2020

Free Python Course

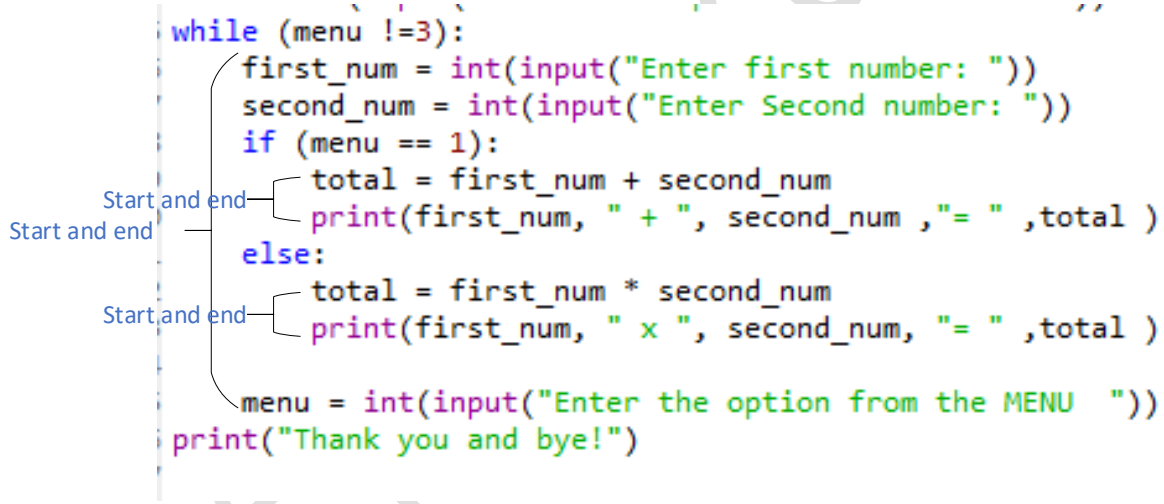
O. Elijah
onlinepythonclass@gmail.com

1 INTRODUCTION

Welcome to chapter three of the free python course. In this chapter we shall look at use of flow controls. Flow control in Python comes in two forms: decision making and loops. Decision making include the if statements, if…else statements and nested if statements. The loop includes the while loop, for loop and nested loop.

Note:

Python uses white space changes to indicate the start and end of flow control blocks, and so indention matters. For example, when using if . . . elif . . . else blocks (block means multiple lines of codes), all of the control blocks must have the same indentation level and all of the statements inside the control blocks should have the same level of indentation. Returning to the previous indentation level instructs Python that the block is complete. Example is shown as follows. The start and end of each block are shown. The spyder editor automatically indents when you start a block of code.




```
while (menu !=3):
    first_num = int(input("Enter first number: "))
    second_num = int(input("Enter Second number: "))
    if (menu == 1):
        total = first_num + second_num
        print(first_num, " + ", second_num ,"= " ,total )
    else:
        total = first_num * second_num
        print(first_num, " x ", second_num, "= " ,total )

    menu = int(input("Enter the option from the MENU  "))
print("Thank you and bye!")
```

What we intend to achieve

1. Learn how to use the if, if..else and nested if statements
2. Learn how to use the while loop, for loop and nested loop.
3. Learn how to use the loop control statements.

1.1 Review of chapter 2

In chapter two, we looked at the different data types and some basic operators in python. This forms the foundation of your python programming. You need to know when to use the different data types. We shall apply the knowledge in the following chapters.

1.2 Basic operators - Continued

Bitwise operators

Bitwise operator works on bits and performs bit by bit operation

Logics

OR, AND, exclusive OR, NOR

Truth Table of Logics

1. OR - if you have two inputs x and y and an output z

Input x	Input y	Output z
0	0	0
0	1	1
1	0	1
1	1	1

2. AND - if you have two inputs x and y and an output z

Input x	Input y	Output z
0	0	0
0	1	0
1	0	0
1	1	1

3. Exclusive OR (XOR)

Input x	Input y	Output z
0	0	0
0	1	1
1	0	1
1	1	0

4. NOR or NOT or complement

Input x	Output
0	1
1	0

Bitwise operator works on bits and performs bit by bit operation

Bitwise operators	Syntax
Or	|
And	&
Exclusive Or	^
Complement	~
Binary Left shift	<< The left operands value is moved left by the number of bits specified by the right operand.
Binary right shift	>>

Examples of bitwise operators

OR	>>> c = 2
	>>> b = 3
	>>> d = c|b
	>>> 3
		1 0
		1 1
		
		1 1

AND	>>> a = 2
	>>> b = 3
	>>> c = a& b
	>>> 2
		1 0
		1 1
		
		1 0

Exclusive OR	>>> a = 2
	>>> b = 3
	>>> e = a^b
	>>> e
	1
		1 0
		1 1
		
		0 1

Complement	>>> a = ~(60)
	>>> a
	-61
		a = 0011 1100
		(~a) = -61 (means 1100 0011 in 2's complement form due to a signed binary number.

<< Binary Left Shift	a= 60 a << 2 = 240	a = 0011 1100 (means 1111 0000)
>> Binary Right Shift	a= 60 a >> 2 = 15	(means 0000 1111)

Membership operators

Python's membership operators test for membership in a sequence, such as strings, lists, or tuples. There are two membership operators as explained below –

Operator	Description	Example

In	Evaluates to true if it finds a variable in the specified sequence and false otherwise.	`a = 10`
		
		`a = 20`
		
		`list = [1, 2, 3, 4, 5];`
		
		`if (a in list):`
		
		`  print ("Line 1 - a is available in the given list")`
		
		`else:`
		
		`  print ("Line 1 - a is not available in the given list")`

Not in	Evaluates to true if it does not finds a variable in the specified sequence and false otherwise.	`if (b not in list):`
		
		`  print ("Line 2 - b is not available in the given list")`
		
		`else:`
		
		`  print ("Line 2 - b is available in the given list")`

Identity Operators

Identity operators compare the memory locations of two objects. There are two Identity operators explained below –

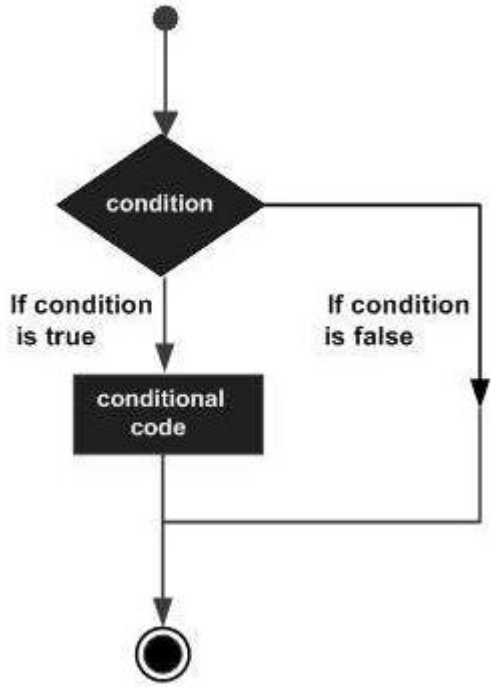
Operators	Description	Example

is	Evaluates to true if the variables on either side of the operator point to the	`a = 20`
		
		`b = 20`
		
		`if (a is b):`

	same object and false otherwise.	print "Line 1 - a and b have same identity"
		
		else:
		
		print ("Line 1 - a and b do not have same identity")
		
		if (id(a) == id(b)):
		
		print ("Line 2 - a and b have same identity")
		
		else:
		
		print "Line 2 - a and b do not have same identity"
		
		b = 30
		
		if (a is b):
		
		print ("Line 3 - a and b have same identity")
		
		else:
		
		print ("Line 3 - a and b do not have same identity")
		
		if (a is not b):
		
		print ("Line 4 - a and b do not have same identity")
		
		else:
		
		print ("Line 4 - a and b have same identity")

is not	Evaluates to false if the variables on either side of the operator point to the same object and true otherwise.	

1.3 Decision Making



The types of decision making statements are

Types	Description
if statement	An if statement consists of a boolean expression followed by one or more statements.
if….else statements	if...else statements An if statement can be followed by an optional else statement, which executes when the boolean expression is FALSE.
Nested if statements	nested if statements You can use one if or else if statement inside another if or else if statement(s).

Examples

If single statement suites -

```
var = 100

if (var == 100) : print ("Value of expression is 100")

print ("Good bye!")
```

result of executed code is

```
Value of expression is 100
Good bye!
```

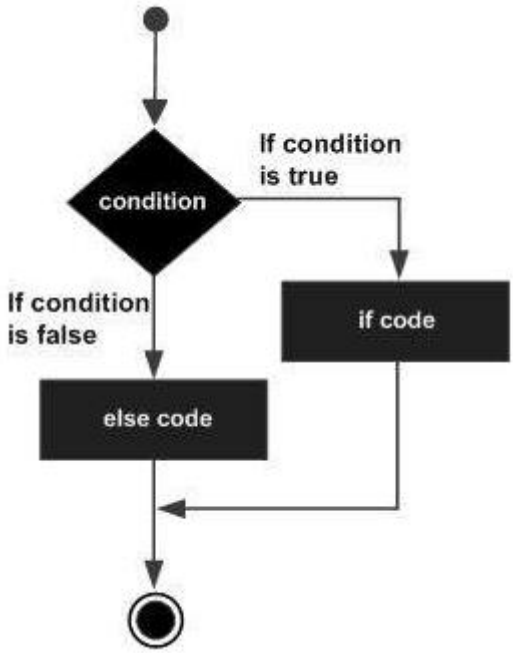
If else statement

An else statement contains the block of code that executes if the conditional expression in the if statement resolves to 0 or a FALSE value. The else statement is an optional statement and there could be at most only one else statement following if.

Syntax

The syntax of the if...else statement is –

```
if expression:
   statement(s)
else:
   statement(s)
```



Flow Diagram

Example

```
var1 = 100

if var1:

   print ("1 - Got a true expression value")
```

```
   print (var1)
else:
   print ("1 - Got a false expression value")
   print (var1)


var2 = 0
if var2:
   print ("2 - Got a true expression value")
   print (var2)
else:
   print ("2 - Got a false expression value")
   print (var2)
print ("Good bye!")
```

The result of executed code is

```
1 - Got a true expression value
100
2 - Got a false expression value
0
Good bye!
```


Elif Statement

The **elif** statement allows you to check multiple expressions for TRUE and execute a block of code as soon as one of the conditions evaluates to TRUE. There can be an arbitrary number of **elif** statements following an **if**

Syntax

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

```
   statement(s)
```

Example

```
var = 100

if var == 200:

   print ("1 - Got a true expression value")

   print (var)

elif var == 150:

   print ("2 - Got a true expression value")

   print (var)

elif var == 100:

   print ("3 - Got a true expression value")

   print (var)

else:

   print ("4 - Got a false expression value")

   print (var)


print ("Good bye!")
```

result of executed code is

```
3 - Got a true expression value
100
Good bye!
```

Nested If Statements

Syntax nested *if…elif…else*

```
if expression1:
   statement(s)
   if expression2:
      statement(s)
   elif expression3:
      statement(s)
   else:
      statement(s)
   elif expression4:
      statement(s)
else:
   statement(s)
```

Example

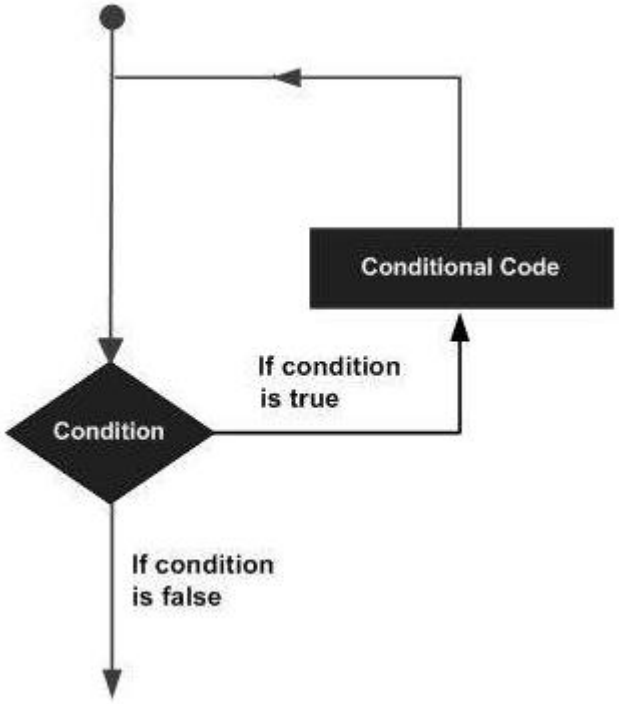
```
var = 100

if var < 200:

   print ("Expression value is less than 200")

   if var == 150:

      print ("Which is 150")

   elif var == 100:

      print ("Which is 100")

   elif var == 50:

      print ("Which is 50")

   elif var < 50:

      print ("Expression value is less than 50")

else:

   print ("Could not find true expression")


print ("Good bye!")
```

result of executed code is

```
Expression value is less than 200
Which is 100
Good bye!
```

1.4 Loops

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



Flow diagram of loop statement

In general, statements are executed sequentially: The first statement in a function is executed first, followed by the second, and so on.

Types of loops

Type	Description
For	Executes a sequence of statements multiple times and abbreviates the code that manages the loop variable.
While	Repeats a statement or group of statements while a given condition is TRUE. It tests the condition before executing the loop body.
Nested Loops	You can use one or more loop inside any another while, for or do..while loop.

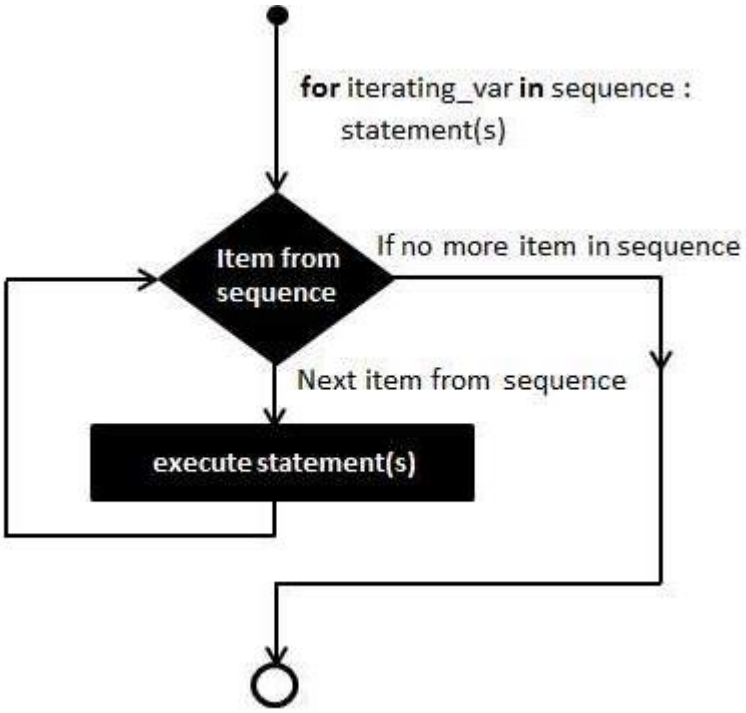
For loop

It has the ability to iterate over the items of any sequence, such as a list or a string.

Syntax

```
for iterating_var in sequence:
   statements(s)
```

If a sequence contains an expression list, it is evaluated first. Then, the first item in the sequence is assigned to the iterating variable iterating_var. Next, the statements block is executed. Each item in the list is assigned to iterating_var, and the statement(s) block is executed until the entire sequence is exhausted.



Flow diagram

Examples

```
for letter in 'Python':     # First Example

   print ('Current Letter :', letter)  # the word letter is a iterating variable


i = ['banana', 'apple',  'mango']

for i in fruits:        # Second Example

   print ('Current fruit :', i)


print ("Good bye!")
```

results of executed codes

```
Current Letter : P
Current Letter : y
Current Letter : t
```

```
Current Letter : h
Current Letter : o
Current Letter : n
Current fruit : banana
Current fruit : apple
Current fruit : mango
Good bye!
```

Iterating by sequence

An alternative way of iterating through each item is by index offset into the sequence itself. Following is a simple example –

Example

```
fruits = ['banana', 'apple',  'mango']

for index in range(len(fruits)):

   print ('Current fruit :', fruits[index])

print ("Good bye!")
```

While Loop

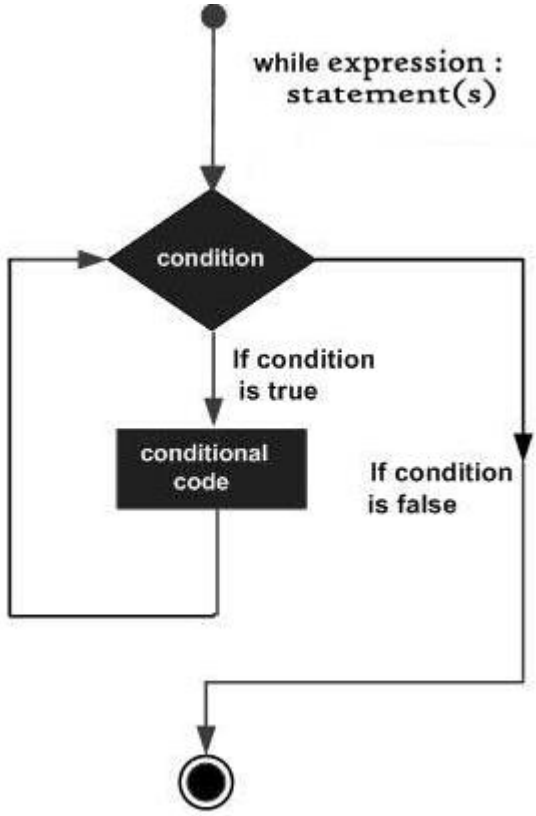
A while loop statement in Python programming language repeatedly executes a target statement as long as a given condition is true.

Syntax

```
while expression:
   statement(s)
```

Here, **statement(s)** may be a single statement or a block of statements. The **condition** may be any expression, and true is any non-zero value. The loop iterates while the condition is true.

Python uses indentation as its method of grouping statements. All the statements indented by the same number of character spaces after a programming construct are considered to be part of a single block of code.



Flow diagram

When the condition becomes false, program control passes to the line immediately following the loop.

Examples

```
count = 0

while (count < 9):

   print ('The count is:', count)

   count = count + 1

print ("Good bye!")
```

result of executed code is

```
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
Good bye!
```

The infinite loop

A loop becomes infinite loop if a condition never becomes FALSE. You must use caution when using while loops because of the possibility that this condition never resolves to a FALSE value. This results in a loop that never ends. Such a loop is called an infinite loop.

An infinite loop might be useful in client/server programming where the server needs to run continuously so that client programs can communicate with it as and when required.

Example

```
var = 1

while var == 1 :  # This constructs an infinite loop

   num = input("Enter a number  :")

   print ("You entered: ", num)

print ("Good bye!")
```

result of executed code is

```
Enter a number  :20
You entered:  20
Enter a number  :29
You entered:  29
Enter a number  :3
You entered:  3
Enter a number between :Traceback (most recent call last):
   File "test.py", line 5, in <module>
      num = raw_input("Enter a number :")
KeyboardInterrupt
```

Using else statement with loops

If the **else** statement is used with a **while** loop, the **else** statement is executed when the condition becomes false.

Example

```
count = 0

while count < 5:

   print (count, " is  less than 5")

   count = (count + 1)

else:

   print (count, " is not less than 5")
```

the code the while statement prints a number as long as it is less than 5, otherwise else statement gets executed.

Results of the executed code is

```
0 is less than 5
1 is less than 5
2 is less than 5
3 is less than 5
4 is less than 5
5 is not less than 5
```

one-line while clause –

if your while clause consists only of a single statement, it may be placed on the same line as the while header.

Examples

```
flag = 1

while (flag): print 'Given flag is really true!'

print "Good bye!"
```


Loop Control Statements

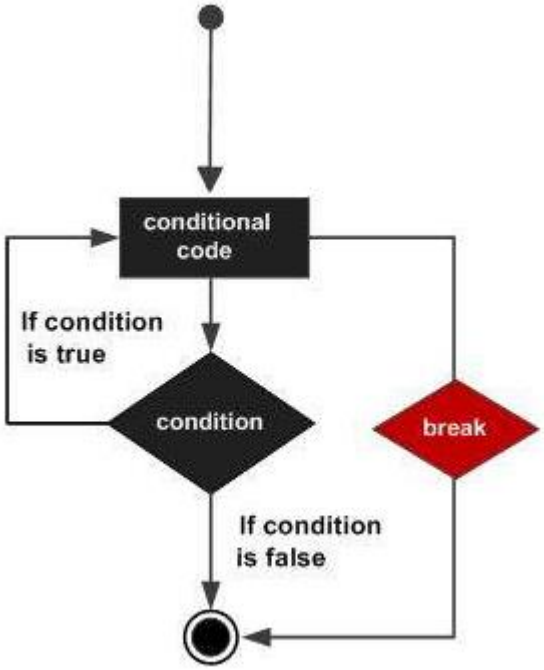
Type	Description
Break statement	Terminates the loop statement and transfers execution to the statement immediately following the loop.
Continue statement	Causes the loop to skip the remainder of its body and immediately retest its condition prior to reiterating.
Pass Statement	The pass statement in Python is used when a statement is required syntactically but you do not want any command or code to execute.

Break Statement

It terminates the current loop and resumes execution at the next statement, just like the traditional break statement in C.

The most common use for break is when some external condition is triggered requiring a hasty exit from a loop. The **break** statement can be used in both *while* and *for* loops.

If you are using nested loops, the **break** statement stops the execution of the innermost loop and start executing the next line of code after the block.



Flow diagram

Examples

```
for letter in 'Python':     # First Example
   if letter == 'h':
      break
   print ('Current Letter :', letter)
var = 10                    # Second Example
while var > 0:
   print ('Current variable value :', var)
   var = var -1
   if var == 5:
      break
print ("Good bye!")
```

Result of executed code

```
Current Letter : P
Current Letter : y
Current Letter : t
```

```
Current variable value : 10
Current variable value : 9
Current variable value : 8
Current variable value : 7
Current variable value : 6
Good bye!
```

Continue Statement

It returns the control to the beginning of the while loop.. The continue statement rejects all the remaining statements in the current iteration of the loop and moves the control back to the top of the loop.

Examples

```
for letter in 'Python':     # First Example
   if letter == 'h':
      continue
   print ('Current Letter :', letter)

var = 10                    # Second Example
while var > 0:
   var = var -1
   if var == 5:
      continue
   print ('Current variable value :', var)
print "Good bye!"
```

results of executed code

```
Current Letter : P
Current Letter : y
Current Letter : t
Current Letter : o
Current Letter : n
Current variable value : 9
Current variable value : 8
Current variable value : 7
Current variable value : 6
Current variable value : 4
Current variable value : 3
```

```
Current variable value : 2
Current variable value : 1
Current variable value : 0
Good bye!
```

Pass statement

It is used when a statement is required syntactically but you do not want any command or code to execute.

The pass statement is a null operation; nothing happens when it executes. The pass is also useful in places where your code will eventually go, but has not been written yet (e.g., in stubs for example) –

Example

```
for letter in 'Python':

   if letter == 'h':

      pass

      print ('This is pass block')

   print ('Current Letter :', letter)

print "Good bye!"
```

1.5 try . . . except

try . . . except blocks are useful for running code which may fail for reasons outside of the programmer's control

Syntax

```
try:
    Dangerous Code
except ExceptionType1:
    Code to run if ExceptionType1 is raised
except ExceptionType2:
    Code to run if ExceptionType1 is raised
...
...
except:
    Code to run if an unlisted exception type is raised
```

Example

```
text = ('a','1','54.1','43.a')
for t in text:
    try:
        temp = float(t)
        print(temp)
    except ValueError:
        print('Not convertible to a float')
```

1.6 Application

Now let us see how we can apply what we have just learnt from this chapter.

Example 1.

Suppose your school uses the following 5 - point grading scale for exams: 75 and above is A, 70 – 74 is AB, 65 – 69 is B, 60 – 64 is BC, 55 – 59 is C, 50 – 54 is CD, 45 – 49 is D, 40 – 44 is E and below 40 is F. Write a python program that will allow a student to enter a test score and then display the grade for that score using the if – else statement.

Solution

Examples

```
score = int(input("Please Enter score: "))

if score > 75:

    print("Grade is A")

if (score < 75) & (score >=70)  :

    print("Grade is AB")

if (score <70) & (score >= 65):

    print("Grade is B")

if (score <65) & (score >=60 ):

    print("Grade is BC")

if (score <60 ) & (score >=55):

    print(" Grade is C")

if (score <55) & (score >=50):

    print("Grade is CD ")

if (score < 50) & (score >=45):

    print("Grade is D")

if (score < 45) & (score >=40):

    print("Grade is E")

if (score < 40) :

    print("Grade is F")
```

Result

```
Please Enter score: 45
Grade is D
```

Example 2

Write a program to create a user menu to add, multiply and exit as shown below

```
********** MENU **********
   (1) Addition
   (2) Multiplication
   (3) Exit
**************************

Choose the operation to do: 1

Enter two numbers: 45 20

45 + 20 = 65

Press y if you want repeat ... y


********** MENU **********
   (1) Addition
   (2) Multiplication
   (3) Exit
**************************

Choose the operation to do: 2

Enter two numbers: 3 7

3 x 7 = 21

Press y if you want repeat ... n
```

Solution

There are several ways to implement this in python code. We shall use while statement and if ..else statement

```
# Program menu

print(" ********** MENU  **************")

print("  (1) Addition                  ")

print("  (2) Multiplication            ")

print("  (3) Exit                      ")

menu = int(input(" Enter the option from the MENU:  "))

while (menu !=3):

    first_num = int(input("Enter first number: "))

    second_num = int(input("Enter Second number: "))

    if (menu == 1):

        total = first_num + second_num

        print(first_num, " + ", second_num ,"= " ,total )

    else:

        total = first_num * second_num

        print(first_num, " x ", second_num, "= " ,total )


    menu = int(input("Enter the option from the MENU  "))

print("Thank you and bye!")
```

Result

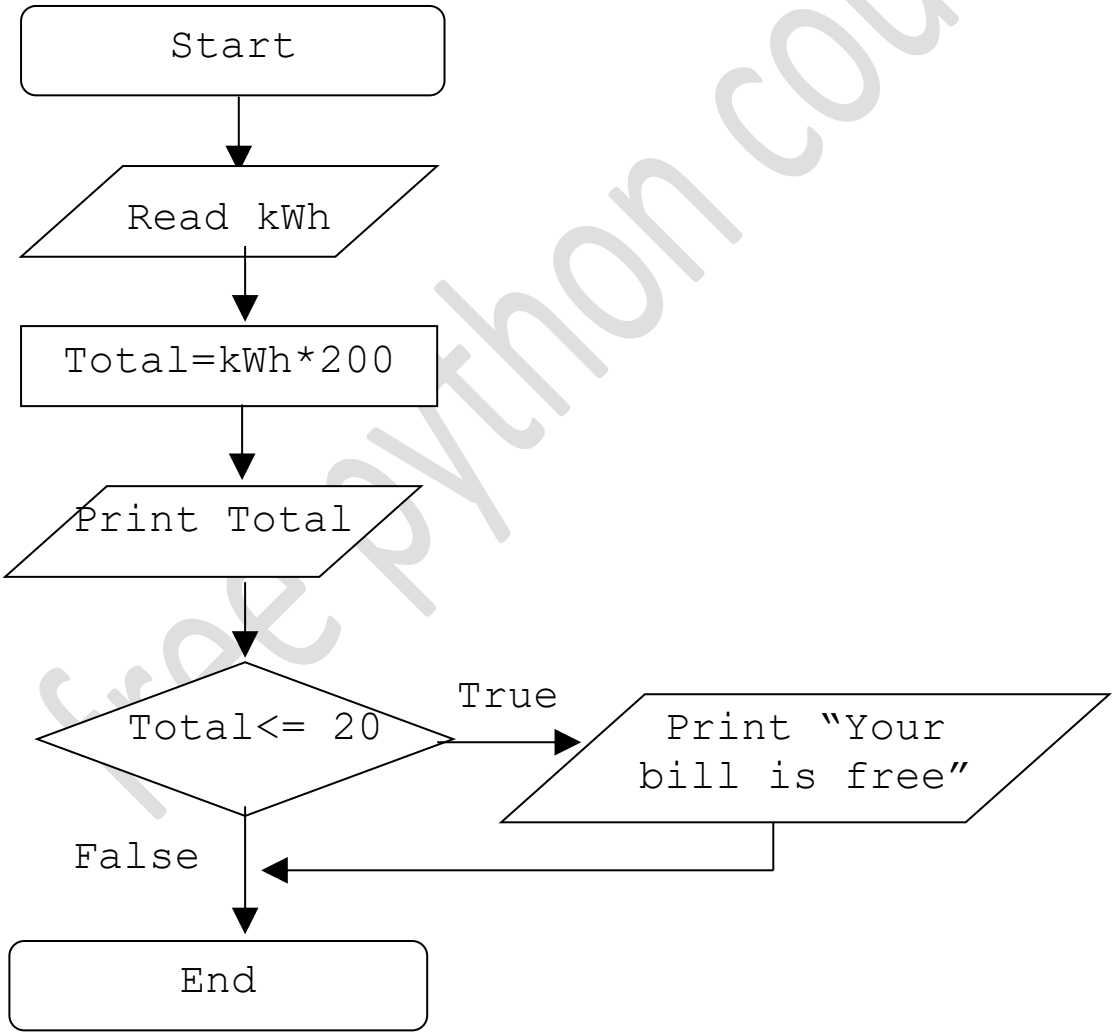
```
********** MENU  **************
  (1) Addition
  (2) Multiplication
  (3) Exit

 Enter the option from the MENU:   2

Enter first number: 22

Enter Second number: 22
22  x  22 =  484

Enter the option from the MENU  1

Enter first number: 22
```

```
Enter Second number: 33
22  +  33 =  55

Enter the option from the MENU  3
Thank you and bye!
```

Example 3

A domestic electricity usage will charge with basic rate (kWh) #220 (Naira Nigerian currency). Calculate the bill by giving the power consumed (kWh). Print the total amount bill and if it is less than or equal to #2000, print "Your bill is free".

The algorithm for the solution using flowchart is shown as follows



The python code is as follows

```
# Program to compute electricity bill

tariff = 200

kWh = int(input("enter total consumption (kWh): " ))

bill = kWh* tariff

print("Your total bill is #", bill)

if (bill <=2000):

      print("Your bill is free!!!")
```

Result

```
Enter total consumption (kWh): 100
Your total bill is # 20000


Enter total consumption (kWh): 5
Your total bill is # 1000
Your bill is free!!!
```


1.7 Summary

1. First, we have learnt about logical operators, decision making and loops.
2. We have practiced some example and seen applications of the theory learnt.

1.8 Exercise

1. Michael teaches a python class and his students are required to take three tests. Write a program that will ask the teacher to input the 3 test scores and calculate the average test score. The program should also congratulate the student if the average is greater than 95. (**Hint:** Use the if statement).
2. Suppose your school uses the following 5 - point grading scale for exams: 75 and above is A, 70 – 74 is AB, 65 – 69 is B, 60 – 64 is BC, 55 – 59 is C, 50 – 54 is CD, 45 – 49 is D, 40 – 44 is E and below 40 is F. Write a python program that will allow a student to enter a test score and then display the grade for that score using the using elif statement.
3. The test score of Class A students are given as follows

Names	Maths Score	English score
John	67	98
Annis	88	80
Fatimah	78	85
Caleb	75	89
Aruna	60	93

Write a program to the names of the student in a list, the maths score in a list and the English score in a list. Compute the average score of maths and the average score of English for the class. Compute the total score for each student and print them out using a for loop. Your result should be something like this

Average score of students in Maths is: 73.6

Average score of students in English is: 89

Names	Total score
John	165
Annis	168
Fatimah	163
Caleb	164
Aruna	153

Exercise is due on the 5Th of February

1.9 Reference

[1] http://www.tutorialspoint.com/python/python_basic_operators.htm

[2] https://www.tutorialspoint.com/python/python_loops.htm

Chinese Proverbs.

Teach a Man to Fish Shòu rén yǐ yú bùrú shòu rén yǐ yú

– meaning Give a man a fish and you feed him for one day. Teach a man to fish and you feed him for a lifetime'.