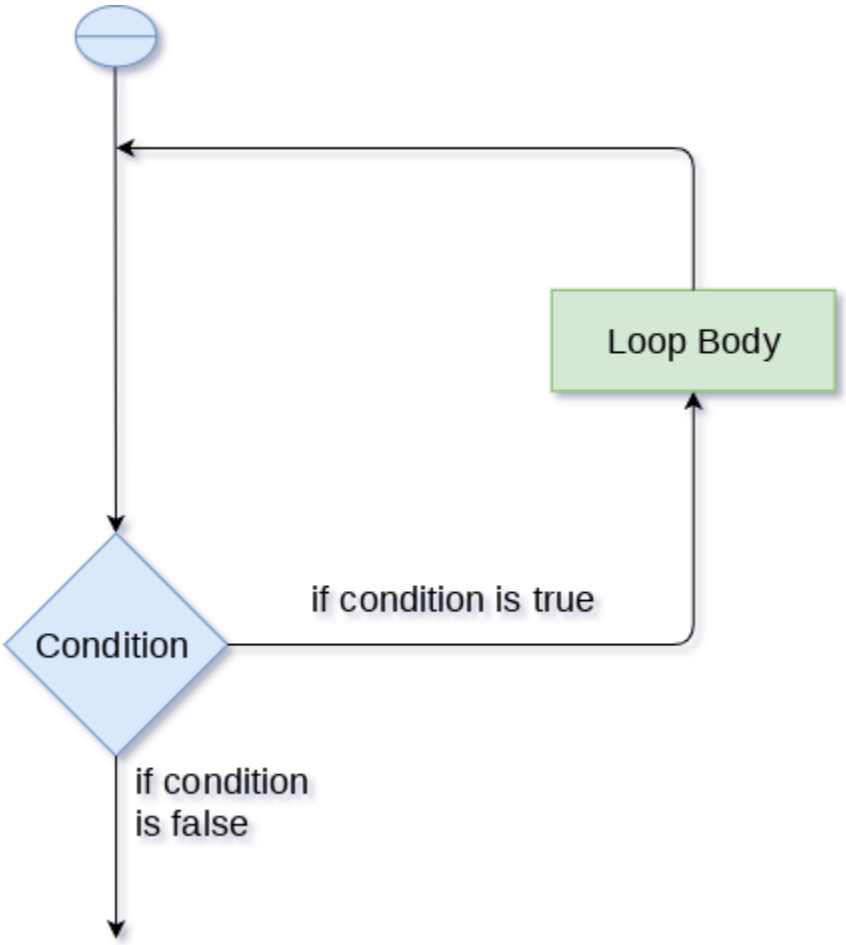
Python Loops

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. Consider the following diagram to understand the working of a loop statement.



Why we use loops in python?

The looping simplifies the complex problems into the easy ones. It enables us to alter the flow of the program so that instead of writing the same code again and again, we can repeat the same code for a finite number of times. For example, if we need to print the first 10 natural numbers then, instead of using the print statement 10 times, we can print inside a loop which runs up to 10 iterations.

Advantages of loops

There are the following advantages of loops in Python.

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

There are the following loop statements in Python.

Loop Statement	Description
for loop	The for loop is used in the case where we need to execute some part of the code until the given condition is satisfied. The for loop is also called as a per-tested loop. It is better to use for loop if the number of iteration is known in advance.
while loop	The while loop is to be used in the scenario where we don't know the number of iterations in advance. The block of statements is executed in the while loop until the condition specified in the while loop is satisfied. It is also called a pre-tested loop.
do-while loop	The do-while loop continues until a given condition satisfies. It is also called post tested loop. It is used when it is necessary to execute the loop at least once (mostly menu driven programs).

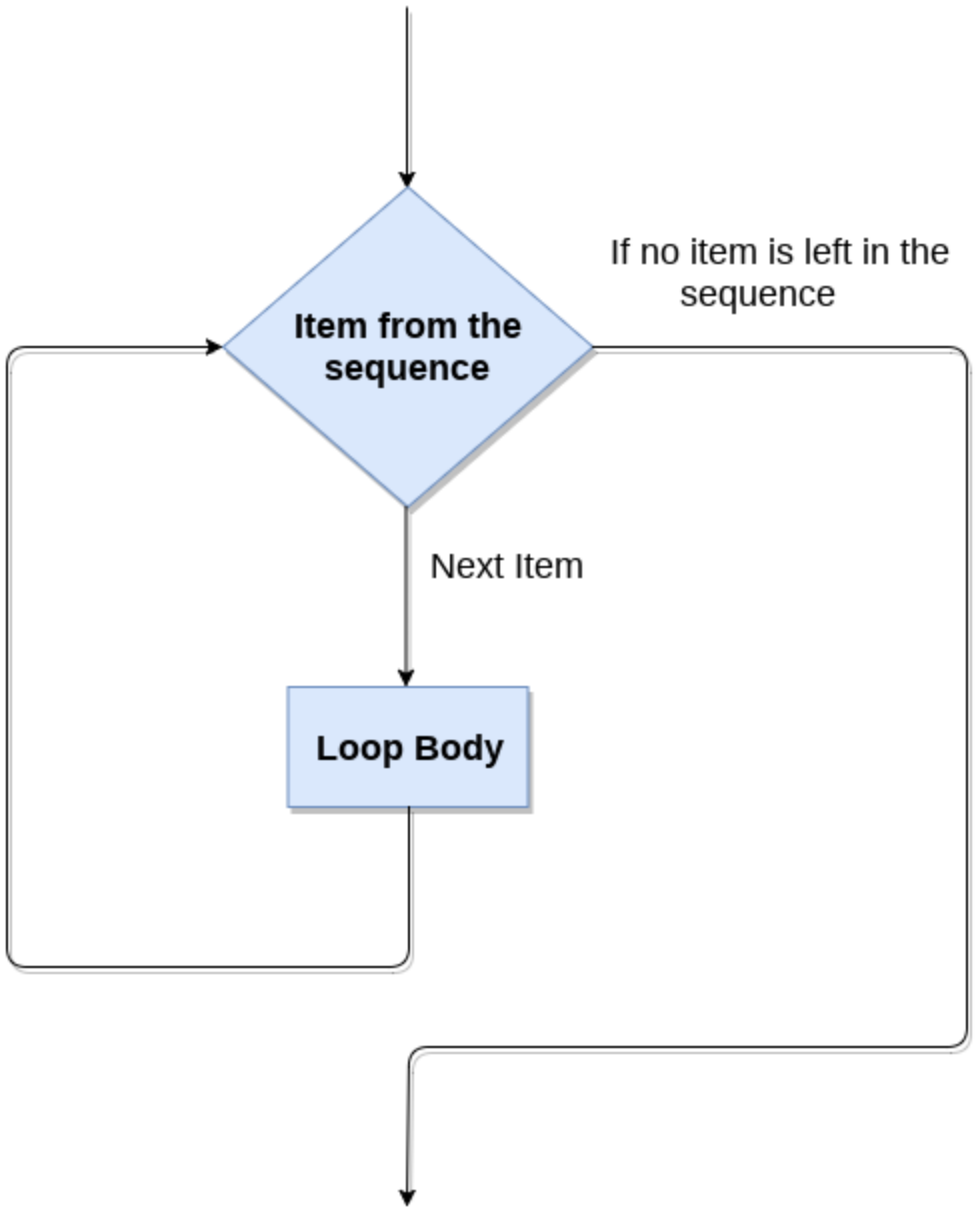
Python for loop

The for **loop in Python** is used to iterate the statements or a part of the program several times. It is frequently used to traverse the data structures like list, tuple, or dictionary.

The syntax of for loop in python is given below.

```
1. for iterating_var in sequence:
2.     statement(s)
```

The for loop flowchart



For loop Using Sequence

Example-1: Iterating string using for loop

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

Output:

```
P
```

```
y
t
h
o
n
```

Example- 2: Program to print the table of the given number .

```
list = [1,2,3,4,5,6,7,8,9,10]
n = 5
for i in list:
    c = n*i
    print(c)
```

Output:

```
5
10
15
20
25
30
35
40
45
50s
```

Example-4: Program to print the sum of the given list.

```
list = [10,30,23,43,65,12]
sum = 0
for i in list:
    sum = sum+i
print("The sum is:",sum)
```

Output:

```
The sum is: 183
```

For loop Using range() function

The range() function

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)
    - The start represents the beginning of the iteration.
    - The stop represents that the loop will iterate till stop-1. The **range(1,5)** will generate numbers 1 to 4 iterations. It is optional.
    - The step size is used to skip the specific numbers from the iteration. It is optional to use. By default, the step size is 1. It is optional.

Consider the following examples:

Example-1: Program to print numbers in sequence.

```
for i in range(10):
    print(i,end = ' ')
```

Output:

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

Example - 2: Program to print table of given number.

```
1. n = int(input("Enter the number "))
2. for i in range(1,11):
3.     c = n*i
4.     print(n,"*",i,"=",c)
```

Output:

```
Enter the number 10
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
```

Example-3: Program to print even number using step size in range().

```
n = int(input("Enter the number "))
for i in range(2,n,2):
    print(i)
```

Output:

```
Enter the number 20
2
4
6
8
10
12
14
16
18
```

We can also use the **range()** function with sequence of numbers. The **len()** function is combined with range() function which iterate through a sequence using indexing. Consider the following example.

```
list = ['Peter','Joseph','Ricky','Devansh']
for i in range(len(list)):
    print("Hello",list[i])
```

Output:

```
Hello Peter
Hello Joseph
Hello Ricky
Hello Devansh
```

Nested for loop in python

Python allows us to nest any number of for loops inside a **for** loop. The inner loop is executed n number of times for every iteration of the outer loop. The syntax is given below.

Syntax

```
1. for iterating_var1 in sequence:  #outer loop
2.     for iterating_var2 in sequence:  #inner loop
3.         #block of statements
4. #Other statements
```

Example- 1: Nested for loop

```
# User input for number of rows
rows = int(input("Enter the rows:"))
# Outer loop will print number of rows
for i in range(0,rows+1):
# Inner loop will print number of Astrisk
    for j in range(i):
        print("*",end = '')
    print()
```

Output:

```
Enter the rows:5
*
**
***
****
*****
```

Example-2: Program to number pyramid.

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

Output:

```
1
22
333
4444
55555
```

Using else statement with for loop

Unlike other languages like C, C++, or Java, Python allows us to use the else statement with the for loop which can be executed only when all the iterations are exhausted. Here, we must notice that if the loop contains any of the break statement then the else statement will not be executed.

Example 1

```
for i in range(0,5):
    print(i)
else:
    print("for loop completely exhausted, since there is no break.")
```

Output:

```
0
1
2
3
4
for loop completely exhausted, since there is no break.
```

The for loop completely exhausted, since there is no break.

Example 2

```
for i in range(0,5):
    print(i)
    break
else:print("for loop is exhausted")
print("The loop is broken due to break statement...came out of the loop")
```

In the above example, the loop is broken due to the break statement; therefore, the else statement will not be executed. The statement present immediate next to else block will be executed.

Output:

```
0
```

The loop is broken due to the break statement...came out of the loop. We will learn more about the break statement in next chapter.

Python 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.

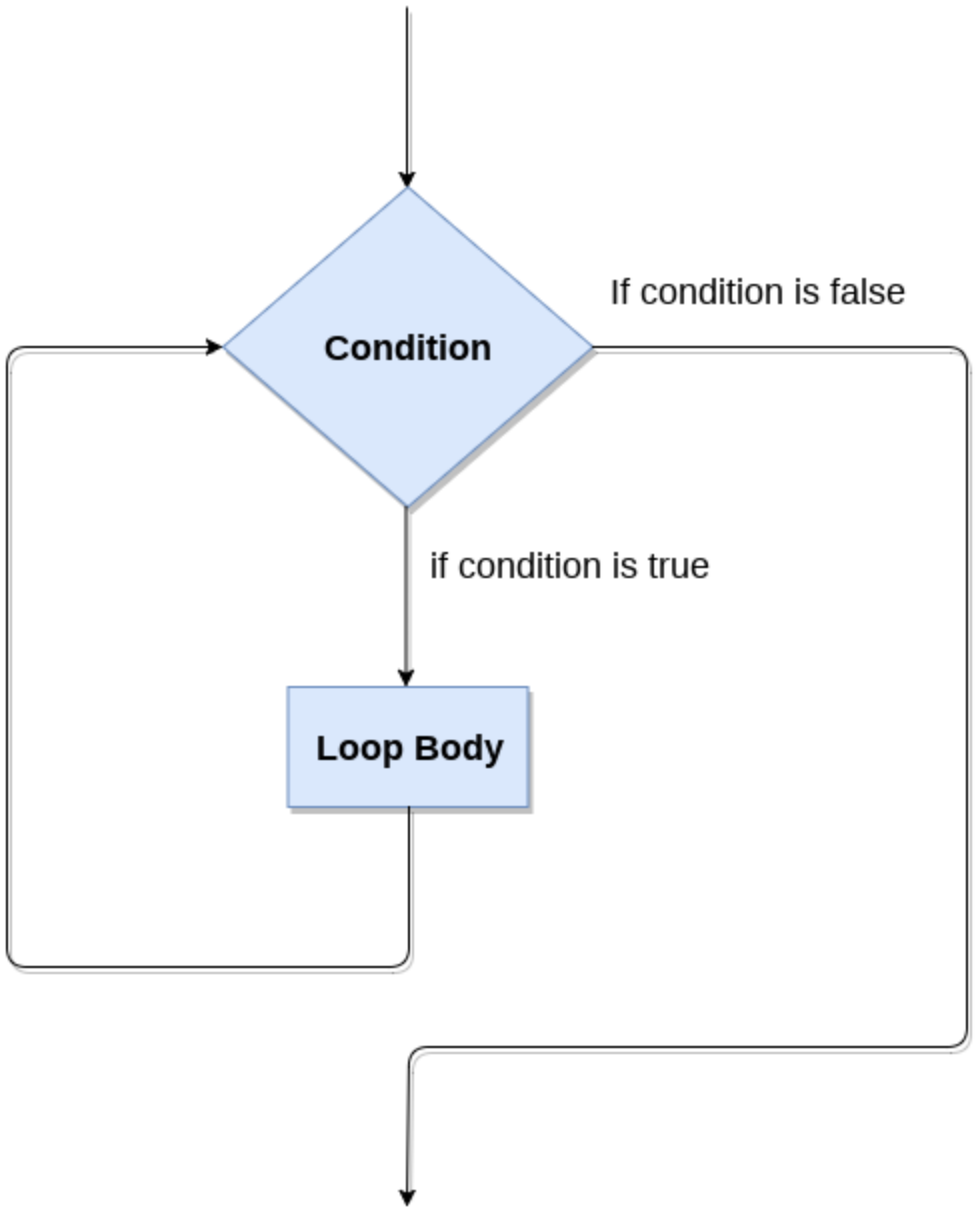
It can be viewed as a repeating if statement. When we don't know the number of iterations then the while loop is most effective to use.

The syntax is given below.

1. **while** expression:
2. statements

Here, the statements can be a single statement or a group of statements. The expression should be any valid Python expression resulting in true or false. The true is any non-zero value and false is 0.

While loop Flowchart



Loop Control Statements

We can change the normal sequence of **while** loop's execution using the loop control statement. When the while loop's execution is completed, all automatic objects defined in that scope are demolished. Python offers the following control statement to use within the while loop.

1. Continue Statement - When the continue statement is encountered, the control transfer to the beginning of the loop. Let's understand the following example.

Example:

```
# prints all letters except 'a' and 't'
i = 0
str1 = 'javaprogrammingpoint'

while i < len(str1):
    if str1[i] == 'a' or str1[i] == 't':
        i += 1
        continue
    print('Current Letter :', str1[i])
    i += 1
```

Output:

```
Current Letter : j
Current Letter : v
Current Letter : p
Current Letter : o
Current Letter : i
Current Letter : n
```

2. Break Statement - When the break statement is encountered, it brings control out of the loop.

Example:

```
# The control transfer is transfered
# when break statement soon it sees t
i = 0
str1 = 'javatprogramming'

while i < len(str1):
    if str1[i] == 't':
        i += 1
        break
    print('Current Letter :', str1[i])
    i += 1
```

3. Pass Statement - The pass statement is used to declare the empty loop. It is also used to define empty class, function, and control statement. Let's understand the following example.

Example -

```
# An empty loop
str1 = 'javaprog'
i = 0

while i < len(str1):
    i += 1
    pass
print('character in str1 :', i)
```

Example-1: Program to print 1 to 10 using while loop

```
i=1
#The while loop will iterate until condition becomes false.
while(i<=10):
    print(i)
    i=i+1
```

Output:

```
1
2
3
4
5
6
7
8
9
10
```

Example -2: Program to print table of given numbers.

```
i=1
number = int(input("Enter the number:"))
while i<=10:
    print("%d X %d = %d \n"%(number,i,number*i))
    i = i+1
```

Output:

```
Enter the number:10
10 X 1 = 10

10 X 2 = 20

10 X 3 = 30

10 X 4 = 40

10 X 5 = 50

10 X 6 = 60

10 X 7 = 70

10 X 8 = 80

10 X 9 = 90

10 X 10 = 100
```

Infinite while loop

If the condition is given in the while loop never becomes false, then the while loop will never terminate, and it turns into the **infinite while loop.**

Any **non-zero** value in the while loop indicates an **always-true** condition, whereas zero indicates the always-false condition. This type of approach is useful if we want our program to run continuously in the loop without any disturbance.

Example 1

```
1. while (1):
2.     print("Hi! we are inside the infinite while loop")
```

Output:

```
Hi! we are inside the infinite while loop
Hi! we are inside the infinite while loop
```

Example 2

```
1. var = 1
2. while(var != 2):
3.     i = int(input("Enter the number:"))
4.     print("Entered value is %d"%(i))
```

Output:

```
Enter the number:10
Entered value is 10
Enter the number:10
Entered value is 10
Enter the number:10
Entered value is 10
Infinite time
```

Using else with while loop

Python allows us to use the else statement with the while loop also. The else block is executed when the condition given in the while statement becomes false. Like for loop, if the while loop is broken using break statement, then the else block will not be executed, and the statement present after else block will be executed. The else statement is optional to use with the while loop. Consider the following example.

Example 1

```
i=1
while(i<=5):
    print(i)
    i=i+1
else:
    print("The while loop exhausted")
```

Example 2

```
i=1
while(i<=5):
    print(i)
    i=i+1
    if(i==3):
        break
else:
    print("The while loop exhausted")
```

Output:

```
1
2
```

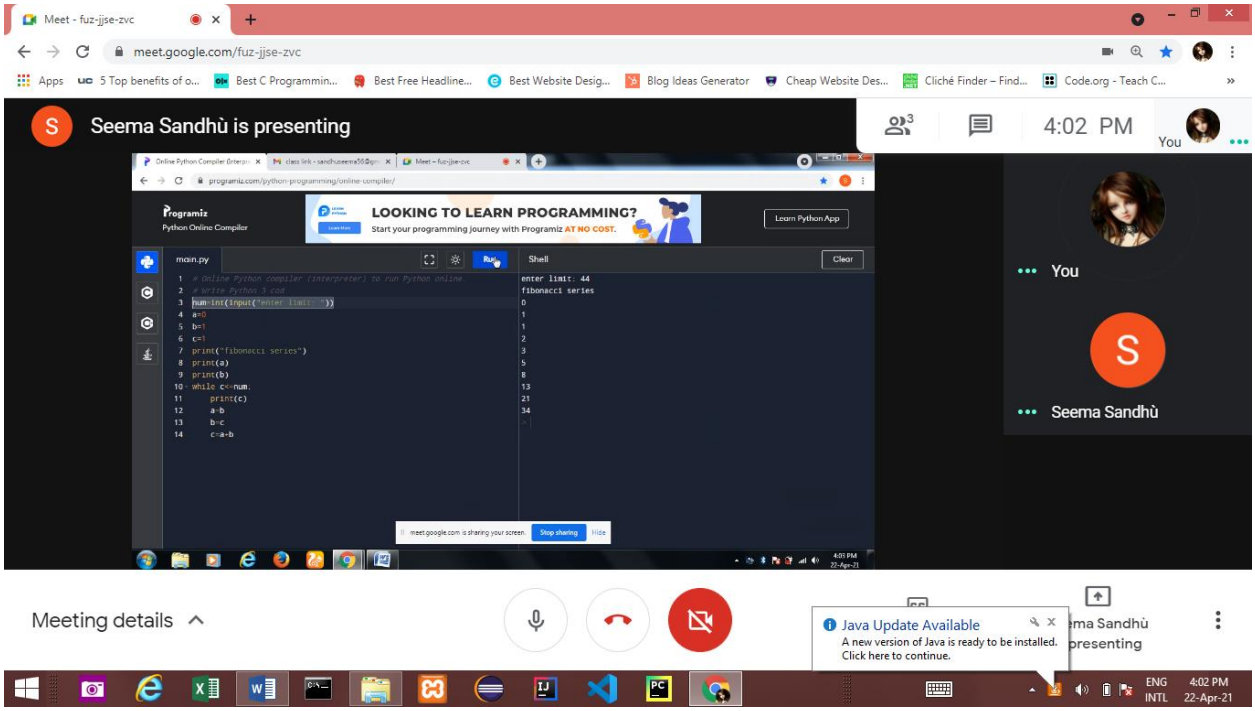
In the above code, when the break statement encountered, then while loop stopped its execution and skipped the else statement.

Example-3 Program to print Fibonacci numbers to given limit

```
terms = int(input("Enter the terms "))
# first two intial terms
a = 0
b = 1
count = 0

# check if the number of terms is Zero or negative
if (terms <= 0):
   print("Please enter a valid integer")
elif (terms == 1):
   print("Fibonacci sequence upto",limit,":")
   print(a)
else:
   print("Fibonacci sequence:")
   while (count < terms) :
       print(a, end = ' ')
       c = a + b
       # updateing values
       a = b
       b = c

    count += 1
```



Output:

```
Enter the terms 10
Fibonacci sequence:
0 1 1 2 3 5 8 13 21 34
```

Do-While Style using while loop

```
1. i = 1
2.
3. while True:
4.     print(i)
5.     i = i + 1
6.     if(i > 5):
7.         break
```