LOOPING / ITERATIVE STATEMENTS:

Iterative statements are decision control statements that repeat the execution of list of statements.

Two types of iterative statements supported by Python are as follows:

- While loop
- For loop

The decision for repeating the set of statements depends on the value of the Boolean expression.

WHILE LOOP:

Repeats a set of statements until a condition is satisfied.

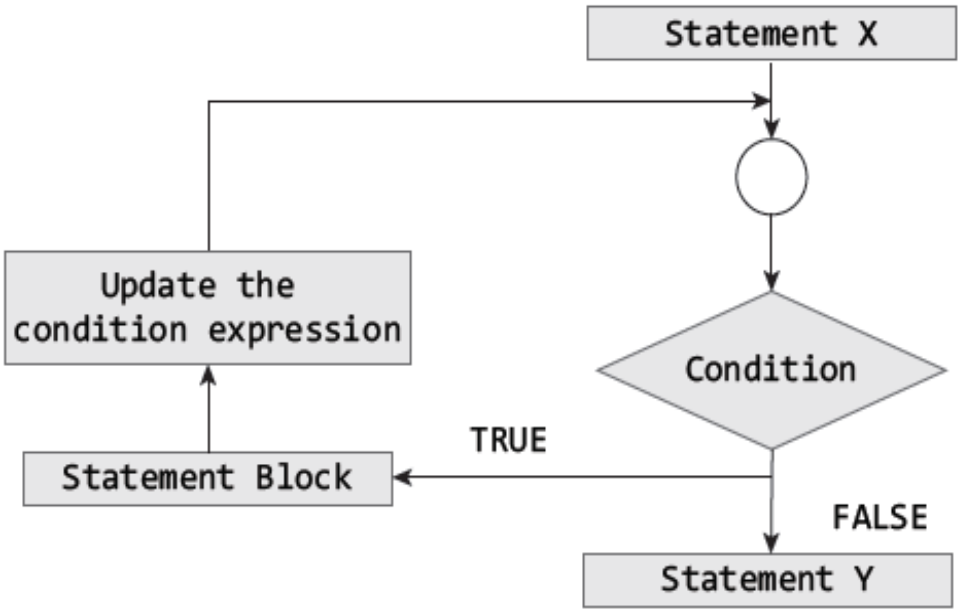
The general form of while loop statement is given as:

```
SYNTAX OF WHILE LOOP
statement x
while (condition):
            statement block
statement y
```

In while loop, the condition is tested before any statements are executed. Set of statements inside the loop are executed only when the condition is true. If it is false, they are not executed.

The while loop is a top checking loop as the control condition is placed as the first line of code.

The flow diagram of while loop statement is as follows:



Example 1: Program to print first 'n' Natural numbers: (**Observation)**

```
n = int(input('Enter the value of N:'))
i = 1
while (i<=n):
    print (i)
    i = i + 1
```

```
Enter the value of N:10
1
2
3
4
5
6
7
8
9
10
```

Example 2: Program to print the sum and average of first 'n' natural numbers: (**Observation)**

```
n = int(input('Enter the value of N:'))
i = 1
sum = 0
while (i<=n):
    sum = sum + i
```

```
        i = i + 1
avg = sum / n
print ("The sum of %d numbers is: %d" %(n,sum))
print ("The average of %d numbers is: %f" %(n,avg))
```

```
Enter the value of N:10
The sum of 10 numbers is: 55
The average of 10 numbers is: 5.500000
```

Example 3: Program to print the sum of digits of a number: (**Observation)**

```
n = int(input ("Enter the number: "))
sum = 0
while(n!=0):
        rem = n % 10
        sum = sum+rem
        n = int(n / 10)
print("\n The Sum of the digits is: ",sum)
```

```
Enter the number: 2365

 The Sum of the digits is:  16
```

FOR LOOP:

For loop provides a mechanism to repeat a task until a particular condition is True.

It is usually known as a **determinate or definite loop** because the programmer knows exactly how many times the loop will repeat.

The ***for...in*** statement is a looping statement used in Python to iterate over a sequence of objects.

The general form of for statement is:

```
Syntax of for Loop

for loop_contol_var in sequence:
   statement block
```

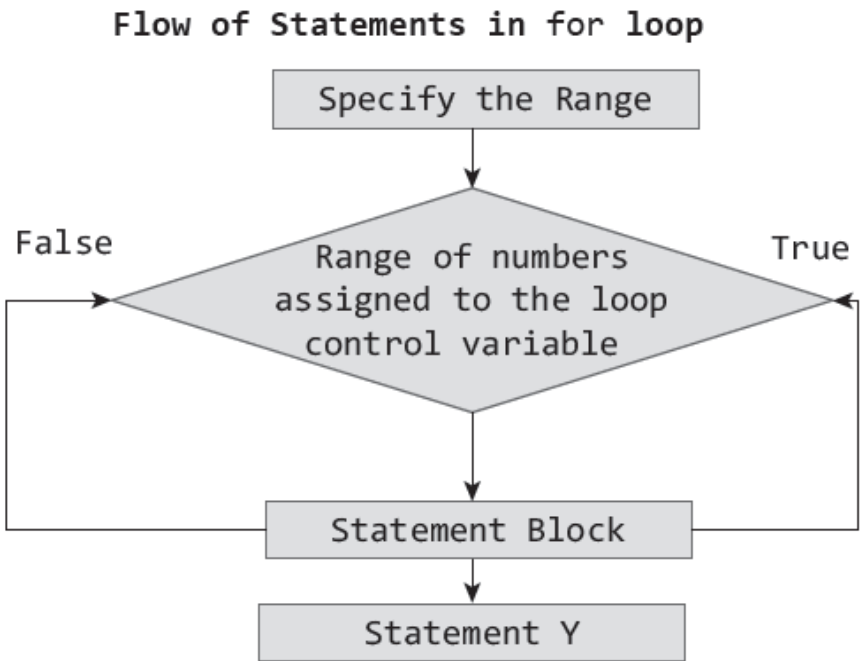
In for loop, the condition is tested before any statements are executed.

The for loop is executed for each item in the sequence.

The loop control variable is incremented / decremented automatically and it is checked every time before entering the loop.

Loop control variable is not initialized or incremented / decremented like while loop.

The flow diagram of for loop statement is as follows:



Example 1: Program to print first 'n' Natural numbers using for loop:

```
n = int(input('Enter the value of N:'))
for i in range (1,n+1):
        print (i)
```

```
Enter the value of N:10
1
2
3
4
5
6
7
8
9
10
```

Example 2: Program to print Multiplication table using for loop: (**Observation)**

```
n = int(input ("Enter the multiplication table series: "))
print("The Multiplication Table is as follows:")
```

```
    for i in range(1,11):
        print ("%d * %d = %d" % (i,n,i*n))
```

```
Enter the multiplication table series: 7
The Multiplication Table is as follows:
1 * 7 = 7
2 * 7 = 14
3 * 7 = 21
4 * 7 = 28
5 * 7 = 35
6 * 7 = 42
7 * 7 = 49
8 * 7 = 56
9 * 7 = 63
10 * 7 = 70
```

Example 3: Program to print the list of odd numbers and even numbers using for loop: (**Observation)**

```
n = int(input ("Enter the number limit: "))
print("The odd numbers within %d are:" % (n))
for i in range(2,n,2):
    print (i)
print("The even numbers within %d are:" % (n))
for i in range(1,n,2):
    print (i)
```

```
Enter the number limit: 20
The odd numbers within 20 are:
2
4
6
8
10
12
14
16
18
The even numbers within 20 are:
1
3
5
7
9
11
13
15
17
19
```

range( ) Function:

range( ) is a built-in function that is used to iterate over a sequence of numbers. The syntax of range( ) is

range (beg, end, [step])

The range( ) function produces a sequence of numbers starting from ***beg*** and ending with ***end – 1***.

The **step** argument is optional (that is why it is placed in brackets).

By default, every number in the range is incremented by 1.

Step can either hold positive or negative value, but it cannot be equal to zero.

If range() function is given a single argument, it produces an object with values from 0 to argument-1. For example:

range (10) is equal to writing range(0, 10)

If range() is called with two arguments, it produces values from the first to the second - 1. For example:

range(0,10) gives values from 0 to 9

If range() has three arguments then the third argument specifies the interval of the sequence produced. In this case, the third argument must be an integer. For example:

range(1,20,3) gives values as 1,4,7,10,13,16,19

NESTED LOOP:

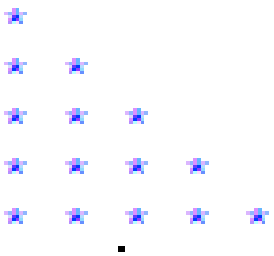
In Python loops that can be placed inside other loops. This feature is applicable in both *for* and *while* loops but it is most preferred in for loop.

For example: A for loop can be used to control the number of times a particular set of statements will be executed. Another outer loop could be used to control the number of times that a whole loop is repeated.

Loops should be properly indented to identify which statements are contained within each **for** statement.

Example 1: Program to print the following pattern using nested loop: **(Observation)**

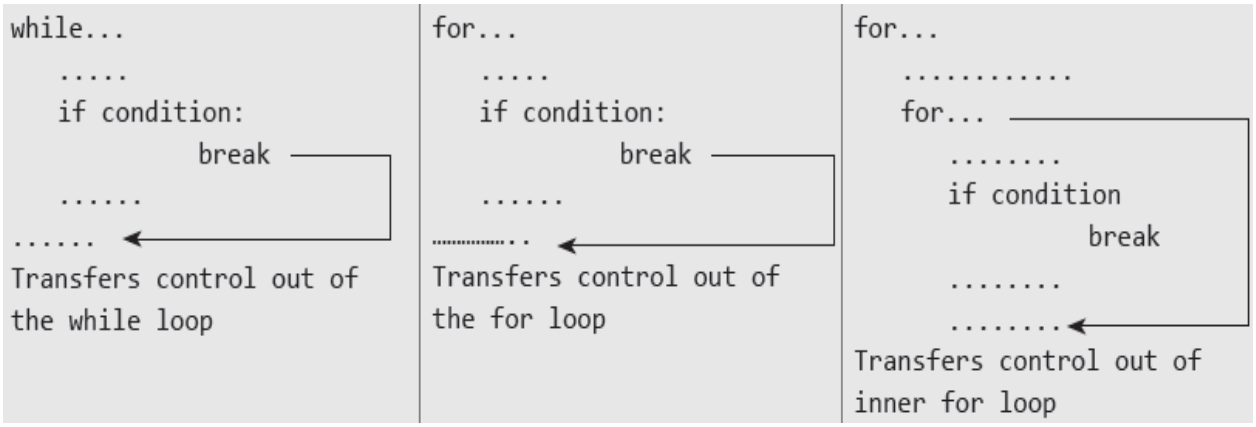
```
n=6
for i in range(0,n):
        print(" ")
        for j in range (0,i):
                print('*', end=' ')
```



BREAK STATEMENT:

The break statement is used to terminate the execution of the nearest enclosing loop in which it appears.

Break statement is widely used with for loop and while loop. When compiler encounters a break statement, the control passes to the statement that follows the loop in which the break statement appears.



Example:

```
i = 1
while i <= 10:
        print(i, end=" ")
        if i==5:
                break
        i = i+1
print("\n Done")
```

OUTPUT

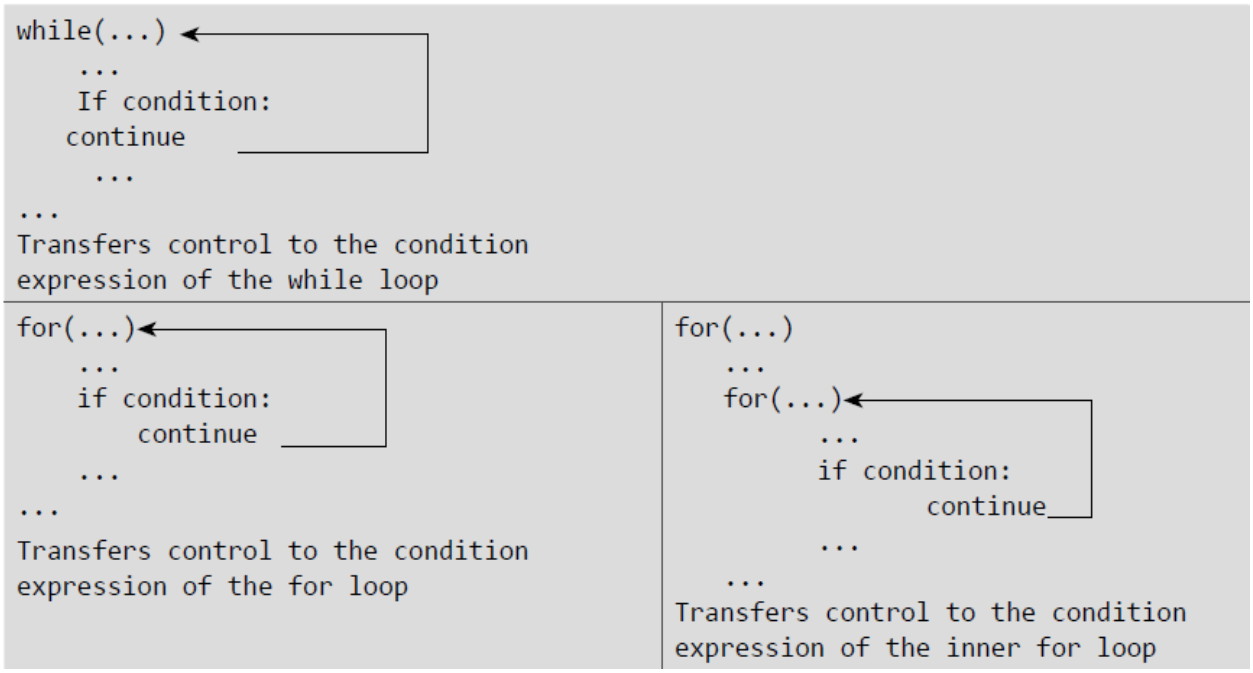
```
1 2 3 4 5
Done
```

CONTINUE STATEMENT:

Like the break statement, the continue statement can only appear in the body of a loop.

When the compiler encounters a continue statement, then the rest of the statements in the loop are skipped and the control is unconditionally transferred to the loop-continuation portion of the nearest enclosing loop.

The continue statement is used to stop the current iteration of the loop and continues with the next one.



Example:

```
for i in range(1,11):
    if(i==5):
        continue
    print(i, end=" ")
print("\n Done")
```

OUTPUT

```
1 2 3 4 6 7 8 9 10
Done
```

PASS STATEMENT:

Pass statement is used when a statement is required syntactically but no command or code has to be executed.

It specified a null operation or simply No Operation (NOP) statement, i.e: Nothing happens when the pass statement is executed.

Difference between comment and pass statements is while the interpreter ignores a comment entirely, pass is not ignored. Comment is not executed but pass statement is executed but nothing happens.

```
for letter in "HELLO":
        pass     #The statement is doing nothing
        print("Pass : ", letter)
print("Done")
```

OUTPUT

```
Pass :  H
Pass :  E
Pass :  L
Pass :  L
Pass :  O
Done
```

ITERATING STRINGS:

String can be iterated using for loop and while loop. Example:

```
str = "Welcome to Python"
for i in str:
print(i, end=' ')
```

OUTPUT

```
W e l c o m e   t o   P y t h o n
```

```
message = " Welcome to Python "
index = 0
while index < len(message):
   letter = message[index]
   print(letter, end=' ')
   index += 1
```

OUTPUT

```
W e l c o m e   t o   P y t h o n
```

ILLUSTRATIVE PROGRAMS:

1. LINEAR SEARCH: Observation

Program:

```
n = int(input("Enter the number of numbers in the list: "))
a = []
for i in range (0,n):
    x = int(input("Enter the element: "))
    a.append(x)
x = int(input("Enter the number to search: "))
f = 0
for i in range (0,n):
    if(a[i] == x):
        f = 1
        break
if(f == 1):
    print("%d is found in the list" % x)
else:
    print("%d is not found in the list" % x)
```

Output:

```
Enter the number of numbers in the list: 10
Enter the element: 25
Enter the element: 62
Enter the element: 3
Enter the element: 21
Enter the element: 41
Enter the element: 89
Enter the element: 842
Enter the element: 20
Enter the element: 18
Enter the element: 35
Enter the number to search: 41
41 is found in the list
```

2. BINARY SEARCH: Observation

Program:

```
n = int(input("Enter the number of numbers in the list: "))
a = []
for i in range (0,n):
    x = int(input("Enter the element: "))
    a.append(x)
x = int(input("Enter the number to search: "))
first = 0
last = n
f = 0
while ((first<=last) and (f==0)):
    mid = (first + last)//2
    if (a[mid] == x):
        print ("%d is found in the list" % x)
        f = 1
        break
    else:
```

```
            if x < a[mid]:
                last = mid - 1
            else:
                first = mid + 1
    if(f != 1):
        print("%d is not found in the list" % x)
```

Output:

```
Enter the number of numbers in the list: 6
Enter the element: 12
Enter the element: 56
Enter the element: 89
Enter the element: 95
Enter the element: 51
Enter the element: 62
Enter the number to search: 84
84 is not found in the list
```

Observation

1. Write a Program to print first ‘n’ Natural numbers:
2. Program to print the sum and average of first ‘n’ natural numbers:
3. Program to print the sum of digits of a number
4. Program to print Multiplication table using for loop:
5. Program to print the list of odd numbers and even numbers using for loop:
6. Program to print the following pattern using nested loop:

7. One example for break statement
8. One example for continue statement

9. One example for iterating string
10. Linear search
11. Binary search

Record

1. **Write a Python program to print those numbers which are divisible by 3 and 5, between 1000 and 2000 (both included).**
2. **Sum of odd and even numbers for the first N natural numbers.**
3. **Write a Python program to get the Fibonacci series between 0 to 50.**
4. **Reverse a Number and then check for Palindrome or not**
5. **If we list all the natural numbers below 10 that are multiples of 3 or 5, we get 3, 5, 6 and 9. The sum of these multiples is 23. Find the sum of all the multiples of 3 or 5 below 100.**
6. **Generate Prime numbers between 1 and 100.**