Chapter 3: Python and Control Structures

3.1 Introduction

In most of the programs presented above, the statements have been sequential, executed in the order they appear in the main program. Control Statements are meant to control the logic of program's execution i.e., the order in which the instructions in a program must be executed. They make it possible to make decisions, to perform tasks repeatedly or to jump from one section of code to another.

Programmers may need to test values of variables and depending on the result, alternative statements may be executed. This facility can be used to select among alternative courses of action. It can also be used to build loops for the repetition of basic actions.

This chapter therefore aims at familiarizing the learners to python control structures. The chapter also aims at providing insights on which control structure can be used to achieve a given task appropriately.

3.2 Control structures in python

There are three broad categories of control structures supported by python

i). **Sequential control structures** whereby python statements are executed in the order in which they appear in the program

ii). **Decision control structures** which use a Boolean expression to determine if statements shall be executed or not include control structures

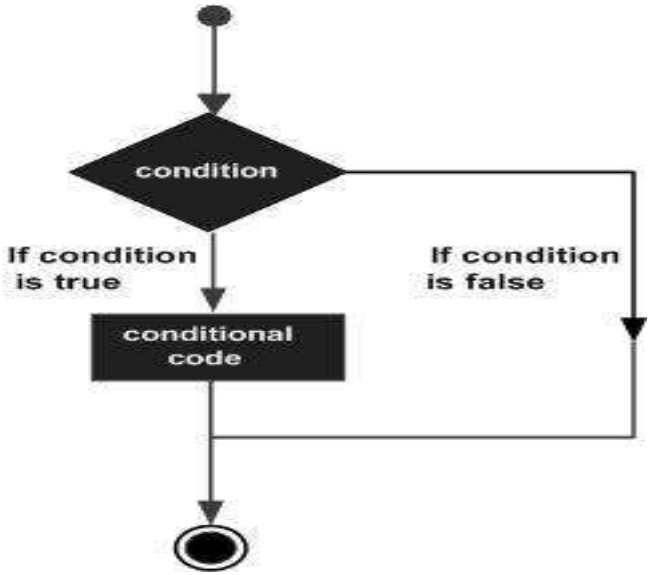
iii). **Iterative/Looping/Repetitive control structures** which enables a section of a program to be executed repeatedly until a given condition has been met or for a specified number of times. Decision control structures

3.3 Decision Control Structures

Decision making is anticipation of conditions occurring while execution of the program and specifying actions taken according to the conditions.

Decision structures evaluate multiple expressions which produce TRUE or FALSE as outcome. You need to determine which action to take and which statements to execute if outcome is TRUE or FALSE otherwise.

Following is the general form of a typical decision-making structure found in most of the programming languages –



Python programming language supports the following types decision control structures.

3.3.1. If statement

An if statement consists of a Boolean expression followed by one or more statements.

```
x = 14
y = 20
if x > y:
        print("x is greater than y")
```

In this example we use two variables, x and y, which are used as part of the if statement to test whether x is greater than y. As x is 14, and y is 20, we know that 14 is not greater than 20, therefore the program will not we print on the screen "x is greater than y".

Indentation

Python relies on indentation (whitespace at the beginning of a line) to define scope in the code unlike other programming languages which often use curly-brackets for this purpose.

If the statement is written without indentation, the program will raise an error

```
x = 14
y = 20
if x > y:
    print("x is greater than y")
```

3.3.2. Elif

The elif keyword is pythons' way of saying "*if the previous conditions were not true, then try this condition*".

Example

```
x= 14
y = 20
if x > y:
  print ("x  is greater than y")
elif y>x b:
        print ("y is greater than x")
elif y>==x:
        print ("y is equal to x")
```

In this example a is equal to b, so the first condition is not true, but the elif condition is true, so we print to screen that "a and b are equal".

3.3.3. if...else statement

An if statement can be followed by an optional else statement, which executes when the Boolean expression is **FALSE.** The else keyword catches anything which isn't caught by the preceding conditions. An example is presented below

Example

```
x= 14
y= 20
if x> y:
        print("x is greater than y")
elif x == y:
         print("x and y are equal")
else:
         print("y  is greater than x")
```

3.3.4. The pass Statement

If statements cannot be empty, but if you for some reason have an if statement with no content, put in the pass statement to avoid getting an error. An example on how to use pass is shown below

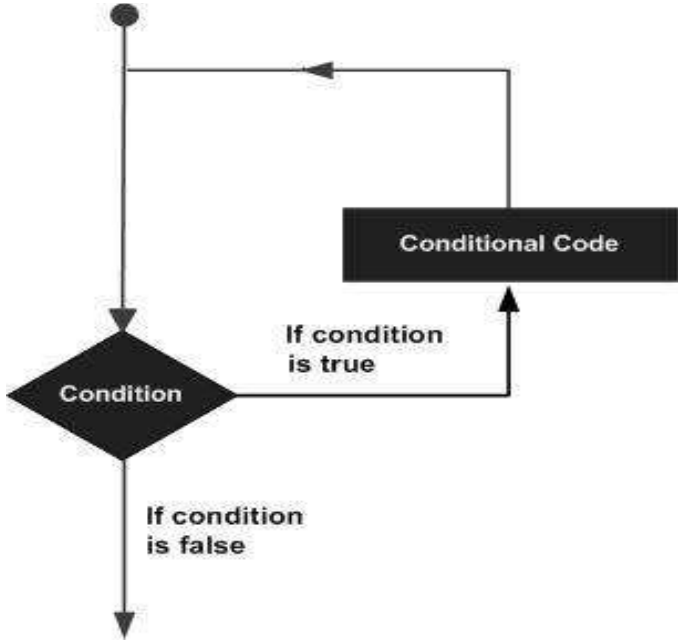
Example

```
x = 14
y= 20
if x > y:
        pass
```

3.4 Python and loops

There may be a situation when you need to execute a block of code several number of times. Programming languages provide various control structures that allow for more complicated execution paths.

A loop statement allows us to execute a statement or group of statements multiple times. The following diagram illustrates a loop statement –



Python programming language provides two loops to programmers; while and for loops types of loops to handle looping requirements.

3.4.1. The while Loop

With the while loop, we can execute a set of statements as long as a condition is true. The loop repeats a statement/ group of statements while a given condition is TRUE.

The condition is first evaluated before executing the loop body e.g.

Write a program for printing even numbers between 1 and 100 using a while loop

```
x = 1
while x < 100:
        if x%2==0
                print(x)
x ++
```

Note: remember to increment x, or else the loop will continue forever.

The while loop requires relevant loop variables to be initialized to the starting value. In this example we need to define an indexing variable, x, which we set to 1.

The break Statement

We use the break statement to stop the loop even if the while condition is true:

```
j = 1
while j < 100:
 print(j)
 if j == 50:
   break
 j += 1
```

The above program terminates when the value of j becomes 50 even though the condition j<100 is still true

The continue Statement

With the continue statement we can stop the current iteration, and continue with the next:

The else Statement

We use else statement to run a block of code once when the condition no longer is true:

```
sum=0
average=0
j=0
while j<=2:
```

```
    sum+=j
    j=j+1
else:
    average=sum/3
    print(average)
```

3.4.2. Python For Loops

A for loop is used for iterating over sequence data types; *list, a tuple, a dictionary, a set, or a string*

This is the opposite of for keyword found in other programming languages, and works more like an iterator method as found in other object-orientated.

An example is presented below whereby a for loop is used to iterate through a list known as cars. The program prints each car in car list

```
cars = ["benz", "volvo", "prado"]
for i in cars:
  print(i)
```

The for loop does not require an indexing variable to set beforehand.

Strings are iterable objects since they contain a sequence of characters hence, we can use the for loop to iterate each character in the string

```
for x in "programming":
  print(x)
```

3.4.2.1 The break Statement

Just like while loop, t break statement can be used to stop the loop before it has looped through all the items:

```
Exit the loop when i is "volvo":
cars = ["benz", "volvo", "prado"]
for i in cars:
  print(i)
  if i== "banana":
    break
```

3.4.2.2 The range () Function

We can decide to loop through a set of code a specified number of times, by using the range () function. The returns a sequence of numbers, starting from 0 by default, and increments by 1 (by default), and ends at a specified number. An example is shown below

```
for k in range(4):
  print(k)
```

Note that range(5) is not the values of 0 to 6, but the values 0 to 5.

The range() function defaults to 0 as a starting value, however it is possible to specify the starting value by adding a parameter: range(2, 4), which means values from 2 to 4 but excluding 4

The range() function defaults to increment the sequence by 1, however it is possible to specify the increment value by adding a third parameter: range(2, 30, **3**):

We can also decide to change the increment value from default value of 1 as shown below. The following example increments by value of 2

```
for k in range(2, 100, 3):
  print(k)
```

3.4.2.3 Else in For Loop

The else keyword in a for loop specifies a block of code to be executed when the loop has terminated

```
sum=0
even=0
for j in range(0,101,2):
   if j%2==0:
      sum+=j
      even+=1
   j=j+1
else:
   print((sum/even))
```

The above program prints the average of even numbers from 0 to 100 using a for loop.

3.5 chapter summary