Loops

A loop statement allows us to execute a statement or group of statements multiple times.

The flow of the programs written in any programming language is sequential by default. Sometimes we may need to alter the flow of the program. The execution of a specific code may need to be repeated several numbers of times.

For this purpose, The programming languages provide various types of loops which are capable of repeating some specific code several numbers of times.

Advantages of loops

1. Using loops, we do not need to write the same code again and again.
2. Using loops, we can traverse over the elements of data structures (array or linked lists).

Iterating string using for loop

```
str = "Python"
for i in str:
   print(i)
```

The range() function with For Loop:

The **range()** function is used to generate the sequence of the numbers. If we pass the range(10), it will generate the numbers from 0 to 9. The syntax of the range() function is given below.

Syntax:

1. range(start,stop,step size)

Example 1:

```
# for with range(start)
for i in range(11):
   print(i)
```

Example 2:

```
# for with range(start, end)
for i in range(1,11):
   print(i)
```

Example 3:

```
# for with range(start,end,step)
for i in range(1,11,2):
   print(i)
```

Example 4:

```
# Program to print table of given number.
n = int(input("Enter the number :"))
for i in range(1,11):
    c = n*i
    print(n,"*",i,"=",c)
```

Nested for loop

Example: 1

```
for i in range(6):
   for j in range(3):
      print(i,"and",j)
      pass
   pass
```

Example 2:

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

```
   pass
```

Example 3:

```
rows = int(input("Enter the rows: "))
for i in range(0,rows+1):
    for j in range(i):
        print(i,end = '')
    print()
```

Example 4:

```
for i in range(0,5):
   print(i)
else:
    print("ELSE block executes when for is completed without any break")
```

Example 5:

```
# for loop with  a condition
for i in range(1,11):
   if(i == 6):
      break
   print(i)
```

Example 6:

```
for i in range(1,6):
   print(i)
   #break;
else:
   print("Statement of else block");
print("The loop is broken due to break statement...came out of the loop")
```

Example 7:

```
# for with continue
for i in range(1,11):
   if(i == 6):
      continue
   print(i)
```

Example 8:

```
# for loop to print negative values
for i in range(-10,11):
   print(i,' ',end='')
```

While loop

The Python while loop allows a part of the code to be executed until the given condition returns false. It is also known as a pre-tested loop.

It can be viewed as a repeating if statement. When we don't know the number of iterations then the while loop is most effective to use.

Example 1:

```
# Prog to print 1 to 10 numbers using while loop
i = 1
while(i<=10):
   print(i)
   i = i+1
   pass
```

Example 2:

```
# Prog for nested while loop
i = 1
while(i<=3):
   j = 1
   while(j<=3):
      print(i,'and',j)
      j = j+1
      pass
```

```
   i = i+1
   pass
```

Example 3:

```
# While with break statement
i = 1
while(i<=10):
   if(i == 6):
      break
   print(i)
   i=i+1
```

Example 4:

```
# nested for with while

for i in range(1, 6):
   j = 1
   while(j<=i):
      print(i,'and',j)
      j=j+1
      pass
   pass
```

Example 5:

```
# Infinite while loop
# We can use while(True) instead of while(1)
i = 1
while(1):
   print("Infinite while loop: ",i)
   i = i+1
```

Example 6:

```
# Program to display the Fibonacci sequence up to n-th term

nterms = int(input("How many terms? "))
```

```
# Fibonacci series: 0 1 1 2 3 5… so on
# first two terms
n1, n2 = 0, 1
count = 0

# check if the number of terms is valid
if nterms <= 0:
   print("Please enter a positive integer")
# if there is only one term, return n1
elif nterms == 1:
   print("Fibonacci sequence upto",nterms,":")
   print(n1)
# generate fibonacci sequence
else:
   print("Fibonacci sequence:")
   while count < nterms:
       print(n1)
       nth = n1 + n2
       # update values
       n1 = n2
       n2 = nth
       count += 1
```