Python Looping Statements (for, while)

The range() Function

- With the help of the range() function, we may produce a series of numbers. range(10) will produce values between 0 and 9. (10 numbers).
- range() method is used to generate a sequence of numbers.
- range() method is accepting 3 parameters. We can give specific start, stop, and step size values as a input to the range() function.

  Syntax: range(start_value, stop_value, step_size).
- If the start value is not specified, it defaults to 0
- If the step size is not specified, it defaults to 1.
- stope_value is must be user-defined value. stop_value is always exclusive like n-1.

Examples: # Python program to show the working of range() function

```
print(range(15))
print(list(range(15)))
print(list(range(4, 9)))
print(list(range(5, 25, 4)))
```

Output:

```
range(0, 10)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
[4, 5, 6, 7, 8]
[5, 9, 13, 17, 21]
```

Explanation:

range(10) converts into range(0,10,1). So generates 0,1,2,3,4,5,6,7,8,9 numbers
Here by default start-value is 0 and step-size is 1. Given value is stop-value.

range(5, 10)------->> range(5, 10, 1) ----->> So generates 5, 6, 7, 8, 9 numbers
Here start-value is 5 and stope-value is 10 and step-value is 1

range(3, 10, 2)----->> range(3,10,2) ----->> So generates 3, 5, 7, 9 numbers
Here start-value is 3 , stop-value is 10 and step-value is 2. It means every 2nd element generates.

range(10, 5, -1)---->> range(10, 5, -1)----->> 10, 9, 8, 7, 6
Here start-value is 10, stop-value is 5 and step-value is -1. It means from 10 to 6 values generates & each time -1 value is decreament.

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.

Python loops are fundamental programming constructs that allow you to execute a block of code repeatedly, based on a condition or over a sequence of elements.

They are widely used in various scenarios for tasks such as iterating over data, automating repetitive actions, and managing control flow in programs.

Python provides two main types of loops: They are

1. for – loops
2. while – loops

For loops Concept:-

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.

for loop used for iterating over a sequence (like lists, tuples, strings, or ranges) of items.

Syntax : **for item in sequence_object :**
{ code block }

In this case, the variable item is used to hold the value of every item present in the sequence before the iteration begins until this particular iteration is completed.

Loop iterates until the final item of the sequence are reached.

Example1: Creating a string sequence and iterating each charecter

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

Example2: Creating a List sequence and iterating each item from this

```
lst = [1,2,3.5,"Python",4+5j,True]
```

```
for i in lst:
   print(i)
```

Example3: Creating a Tuple sequence and iterating each item from this

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

Example4: Creating a Set object and iterating each item from this

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

Example5: Creating a Dictionary object and iterating each item from this

```
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. Write a for-loop program to generate 10 to 1 numbers?

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

Q. Write a for-loop program to display 1 to n even numbers?

Q. Write a for-loop program to display sum of 1 to n numbers?

```
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 Concepts

- 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 continuous 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. Write a program 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

Python Nested Loops:

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

Code:

```
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 like

1. break
2. continue
3. pass

1. break

- **break** is a keyword which is used only in looping statements.
- when ever break keyword occurs it will stop entire iteration and control goes outside the loop.

For example:

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

Output:

1

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

Output:

1 3 5 7 9

Working with both *continue* & *break* keyword statements

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

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

Output:

```
1  3
```

3. pass

- The **pass** statement is used as a placeholder for future code. When the pass statement is executed, nothing happens, but you avoid getting an error when empty code is not allowed.
- Empty code is not allowed in loops, function definitions, class definitions, or in if - else statements.
- **pass** is a keyword which is used in program at the time of partial development of code blocks.

Example1:

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

Example2:

```
for  i  in range(5):
     pass
```

Example3:

```
while condition:
     pass
```

Example4:

```
def  addition(a,b):
     pass
```

Using else Statement with for Loop

As already said, a for loop executes the code block until the sequence element is reached end. The statement is written right after the for loop is executed after the execution of the for loop is complete.

Only if the execution is complete does the else statement comes into play. It won't be executed if we exit the loop or if an error is thrown.

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

Syntax:

```
for value in sequence:
    # executes the statements until sequences are completed
else:
    # executes these statements when for loop is completed
```

Example1:

```
for i in range(1, 5):
   if(i%3 == 0):
      break
   print(i, end=' ')
else:
   print("for loop iterations completed")
print('ok')
```

Output:

```
1 2 ok
```

Example2:

```
for i in range(1, 5):
   if(i%3 == 0):
      continue
   print(i, end=' ')
else:
   print("for loop iterations completed")

print('ok')
```

Output:

```
1 2 4 for loop iterations completed
```

ok

Using else Statement with while Loops

As discussed earlier in the for loop section, we can use the else statement with the while loop also. It has the same syntax.

Code: **Python program to show how to use else statement with the while loop**

```
counter = 0
# Iterating through the while loop
while (counter < 10):
    counter = counter + 3
    print("Python Loops")    # Executed untile condition is met
# Once the condition of while loop gives False this else statement will be executed
else:
    print("Code block inside the else statement")
```

Output:

```
Python Loops
Python Loops
Python Loops
Python Loops
Code block inside the else statement
```