Q. what is the use of range()?

---->> range() method is used to generate a sequence of numbers.

--->> range() method is accepting 3 parameters.

syntax : range(start_value, stope_value, step_value)

---->> default value for start_value is 0

---->> default value for step_value is 1

---->> stope_value is must be user-defined value.stop_value is always n-1.

For example,

range(10) --------->> range(0,10,1) ----->> 0,1,2,3,4,5,6,7,8,9

range(5,10)------->> range(5,10,1) ----->> 5,6,7,8,9

range(3,10,2)----->> range(3,10,2) ----->> 3,5,7,9

range(10,5,-1)---->> range(10,5,-1)----->> 10,9,8,7,6

Example:

```
>>> for i in range(10):
        print(i)
```

output

```
0
1
2
3
4
5
6
7
8
9
>>>
```

---->> to print output in same line (horizontal) format then use end=' ' attribute inside print() method.

```
>>> for i in range(10):
```

```
        print(i,end=' ')
```

output

0 1 2 3 4 5 6 7 8 9

Loops or Iterative Statements

If you want to execute the certain statements multiple times then we have to use looping statements.

for loops :-

For loop is a programming language statement, i.e. an iteration statement, which allows a code block to be repeated a certain number of times.

syntax :

```
for item in sequence_object :
        print(item)
```

styntax :

```
for variable in sequence:
        statement1
        statement2
        statement3
```

For examples:

```
st = "Python Developer"
for i in st:
   print(i)

lst = [1,2,3.5,"Python",4+5j,True]
for i in lst:
   print(i)

tup = (1,2,3,True,False,"Srinivas")
for  i  in  tup:
      print(i)

se = {2,3,'Python',0,0,True,2+6j,'Srinivas'}
for  I   in  se:
      print(i)
```

```
dic = {1:'a',2:'b','c':45}
for   i  in dic:
      print(i)
```

Q. Write a program to print the 10th table ?

```
for   I   in  range(1,11):
        r = 10 * i
        print(10 , '*', i, '=', r)
```

Output

```
10 * 1 = 10
10 * 2 = 20
10 * 3 = 30
10 * 4 = 40
10 * 5 = 50
10 * 6 = 60
10 * 7 = 70
10 * 8 = 80
10 * 9 = 90
10 * 10 = 100
>>>
```

Q. WAP to for-loop to generate 10 to 1 numbers ?

```
for i in range(10,0,-1):
       print(i, end=' ')
print('Thank you')
```

Q. WAP to display 1 to n even numbers?

Q. WAP to display sum of 1 to n numbers by using for-loop ?

```
n = int(input('Enter any number :'))
sum = 0
for i in range(1, n+1):
   sum = sum+i
print(sum)
```

output

Enter any number :4

10

While - loop

- while loop is used to execute certain statements multiple times.
- in while loop first it checks the condition, if it is true then it will execute the statements. This process is continueous until while loop condition becomes false.

Syntax:

```
while (condition):
   statement1
   statement2
   statement3
   statement4
```

Example:

```
num = 1
while(num <= 5):
    print("the count is: ",num)
    num +=  1  #  num = num + 1
```

output

```
the count is:  1
the count is:  2
the count is:  3
the count is:  4
the count is:  5
```

Note: In while loop always increament|decreament statements must be required for condition becoming false. Otherwise while loop condition not going to be false. So loop not teriminated. It prints infinite times.

Example

```
num=1
while(num <= 5):
     print("the count is: ",num)
```

---->>> Here condition always True.so infinite times executed condition.

Q) WAP to display 0 to n number Square numbers ?

```
num = int(input('enter any number :'))
i = 0
while ( i < num+1 ):
   print('Square of '+str(i) +' is '+str(i *2))
   i += 1
```

Output:

```
enter any number :10
Square of 0 is 0
Square of 1 is 1
Square of 2 is 4
Square of 3 is 9
Square of 4 is 16
Square of 5 is 25
Square of 6 is 36
Square of 7 is 49
Square of 8 is 64
Square of 9 is 81
Square of 10 is 100
```

Q. WAP to perform sum of digits of a given number ? (123---> 1+2+3 -->> 6)

```
n = int(input('enter any number :'))
sum = 0
while(n!=0):
   r = n%10
   n = n//10
   sum = sum + r
print('The sum of given digit is :',sum)
```

output

```
enter any number :123
The sum of given digit is :6
```

Nested loops:

A loop inside of another loop is called as nested-loop.

Example:

```
for i in range(5):
    for j in range(i+1):
        print(j,end=' ')
    print()
```

output

```
0
0 1
0 1 2
0 1 2 3
0 1 2 3 4
```

Transfer Statements

Transfer statements are used to transfer the program control from one location to another location.

We have different types of transfer statements,

1. break
2. continue
3. pass
4. return
5. assert

1. break

---->> break is a keyword which is used only in looping statements.

---->> when ever break occurs it will stop entire iteration and control goes outside the loop.

2. continue

---->> continue is a keyword which is used only in looping statements.

---->> when ever continue occurs it will stop current iteration and executes from next iteration onwords.

For example:

```
for  i in range(1,10):
   if(i % 2 == 0):
       continue
   if(i % 5 == 0):
      break
   print(i,end=' ')
```

output

```
1 3
```

3. pass

--->> pass is a keyword which is used in program at the time of partial development of code.

Example:

```
if 10 > 5:
     pass
```

4. return

---->> return is a keyword which is used in functions concept.

---->> if you want to transfer any value in functions then we have to use return keyword.

---->> in functions we can return single value as well as multiple value.

Example:

```
def sum():
       a = 10;
        b = 20;
         c = a + b
          print(c)
          return None
```

```
result = sum()
print("sum is:", result  )
```

output:

```
30
sum is: None
```

5. assert

---->> Assert is a keyword which is used to generate exceptions.

---->> If you want to execute certain statements only when the condition is satisfied otherwise program has to stop with error message.

Example:

```
n = int(input("Enter any number : "))
assert n > 0, "Invalid number"
print("Your number is accepted")
print("We will process your request ")
```

Example:

---->> The following example uses the assert statement in the function.

```
def square(x):
    assert x>=0, 'Only positive numbers are allowed'
    return x*x
n = square(2) # returns 4
n = square(-2) # raise an AssertionError
```

Output

```
Traceback (most recent call last):
    assert x > 0, 'Only positive numbers are allowed'
AssertionError: Only positive numbers are allowed
```

for-else :

In for-else, a for loop is executed successfully then only else block statements are executed otherwise else block statements will not executed.

For example:

```
for i in range(1,5):
        if(i%5==0):
                break
        print(i,end=' ')
else:
   print("working completed")
print('ok')
```

output:

```
1 2 3 4 working completed
ok
```