UNIT 2

Decision making and Loops

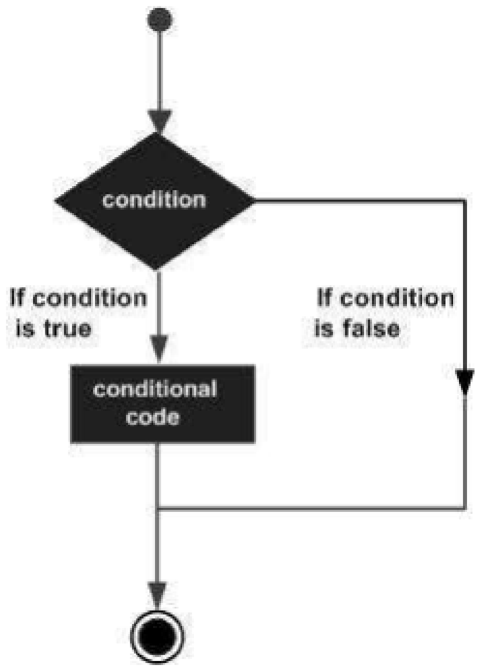
Q1. What is decision making statements? Write down the different decision making statements.

Ans.

Decision-making is the anticipation of conditions occurring during the execution of a program and specified actions taken according to the conditions.

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

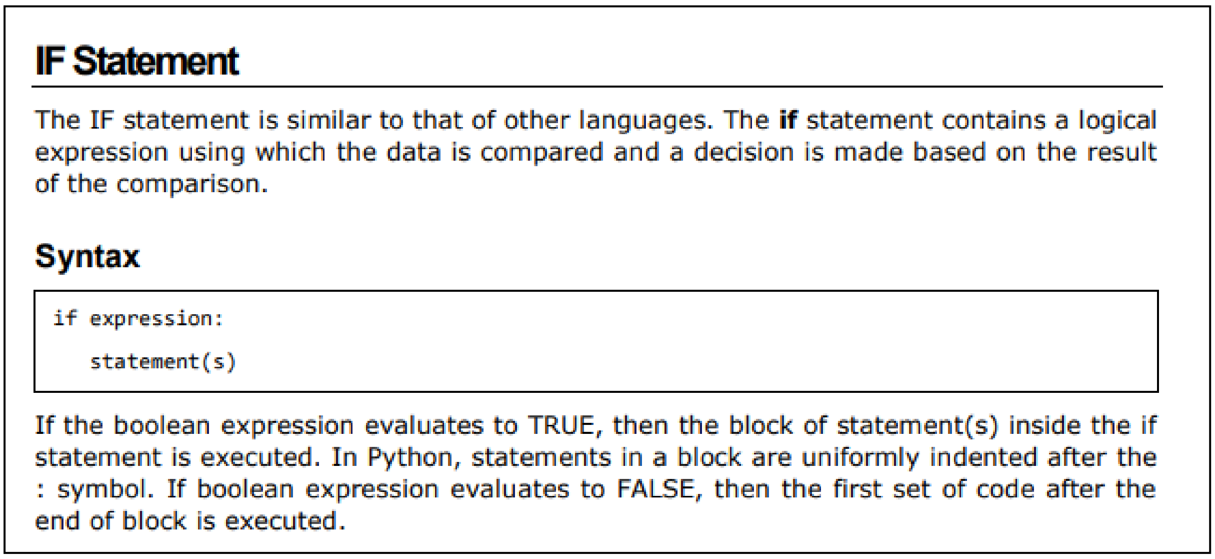
Following is the general form of a typical decision making structure found in most of the programming languages-



Python programming language assumes any **non-zero** and **non-null** values as TRUE, and any **zero** or **null values** as FALSE value.

Python provides various types of conditional statements:

Statement	Description
if Statements	It consists of a Boolean expression which results are either TRUE or FALSE, followed by one or more statements.
if else Statements	It also contains a Boolean expression. The if statement is followed by an optional else statement & if the expression results in FALSE, then else statement gets executed.
Nested Statements	You can use one if or else statement inside another if or else if statements(s)



IF Statement

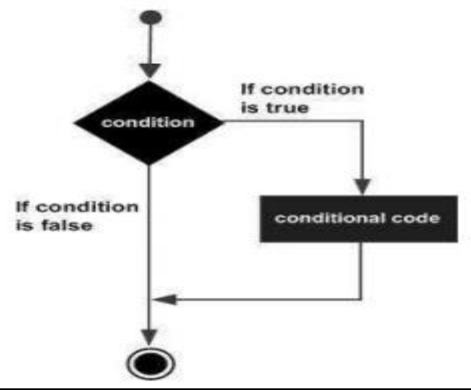
The IF statement is similar to that of other languages. The **if** statement contains a logical expression using which the data is compared and a decision is made based on the result of the comparison.

Syntax

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

If the boolean expression evaluates to TRUE, then the block of statement(s) inside the if statement is executed. In Python, statements in a block are uniformly indented after the : symbol. If boolean expression evaluates to FALSE, then the first set of code after the end of block is executed.

Flow Diagram

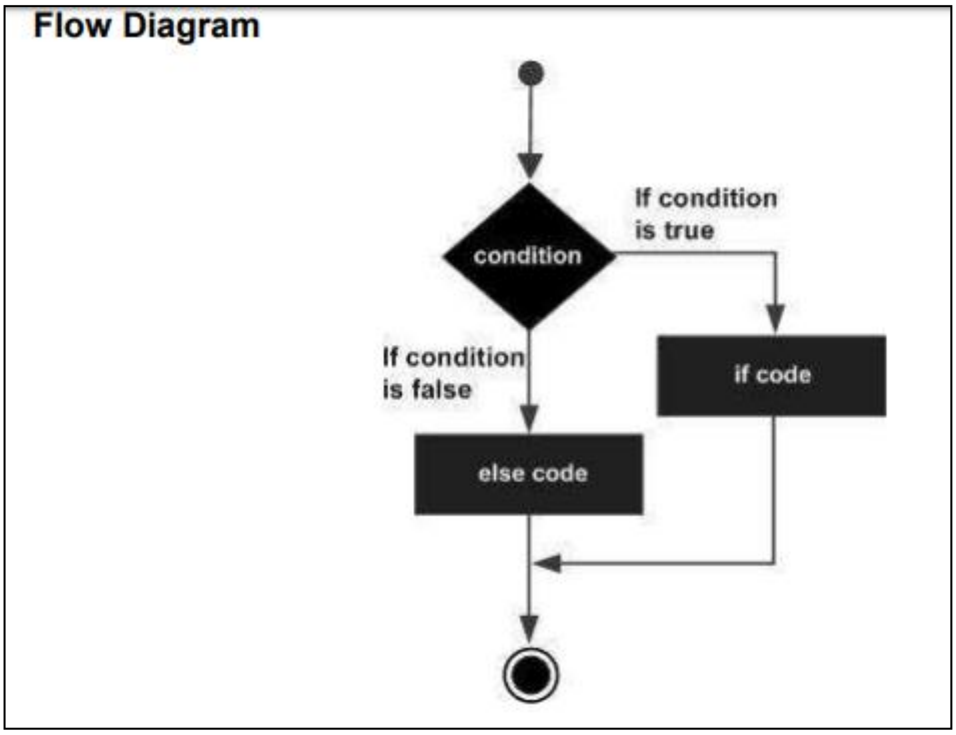


Python If-Else Statement

The if statement alone tells us that if a condition is true it will execute a block of statements and if the condition is false it won't. But if we want to do something else if the condition is false, we can use the *else* statement with *if* statement to execute a block of code when the if condition is false.

Syntax:

```
if (condition):

    # Executes this block if

    # condition is true

else:

    # Executes this block if

    # condition is false
```



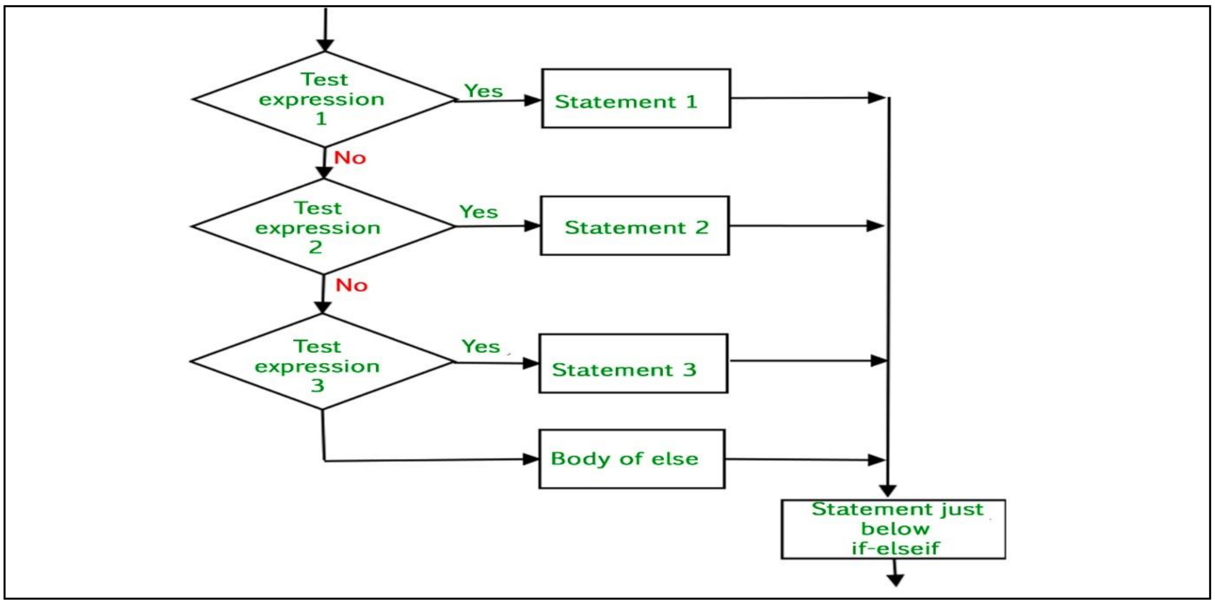
Python if-elif-else Ladder

Here, a user can decide among multiple options. The if statements are executed from the top down. As soon as one of the conditions controlling the if is true, the statement associated with that if is executed, and the rest of the ladder is bypassed. If none of the conditions is true, then the final else statement will be executed.

Syntax:

```
if (condition):

    statement

elif (condition):

    statement

.

.

else:

    statement
```

Flowchart of Python if-elif-else ladder



Q2. Program to check whether a number is positive or negative.

Ans:

```
n=int(input("Enter number: "))
if(n>0):
   print("Number is positive")
else:
   print("Number is negative")
```

using Shift operator

```
if(n&1==0):
        print(“even”)
else:
        print(“odd”)
```

Q3. Program to take in the marks of 5 subjects and display the grade.

Ans:

```
sub1=int(input("Enter marks of the first subject: "))
sub2=int(input("Enter marks of the second subject: "))
sub3=int(input("Enter marks of the third subject: "))
sub4=int(input("Enter marks of the fourth subject: "))
sub5=int(input("Enter marks of the fifth subject: "))
avg=(sub1+sub2+sub3+sub4+sub4)/5
if(avg>=90):
   print("Grade: A")
elif(avg>=80&avg<90):
   print("Grade: B")
elif(avg>=70&avg<80):
   print("Grade: C")
elif(avg>=60&avg<70):
   print("Grade: D")
else:
   print("Grade: F")
```

Q4. **Program to check whether a given year is a leap year or not.**

Ans:

```
year=int(input("Enter year to be checked:"))
if(year%4==0 and year%100!=0 or year%400==0):
   print("The year is a leap year!)
else:
   print("The year isn't a leap year!)
```

Q Program to check whether number is positive or negative.

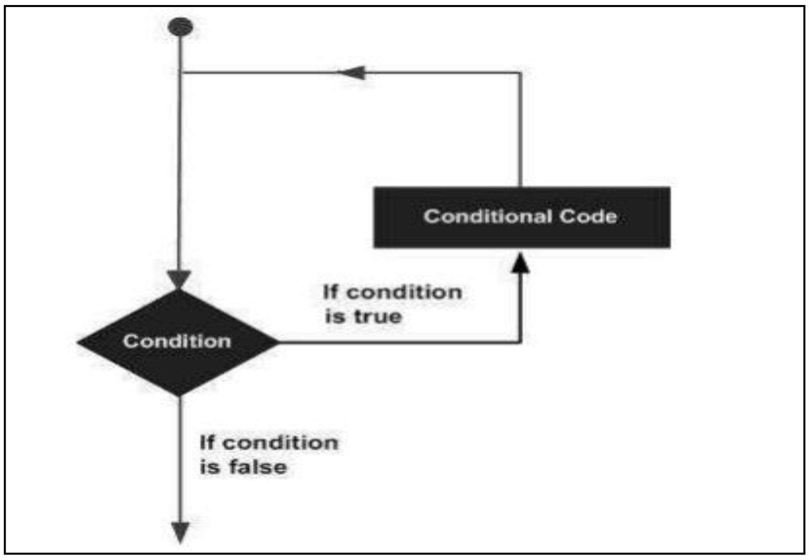
```
n=int(input("Enter number: "))
if(n>>31==0):
        print(“positive”)
else:
        print(“negative”)
```

Loops in python

Generally Statements are executed sequentially, but there sometimes occur such cases where programmers need to execute a block of code several times. The control structures of programming languages allow us to execute a statement or block of statements repeatedly.

Loops are a sequence of instructions that does a specific set of instructions or tasks based on some conditions and continue the tasks until it reaches certain conditions.

Following diagram illustrates a loop statement .



Python language provides the different types of loop –

Loop Type	Description
While Loop	Repeats a statement or group of statements while a given condition is true. It tests the condition before executing the loop body.
for Loop	for loop is used for iterating over a sequence (that is either a list, a tuple, a dictionary, a set, or a string).
Nested loop	You can create one or more loops inside any other while or , for loop

While Loop in Python-

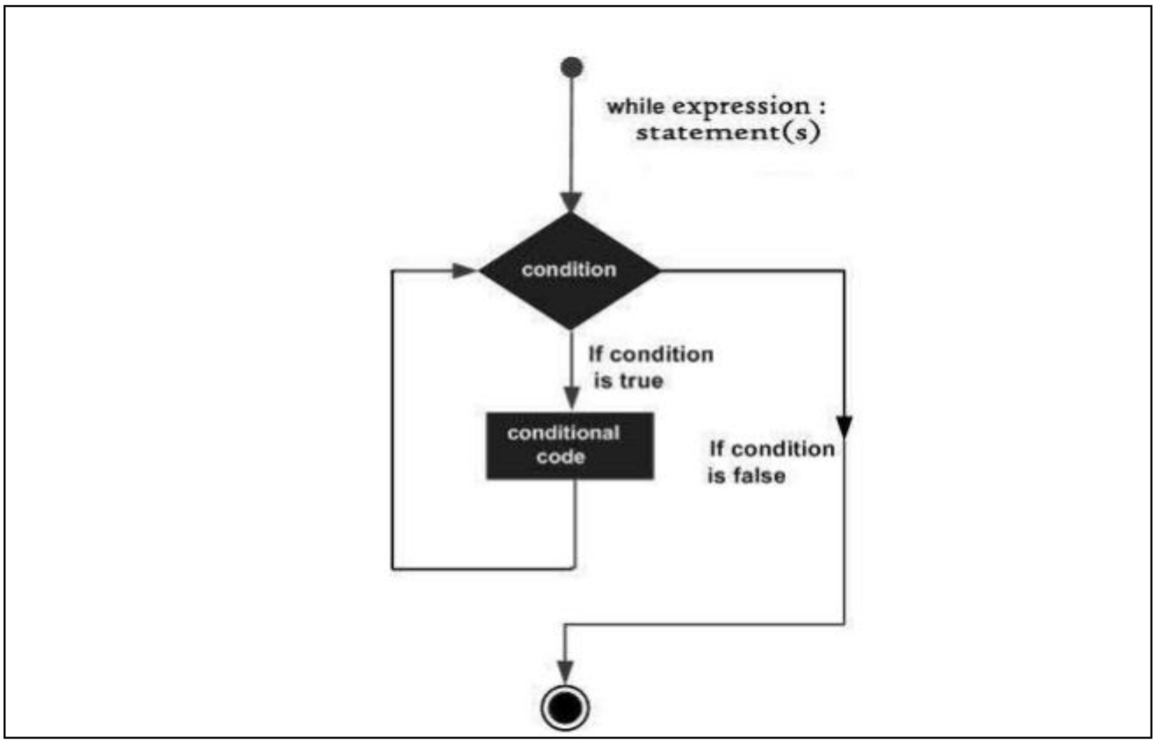
In python, a **while loop** is used to execute a block of statements repeatedly until a given condition is satisfied. And when the condition becomes false, the line immediately after the loop in the program is executed.

Syntax:
```
while expression:

    statement(s)
```

Statements may be a single or a block of statements with same indent .

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. Python uses indentation as its method of grouping statements.

Flowchart depicting while loop -



Example of Python While Loop

Let's see a simple example of while loop in Python.

```
# Python program to illustrate while loop
count = 0
while (count < 3):
    count = count + 1
    print("Hello")
```

Output –

```
Hello
Hello
Hello
```

- **Using else statement with While Loop in Python**

The else clause is only executed when your while condition becomes false. If you break out of the loop, or if an exception is raised, it won’t be executed.

Syntax of While Loop with else statement:

```
while condition:

     # execute these statements

else:

     # execute these statements
```

Examples of While Loop with else statement

```python
count = 0
while (count < 3):
    count = count + 1
    print("Hello Geek")
else:
    print("In Else Block")
```

Output

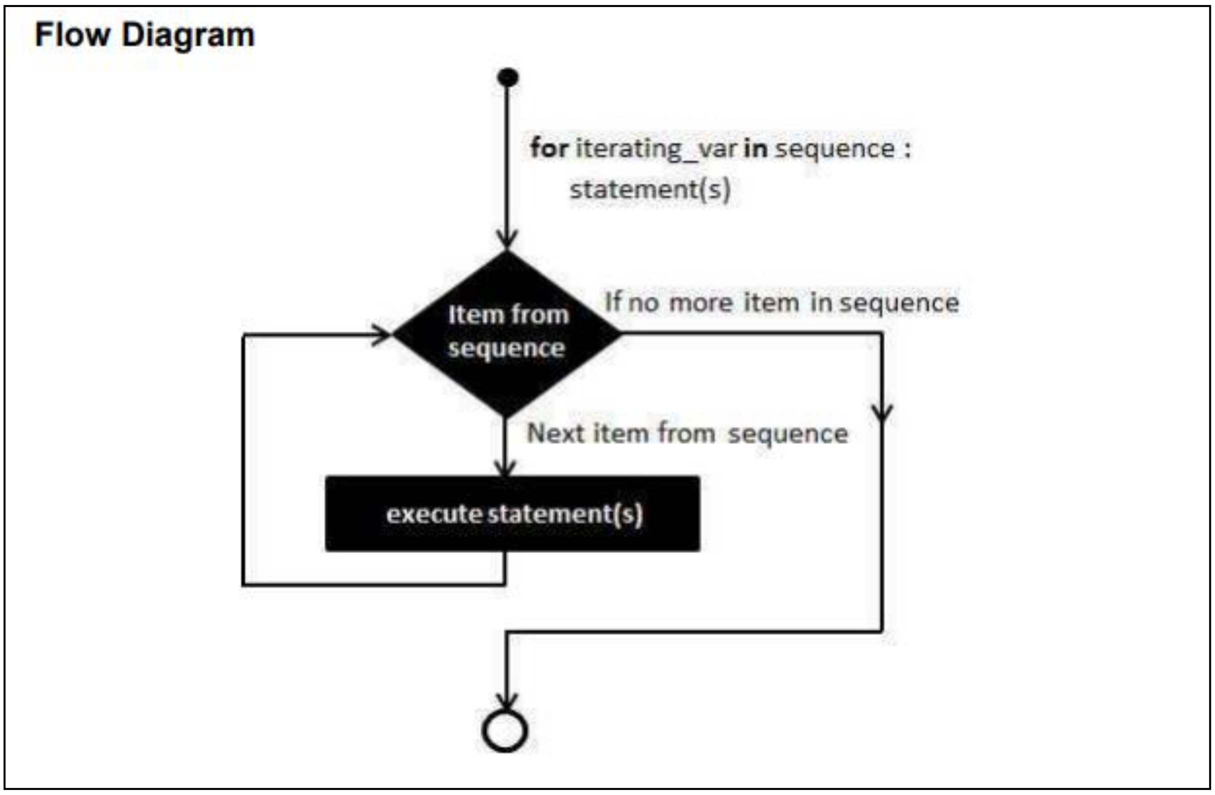
```
Hello Geek

Hello Geek

Hello Geek

In Else Block
```

<u>**For Loop in Python**</u>

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

Syntax:

```
for iterator_var in sequence:

    statements(s)
```



Example –

```
s = "apple"
for i in s:
    print(i)
Output -
a
p
p
l
e
```

The range() Function

To loop through a set of code a specified number of times, we can use the `range()` function,

- The `range()` function returns a sequence of numbers, starting from 0 by default, and increments by 1 (by default), and ends at a specified number.

Example-

```
for x in range(6):
  print(x)
```

- The `range()` function defaults to 0 as a starting value, however it is possible to specify the starting value by adding a parameter:

Example-

```
for x in range(2, 6):
  print(x)
```

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

Example-

```
for x in range(2, 30, 3):
  print(x)
```

Example with List, Tuple, string, and dictionary iteration using For Loops in Python

We can use for loop to iterate lists, tuples, strings and dictionaries in Python.

1. Example with List

```
languages = ["C", "JAVA", "PYTHON"]
for x in languages:
      print(x)
```

```
Output-
C
JAVA
PYTHON
```

2. Example with tuple

```
thistuple = ("apple", "banana", "cherry")

for x in thistuple:
      print(x)
```

```
apple
banana
cherry
```

3. Example with dict

```
thisdict =     {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}

for x in thisdict:
      print(x)
```

```
brand
model
year
```

4. Example with string

```
country=”INDIA”
for i in country:
      print(i)
```

```
I
N
D
I
A
```

Using else statement with for loop in Python

We can also combine else statement with for loop like in while loop. If else statement is used with for loop , then else statement will executed when the loop has finished iterating the given sequence .

Example-

```
languages = ["C", "JAVA", "PYTHON"]
for x in range(len(languages)):
        print(languages[x])
else:
        print("Inside else block")
```

```
C
JAVA
PYTHON
Inside else block
```

Nested loop in python-

Python programming language allows to use one loop inside another loop.

Syntax for for loop :

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

```
for x in range(2):
        for y in range(3):
                print(x,y)
```

Syntax for while loop:

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

```
x = [1, 2]
y = [4, 5]
i = 0
while i < len(x) :
  j = 0
  while j < len(y) :
    print(x[i] , y[j])
    j = j + 1
  i = i + 1
```

- **WAP to print the given sequence -**

  A A A
  B B
  C

ANS -
```
n=3
for i in range(n,0,-1):
            for j in range(0,i,1):
                        print(chr(65+n-i),end=" ")
           print()
```

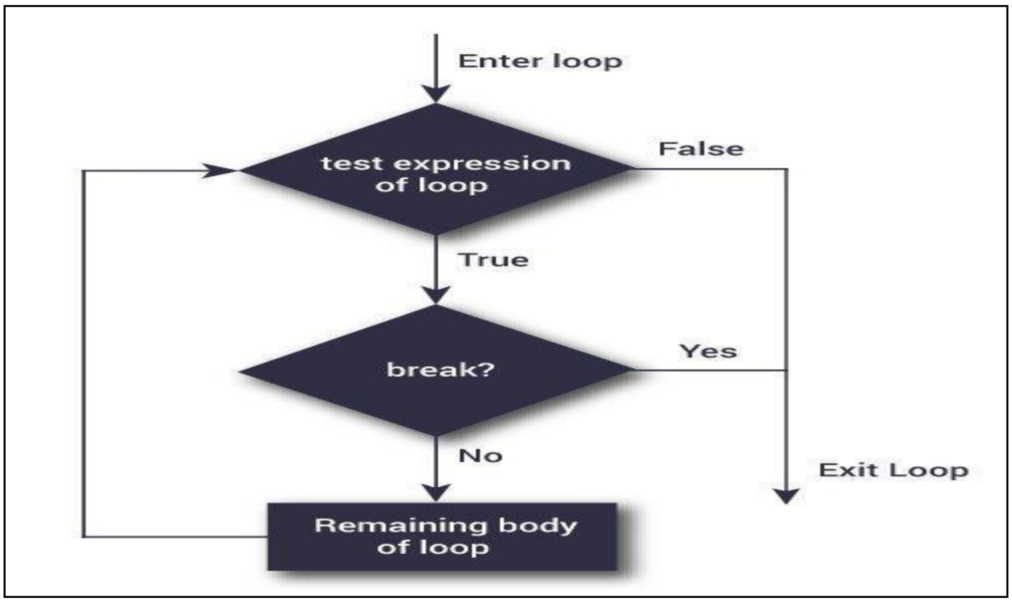
Loop Control Statements-

Loop control statements change execution from their normal sequence. When execution leaves a scope, all automatic objects that were created in that scope are destroyed. Python supports the following control statements.

Control Statement	**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 & immediately retest its condition prior to reiterating.
Pass statement	Used when a statement is required syntactically but you don’t want any command or code to execute

Break statement –

- **break statement in Python** is used to bring the control out of the loop when some external condition is triggered.
- break statement is put inside the loop body (generally after if condition). It terminates the current loop, i.e., the loop in which it appears, and resumes execution at the next statement immediately after the end of that loop.
- If the break statement is inside a nested loop, the break will terminate the innermost loop.



Example-

```
s = 'programming'                                    # example1

for letter in s:
    print(letter)
    if letter == 'o' or letter == 'm':
        break
print("Out of for loop")
```

```
p

r

out of for loop
```

```
 var =10

while(var>0):

        print(var)

        var=var-1

        if var ==5:

                break

print(“out of while loop)
```

```
10

9

8

7

6

Out of while loop
```

Continue Statement

Continue Statement is a loop control statement that forces to execute the next iteration of the loop while skipping the rest of the code inside the loop for current iteration only .

The syntax for continue statement is :

```
Continue
```

Example :

1.
```
for var in "PYTHON":
    if var == "Y":
        continue
    print(var)
```

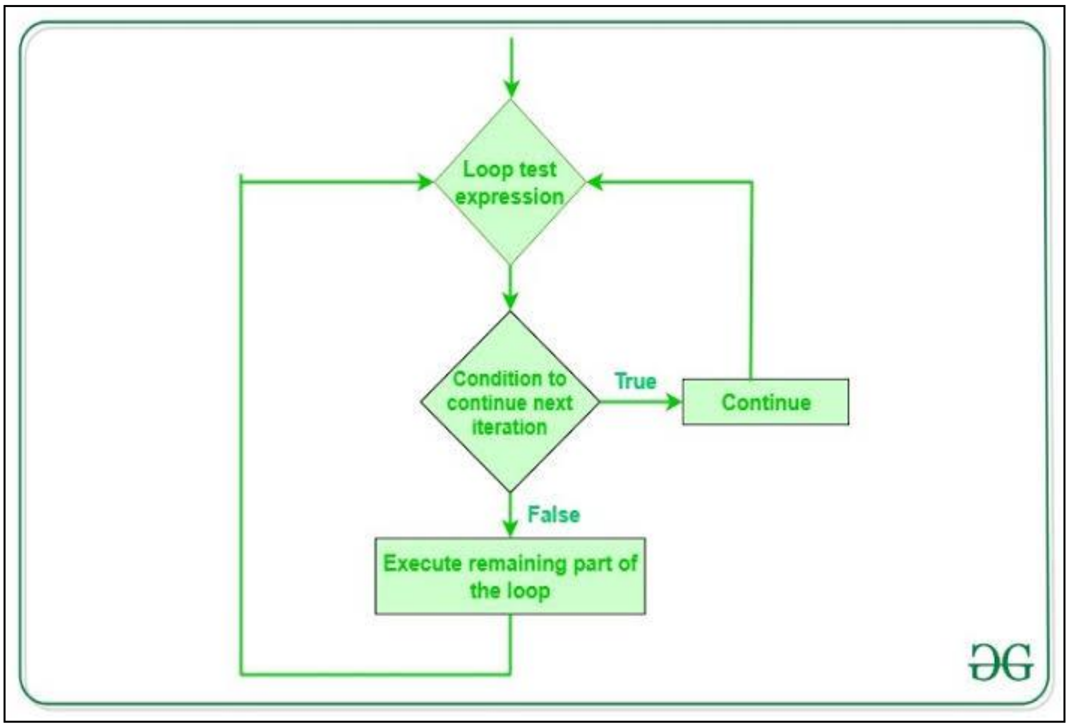
```
P
T
H
O
N
```

2.
```
i = 0
while i < 8:
```

```
0
1
2
3
4
6
7
```

```
    if i == 5:
        i += 1
        continue
    print(i)
    i += 1
```

FlowChart



Loop test expression
Condition to continue next iteration
True
Continue
False
Execute remaining part of the loop

Pass Statement

- It is used when a statement is required syntactically but you don't want any command or code to execute .

- Pass is also used for empty control statements, functions and classes.

Example -

```
for num in range(1,10):
    if num%2 != 0:
        pass
    else:
        print(num)
```

```
2
4
6
8
```

Note- Empty code is not allowed in loops , functions , if statements , class definitions or in function definitions .

```
a = 33
b = 22
if  b > a :
```

Output - raise an error because we cannot leave if statement empty .

```
a = 33
b = 22
if b > a :
    Pass.
```

Output - no error

Difference between Pass and Comments

Pass	Comments
Pass statement allow program to execute , nothing happens but you avoid getting an error when empty code is not allowed .	Comments are used to enhance the readability of the code.
Python interpreter does not ignored the pass statement during the execution of the	Python interpreter ignored all the comments during the execution of the program.

program	

Difference Between Break & Continue Statement

The continue Statement	The break Statement
The continue statement is used to skip an iteration of a for loop or a while loop in Python.	The break statement is used to terminate the execution of a for loop or a while loop in Python.
The continue statement can lead to an infinite loop when using a while loop if the statement to change the loop variable is written after the continue statement.	The break statement never causes an infinite loop.
We cannot use the continue statement outside a for loop or a while loop.	We cannot use the break statement outside a for loop or a while loop.
The continue statement can be executed multiple times in a for loop or while loop in Python.	The break statement can be executed only once inside a loop. After this, the execution of the loop will be terminated.