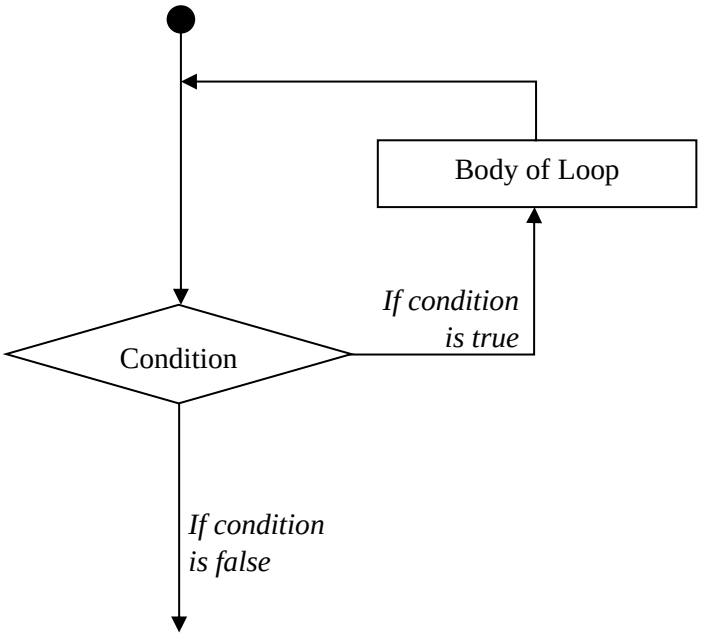
5 - LOOPS

The word **"LOOP"** in general means something which goes on continuously.

1. Why do we need loops in python?

- In general, statements in a code are executed sequentially i.e. the statement written first is executed first, then the second and so on.
- But in some cases we will need to execute a block of code multiple times. In such cases loops come into picture.

The following diagram illustrates a loop statement



- Python programming language provides different types of loops to handle looping requirements.
- While all the ways provide similar basic functionality, they differ in their **syntax**.

2. Types of loops

2.1. For Loop

- '**For loop**' is used when we want to repeat a block of code for fixed number of times.

Syntax in C :-

```
for (initialize_iterator; termination_of_iterator; updation_of_iterator )
{
 Statement 1;
 Statement 2;
 .............
}
```

Now let's see the syntax in python:

Syntax in python:-

```
for <iterator> in <sequence> :
        Statement 1
        Statement 2
        .............
```

There the **"sequence"** can be a list, tuple, strings or range of numbers.

Note: *The detailed description of list, tuple, and string will be covered in upcoming days.*

- If you look at the syntax in C you had to give the following conditions in the loop:

  initialize_iterator: Determine when to *start* the loop.

  termination_of_iterator: Determine when to *end* the loop.

  updation_of_iterator: Determine whether to *increment/ decrement* the loop.

- But in python you have an inbuilt function named range ( ) in which you can give the above conditions.
- There are 3 ways to write the range function:

i. range(**stop**)
ii. range(**start, stop**)
iii. range(**start, stop, step**)

- Let us see an example for each of the type

i. range(**stop**)

Gives a sequence of numbers which start from **'0',** end on **one number before the stop value** given in range function**,** and updates by **'1'.**

Example 1:

```
In [3]: for i in range(5):
            print(i)
```

Output:

```
0
1
2
3
4
```

Note: *In this case the sequence always starts from **'0'** and increments by **'1'.***

Example 2:

Print your name 5 times.

```
In [12]: for i in range(5):
             print("Python.")
```

Output:

```
Python.
Python.
Python.
Python.
Python.
```

ii. range(**start, stop**)

Gives a sequence of numbers which start from **'start value',** end on **one number before the stop value** given in range function**,** and updates by **'1'.**

Example 1:

```
In [5]: for i in range(5,10):
            print(i)
```

Output:

```
5
6
7
8
9
```

Note: *In this case, the numbers will always increment by **'1'.***

iii. range(**start, stop, step**)

Gives a sequence of numbers which start from **'start value',** ends **before the stop value** given in range function**,** and updates by **'1'.**

Example1:

```
In [6]: for i in range(10,20,2):
            print(i)
```

Output:

```
10
12
14
16
18
```

Note: *The above sequence stopped at 18 and not on 20. It is because the sequence given by the range function* ***does not include the values which are equal or greater than "stop value".***

Example 2:

Print numbers from 1 to 5 in descending order.

```
In [10]: for i in range(5,0,-1):
             print(i)
```

Output:

```
5
4
3
2
1
```

Here, **start value = 5, end value = 0** (& not 1 because the sequence stops before the end value), **step value = -1** (because we wanted to decrement the sequence by 1).

We will see more examples in the jupyter notebook.

2.2. While Loop

- '**While loop**' is another type of loop, which is also used to execute a block of codes multiple times.
- It is mainly used where the **number of repetition of code is unknown**.
- It executes a block of statements repeatedly until a given a condition is satisfied/true.

Syntax:

```
while <expression>:
        Statement 1
        Statement 2
      .....................
```

Let us see how to use while loop through some examples.

Example 1:

```
In [14]: # initialize counter to '0'
         count = 0

         # loop until counter is less than 4
         while (count < 2):
                     print("Summer Training")
                     # increment the counter by '1'
                     count = count + 1

         #Print after termination of while loop
         print('Out of the Loop')
```

Output:

```
Summer Training
Summer Training
Out of the Loop
```

Explanation:

1. Initially **count = 0**.
2. The condition in while loop is checked, i.e check if '0 < 2'. As the condition is true, the statements inside the while loop will be executed as below.

    Iteration 1: For count = 0

    Print the statement: **Summer Training**

    Increment the value of count by 1. Now **count = 1**.

    Now check if '1 < 2'. As the condition is true, statements inside the while loop will be executed.

    Iteration 2: For count = 1

    Print the statement: **Summer Training**

    Increment the value of count by 1. Now **count = 2**.

    Now check if '2 < 2'. As the condition is false, statements inside the while loop will **not** be executed.

3. Exit the while loop.
4. Print the statement: **Out of the Loop**

This example was similar to for loop.

Now let us see an example for which for loop is made i.e. an example in which the number of repetition is not known.

Example 2:

Input a number between 0 - 3 from the user. Go on incrementing the input till it 5.

In [25]:
```
# initialization of 'count'
count = int(input("Enter a number between 0 to 3 : "))

# loop until 'count = 5'
while (count < 5):
        print(count)
        # increment value of count by '1'
        count = count+1
```

Output:

```
Enter a number between 0 to 3 : 2
3
4
5
```

Explanation:

So here if you see, the user can input any number between 0 to 3. So for different inputs the number of iterations will be different.

Input by user	Number of iterations	Output
0	5	0 1 2 3 4
1	4	1 2 3 4
2	3	1 2 3
3	2	1 2

So in this case the number of iteration is not fixed. Thus in such cases we can use while loops.

2.3. Nested Loops:

A **nested loop** is a loop that occurs within another loop, structurally similar to nested if statements.

Types of nested loops are:

- Nested *for loops*
- Nested *while loops*
- *for loop inside while loop*
- *while loop inside for loop*

Let us look as nested for loops:

When one for loop is placed inside another for loop it is called nested for loop.

Syntax for nested for loop-:

for < **iterator_1** > in < **sequence_1** > **:**

for < **iterator_2** > in < **sequence** > **:**

statements(s)

statements(s)

.....

So, let us go through some examples.

Example:

```
In [37]: for num1 in range(2):
             for num2 in range(2):
                 print("num1 = ",num1, "num2 = ", num2)
```

Output:

```
num1 =  0 num2 =  0
num1 =  0 num2 =  1
num1 =  1 num2 =  0
num1 =  1 num2 =  1
```

Explanation:

In nested for loop, for each iteration of the outer loop, the inner loop will be re-started.

The inner loop must finish all of its iterations before the outer loop can continue to its next iteration as shown in the table below..

*Iteration **of outer loop***	*Iteration of **inner loop***	*Output*
0	*0*	*num1 = 0 num2 = 0*
	1	*num1 = 0 num2 = 1*
1	*0*	*num1 = 1 num2 = 0*
	1	*num1 = 1 num2 = 1*

Similarly codes can be written for other types of nested loops.

3. Break and continue statement

"**break**" statement is used when we want to stop the loop while the condition is true.
Let us see one example

Example:

```
In [2]: i = 1
        while i < 6:
            print(i)
            if i == 3:
                break
            i += 1
```

Output:

```
1
2
3
```

In this example the loop was supposed to run 5 times and print 1 to 5 values. But the output has only 3 values this is because of the break statement in the loop. When the i = 3 condition is satisfied the break statement terminates the loop.

Now let us see what is a continue statement.

With the **continue statement** we can stop the current iteration, and continue with the next

Let us look at one example.

Example:

```
In [3]: i = 0
        while i < 6:
            i += 1
            if i == 3:
                continue
            print(i)
```

Output:

```
1
2
4
5
6
```

In this example the loop is being executed 6 times. If you look at the output, you will see that '**3**' is missing. This is because when i = 3 was satisfied, the continue statement was invoked, which skips the current iteration (i.e. i = 3) and goes to the next iteration.

References:

1. geeksforgeeks.org/loops-in-python
2. https://www.w3schools.com/python/python_while_loops.asp

… Happy Learning ...!