('The square of your number is', num*num)
print('The loop is now done.')


C:\Users\salima omari> c: && cd "c:\Users\salima omari" && cmd /C "python
"c:\Users\salima omari\.vscode\extensions\ms-python.python-
2020.12.424452561\pythonFiles\lib\python\debugpy\launcher" 55310 --
"c:\Users\salima omari\forloops.py" "
Enter a number: 3
The square of your number is 9
Enter a number: 6
The square of your number is 36
Enter a number: 7
The square of your number is 49
The loop is now done
```

- **Conditional statement:** Sometimes in programs we only want to do something provided a condition is met. Python’s conditional statement which are if, el-if and else, is what we need.
  Example: Below is a game called ‘rock’,’paper’,’scissors’ and it was programmed using the condition statements.

```
•  from random import randint
•  t=['rock','paper','scissors']
•  for i in range(5):
•      computer=t[randint(0,2)]
•      player=False
•      count1=0
•      count2=0
•      while player == False:
•
•              player=input('"rock","paper" or "scissors"\n>>>')
•
•              if player==computer:
•                  print('tie')
•              elif player=='rock':
```

```
                if computer=='paper':
                    print(f'you lose {computer} covers {player}')
                    count2+=1
                else:
                    print(f'right, {player} smashes {computer}')
                    count1+=1
            elif player=='paper':
                if computer=='scissors':
                    print(f'lost!, {computer} cuts {player}')
                    count2+=1
                else:
                    print(f'win! {player} covers {computer}')
                    count1+=1
            elif player=='scissors':
                if computer=='rock':
                    print(f'lost!,{computer} smashes {player}')
                    count2+=1
                else:
                    print(f'win!, {player} cuts {computer}')
                    count1+=1
player=False
computer=t[randint(0,2)]
if count1>count2:
    print('congrats, you won!!')
else:
    print('sorry loser')
```

- **Lists:** In a situation where we need to get a number of ages or scores from a user and then carry out some operations on the data like arrange them in a particular order, we could create variables for each of them and then arrange then accordingly but then would be tedious and that's where lists comes in.

  **L = [1,2,3]**….Here is how a simple list is formed.
  We could also find the position of any element of the list as shown below;

```
L = eval(input('Enter a list: '))
print('The first element is ', L[0])


C:\Users\salima omari> c: && cd "c:\Users\salima omari" && cmd /C "python
"c:\Users\salima omari\.vscode\extensions\ms-python.python-
2020.12.424452561\pythonFiles\lib\python\debugpy\launcher" 55314 --
"c:\Users\salima omari\lists.py" "
Enter a list: [1,2,3]
```

```
The first element is  1
```

Example: Lists as well as conditional statements were used to create a short question-answer thread as shown below

```
questions = ['What is the capital of France?', 'Which state has only one neighbor
?']
answers = ['Paris','Maine']
num_right = 0
for i in range(len(questions)):
    guess = input(questions[i])
    if guess.lower()==answers[i].lower():
        print('Correct')
        num_right=num_right+1
    else:
        print('Wrong. The answer is', answers[i])
    print('You have', num_right, 'out of', i+1, 'right.')


C:\Users\salima omari> c: && cd "c:\Users\salima omari" && cmd /C "python
"c:\Users\salima omari\.vscode\extensions\ms-python.python-
2020.12.424452561\pythonFiles\lib\python\debugpy\launcher" 55318 --
"c:\Users\salima omari\conditionals.py" "
What is the capital of France?Paris
Correct
You have 1 out of 1 right.
Which state has only one neighbor?Maine
Correct
You have 2 out of 2 right.
```

- **A dictionary** is a more overall version of a list. Here is a list that contains the number of days in the months of the year: days = [31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31], however we can assign these days to their specific months using dictionaries.
  Here is a dictionary of the days in the months of the year:

```
days = {'January':31, 'February':28, 'March':31, 'April':30, 'May':31,
    'June':30, 'July':31, 'August':31, 'September':30, 'October':31,
    'November':30, 'December':31}
```

To get the number of days in January ,we use days['January']. One major advantage of dictionary is that the code is more readable, and we don’t have to figure out which index in the list a given month is at. Operations can also be carried using the values of the keys in a dictionary as shown below.

```
d = {'A':100, 'B':200,'C':300}

for key in d:
    print(d[key]**2)

C:\Users\salima omari>python "c:/Users/salima omari/dic.py"
10000
40000
90000
```

**Functions** are useful for breaking up a huge program to make it simpler to read and maintain. They can also be used in a case where there is a repetition of a particular set of codes in your program. You can put those codes in a function and call the function whenever you want to execute that code.

From the program below creating the function makes it easy to print as many ‘Hello’s as possible instead of going through the for loop method

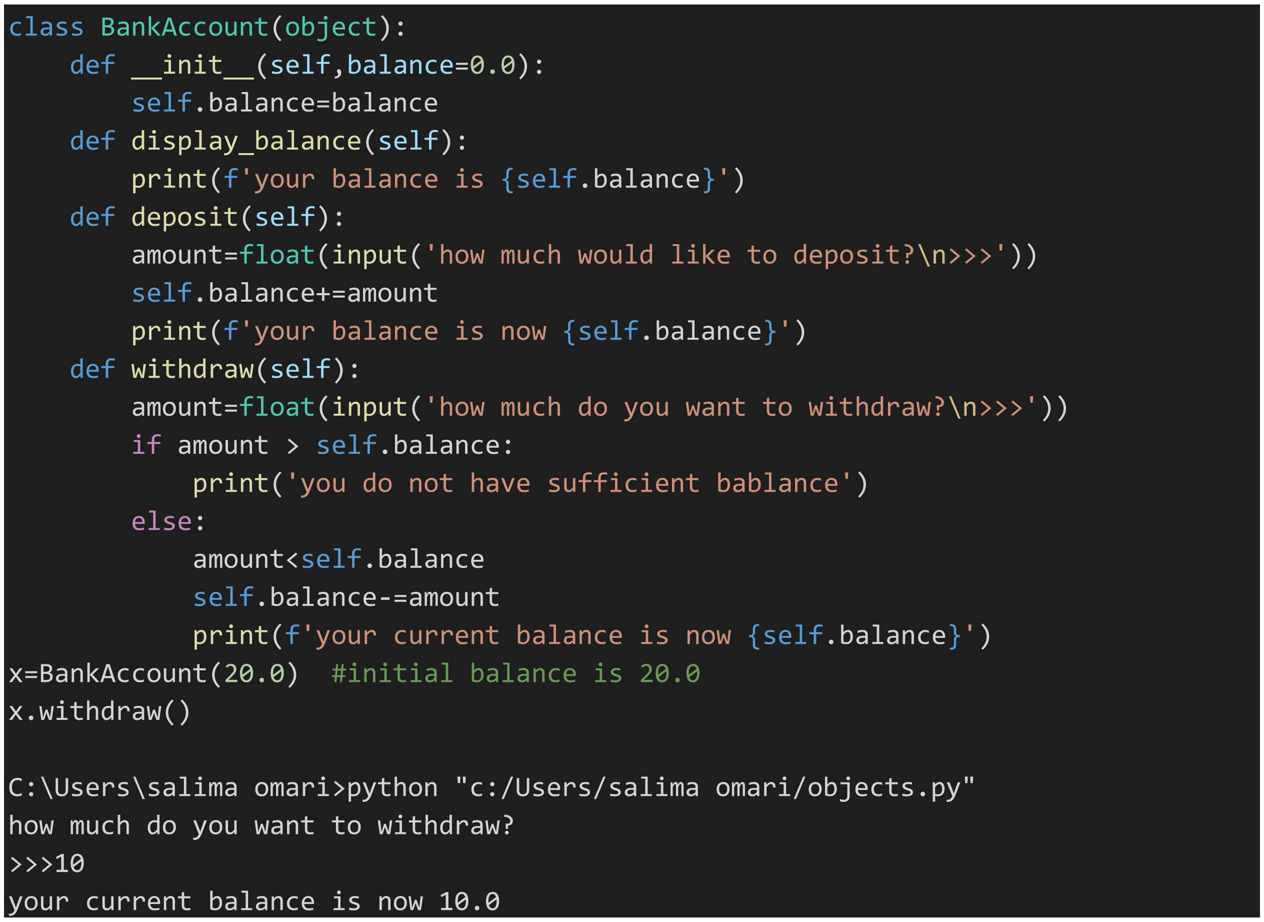
```
def print_hello(n):
    print('Hello ' * n)
    print()

print_hello(3)

C:\Users\salima omari>python "c:/Users/salima omari/functions.py"
Hello Hello Hello
```

Example: A dictionary in a json format was imported and a function was used to find the meanings of a particular word.

```
import json
y=json.load(open("data.json"))
m=input('enter the word you want\n>>>')
from difflib import get_close_matches

def tran(m):
    m=m.lower()
    if m in y:
        return y[m]
    elif m.title() in y:
        return y[m.title()]
    elif m.upper() in y:
        return y[m.upper()]
    elif len(get_close_matches(m,y.keys()))>0:
        print(f"do you mean {get_close_matches(m,y.keys())[0]}")
        u=input("y or n\n>>>")
        if u=="y":
            return y[get_close_matches(m,y.keys())[0]]
        elif u=='n':
            print('not available')
        else:
            print('wrong input')

v=tran(m)
if type(v)==list:
    if len(v)>1:
        print(f'the word could mean about {len(v)} things')
        for i in range(len(v)):
            print(f'{i+1}. {v[i]}')
    else:
        print(v[0])


C:\Users\salima omari>python "c:/Users/salima omari/dictionaryy.py"
enter the word you want
>>>fall
the word could mean about 12 things
1. Move to a lower position due to the effect of gravity.
2. The act of surrendering to the enemy.
3. To die in battle.
4. A downward slope or bend.
5. To go from a higher to a lower place.
6. To lose one's balance and hit the ground.
7. A sudden drop from an upright position.
8. A wrestling move in which a wrestler's shoulders are forced to the mat.
9. To move downward and lower (e.g. of temperature values or falling objects).
10. To pass suddenly and passively into a state of body or mind (e.g. into a
trap, ill, in love, etc.).
11. To come under, be classified or included (e.g. into a category).
12. To touch or seem as if touching visually or audibly.
```

**Object oriented programming:** It relies on the concept of classes and objects. It is used to structure a software program into simple, reusable pieces of code blueprints (usually called classes) which are used to create individual instances of objects. OOP usually contain classes, objects, attributes.

A typical example is shown below in which an object (the bank account) is created with some features such as withdrawal, deposit and account balance.



```
class BankAccount(object):
    def __init__(self,balance=0.0):
        self.balance=balance
    def display_balance(self):
        print(f'your balance is {self.balance}')
    def deposit(self):
        amount=float(input('how much would like to deposit?\n>>>'))
        self.balance+=amount
        print(f'your balance is now {self.balance}')
    def withdraw(self):
        amount=float(input('how much do you want to withdraw?\n>>>'))
        if amount > self.balance:
            print('you do not have sufficient bablance')
        else:
            amount<self.balance
            self.balance-=amount
            print(f'your current balance is now {self.balance}')
x=BankAccount(20.0)  #initial balance is 20.0
x.withdraw()

C:\Users\salima omari>python "c:/Users/salima omari/objects.py"
how much do you want to withdraw?
>>>10
your current balance is now 10.0
```

### 2.0.3 WEB DEVELOPMENT

Python programming language was applied in the development of a website called PaMunda. This project was made to bridge a gap between farmers and consumer. A python webframework was used in the construction of the project. Here, is the link to the code on github; https://github.com/salimah-web/SCA-final-project.

## CONCLUSION

A lot of knowledge on python and programming as a whole was gained during the three-months program and those knowledge were also applied in solving various problems or tasks. Python is a great comprehensive language with many applications which some of them were shown in the previous contents.