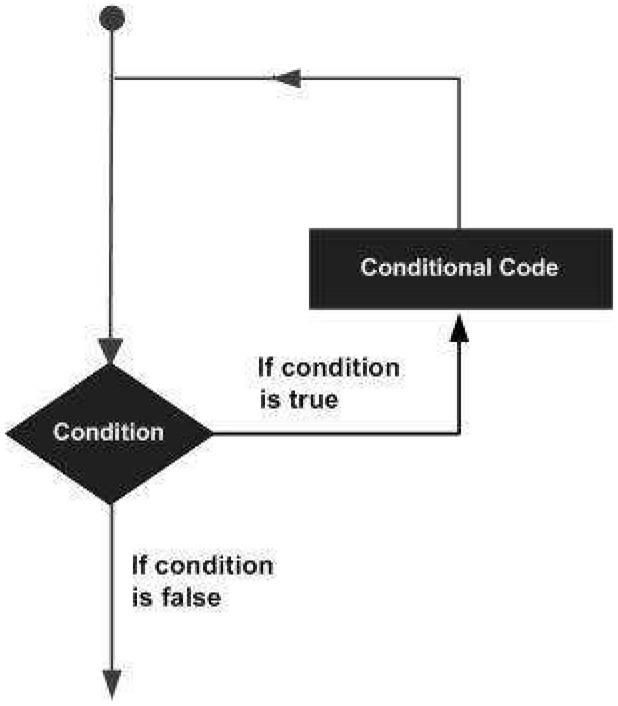
Loops in Python

In general, statements within a program are executed sequentially. The first statement in a program/function is executed first, followed by the second, and so on. There may be a situation when we need to execute a block of code several times. Programming languages provide various control structures that allow more complicated execution paths.
A loop statement allows us to execute a statement or group of statements multiple times.

The following diagram illustrates a loop statement.



Python programming language provides the following types of loops to handle looping requirements.

Loop Type	Description
While loop	Repeats a statement or group of statements while (as long as) a given condition is TRUE. It tests the condition before executing the loop body.
for loop	Executes a sequence of statements multiple times and abbreviates the code that manages the loop variable.
nested loops	we can use one or more loops inside any other while, or for loop.

while Loop Statements

A while loop statement in Python programming language repeatedly executes a target statement as long as a given condition is true.

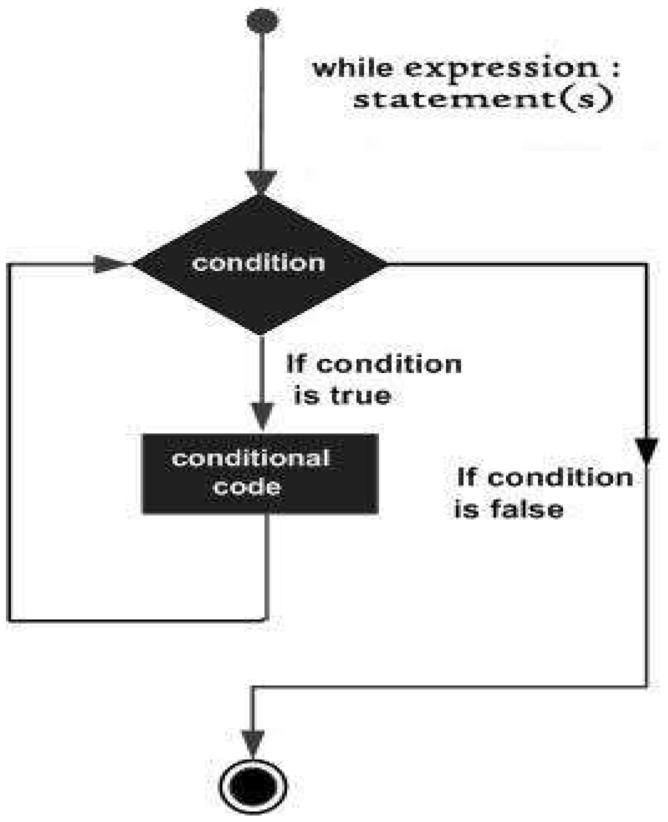
Syntax

The syntax of a while loop in Python programming language is-

```
while expression:
    statement(s)
```

Here, statement(s) may be a single statement or a block of statements with uniform indent. The condition may be any expression, and true is any non-zero value. The loop iterates while the condition is true. When the condition becomes false, program control passes to the line immediately following the loop.
[In Python, all the statements indented by the same number of character spaces after a programming construct are considered to be part of a single block of code. Python uses indentation as its method of grouping statements.]

Flow diagram



Here, a key point of the while loop is that the loop might not ever run. When the condition is tested and the result is false, the loop body will be skipped and the first statement after the while loop will be executed.

Example Code

```
count = 0
while (count < 9):
        print ('The count is:', count)
        count = count + 1
print ("Good bye!")
```

When the above code is executed, it produces the following result-

```
The count is: 0
The count is: 1
The count is: 2
The count is: 3
The count is: 4
The count is: 5
The count is: 6
The count is: 7
The count is: 8
Good bye!
```

The block here, consisting of the print and increment statements, is executed repeatedly until count is no longer less than 9. With each iteration, the current value of the index count is displayed and then increased by 1.

The Infinite Loop

A loop becomes an infinite loop if a condition never becomes FALSE. WE must be cautious when using while loops because of the possibility that this condition never resolves to a FALSE value. This results in a loop that never ends. Such a loop is called an infinite loop.

An infinite loop might be useful in client/server programming where the server needs to run continuously so that client programs can communicate with it as and when required.

```
var = 1
while var == 1 :     # This constructs an infinite loop
        num = int(input("Enter a number:"))
        print ("You entered: ", num)
print ("Good bye!")
```

When the above code is executed, it produces the following result-

```
Enter a number: 20
You entered:20
Enter a number: 29
You entered: 29
Enter a number: -51
You entered: -51
```

The above example goes in an infinite loop and we need to use **CTRL+C to exit the program**.

```
Enter a number:Traceback (most recent call last):
File "examples\test.py", line 5, in
num = int(input("Enter a number:"))
KeyboardInterrupt
```

else Statement with while Loops

Python supports having an else statement associated with a loop statement. **If the else statement is used with a while loop, the else statement is executed when the loop has exhausted iterating the list.**

If the else statement is used with a while loop, the else statement is executed when the condition becomes false.

The following example illustrates the combination of an else statement with a while statement that prints a number as long as it is less than 5, otherwise the else statement gets executed.

```
#!/usr/bin/python3
count = 0
while count < 5:
    print (count, " is less than 5")
    count = count + 1
else:
    print (count, " is not less than 5")
```

When the above code is executed, it produces the following result-

```
0 is less than 5
1 is less than 5
2 is less than 5
3 is less than 5
4 is less than 5
5 is not less than 5
```

Single Statement Suites

Similar to the if statement syntax, if while clause consists only of a single statement,
it may be placed on the same line as the while header.
Here is the syntax and example of a one-line while clause-

```
#!/usr/bin/python3
flag = 1
while (flag): print ('Given flag is really true!')
print ("Good bye!")
```

The above example goes into an infinite loop and you need to **press CTRL+C keys to exit**.

for Loop Statements

The for statement in Python has the ability to iterate over the items of any sequence, such as a list or a string.

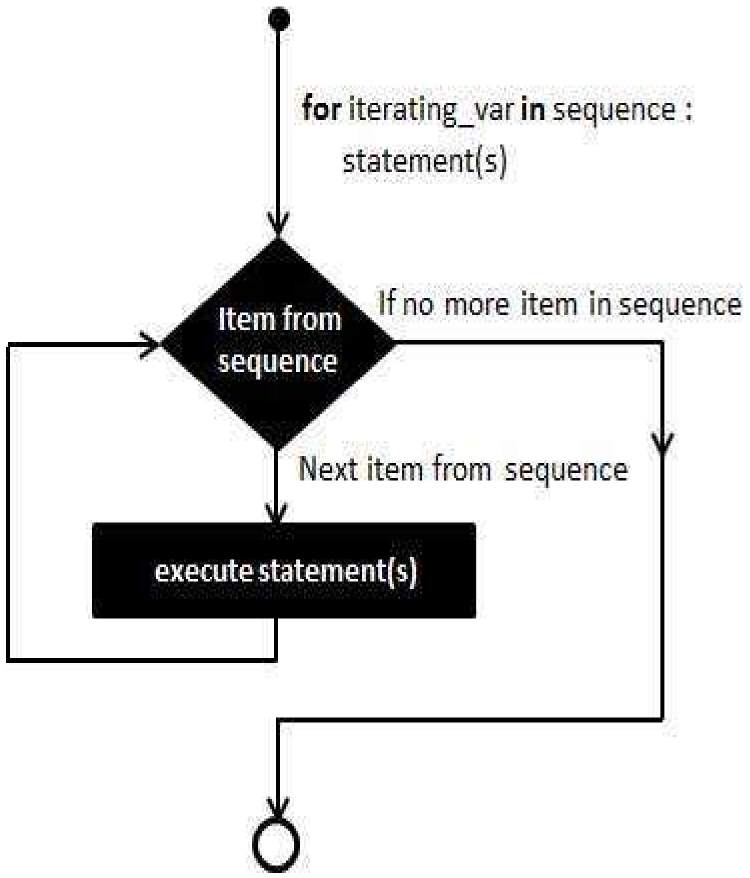
Syntax

The syntax of a for loop in Python programming language is-

```
for iterating_var in sequence:
    statements(s)
```

If a **sequence contains an expression list, it is evaluated first**. Then, the first item in the sequence is assigned to the iterating variable iterating_var. Next, the statements block is executed. **Each item in the list** is **assigned to iterating_var**, and the statement(s) block is executed until the entire sequence is exhausted.

Flow diagram



The range() Function

To loop through a set of code a specified number of times, we can use the range() function, The range() function returns a sequence of numbers, starting from 0 by default, and increments by 1 (by default), and ends at a specified number.

Example: Using the range() function:	**Output**

for *x* ***in*** *range(4):*
  print(x)
	# Note that range(4) is not the values of 0 to 4, but the values 0 to 3.
	0
	1
	2
	3

The range() function defaults to 0 as a starting value, however it is possible to specify the starting value by adding a parameter: range(2, 6), which means values from 2 to 6 (but not including 6):

Code:	Output:
for x in range(2, 6): print(x)	2 3 4 5

The range() function defaults to increment value 1,, however it is possible to specify the increment value as third parameter. Ex **Increment the sequence with -2 (default is 1):**

Code:	Output:
for x in range(6,-1,-2): print(x)	6 4 2 0

Example Code

Code	Output

for var in [1,2,3,4]:
    print('Value is = ',var)
	Value is = 1
	Value is = 2
	Value is = 3
	Value is = 4

for var in ['Engg.','College', 'at','Kolaghat']:
    print('Value is = ',var)
	Value is = Engg.
	Value is = College
	Value is = at
	Value is = Kolaghat

for var in range(4): **print('Value is = ',var)**	Value is = 0 Value is = 1 Value is = 2 Value is = 3
for var in list(range(4)): **print('Value is = ',var)**	Value is = 0 Value is = 1 Value is = 2 Value is = 3
for var in range(-1,3): **print('Value is = ',var)**	Value is = -1 Value is = 0 Value is = 1 Value is = 2

for letter in 'Python': **# traversal of a string sequence** **print ('Current Letter :', letter)** **print() # Print a blank line** **fruits = ['banana', 'apple','mango','banana']** **for fruit in fruits:** **# traversal of List sequence** **print ('Current fruit :', fruit)** **print ("Good bye!")**	Current Letter : P Current Letter : y Current Letter : t Current Letter : h Current Letter : o Current Letter : n Current fruit : banana Current fruit : apple Current fruit : mango Current fruit : banana Good bye!
An alternative way of iterating through each item is by index offset into the sequence itself. Following is a simple example- **#!/usr/bin/python3** **fruits = ['banana', 'apple','mango']** **for index in range(len(fruits)):** **print ('Current fruit :', fruits[index])** **print ("Good bye!")**	Current fruit : banana Current fruit : apple Current fruit : mango Good bye!

Using else Statement with for Loops

Python supports having an else statement associated with a loop statement. If the else statement is used with a for loop, the else block is executed only if for loops terminates normally (and not by encountering a break statement).

The following example illustrates the combination of an else statement with a for statement that searches for even numbers in a given list.

```
#!/usr/bin/python3
numbers=[11,33,55,39,55,75,,37,21,23,41,13]
for num in numbers:
        if num%2==0:
             print ('the list contains at least one even number')
             break
else:
       print ('the list does not contain any even number')
```

When the above code is executed, it produces the following result-

```
          the list does not contain any even number
```

Nested Loops

Python programming language allows the use of one loop inside another loop. The following section shows a few examples to illustrate the concept.

Syntax: while and for nested

The syntax for a nested while loop statement in Python programming language is as follows-

```
while expression:
        while expression:
              statement(s)
       statement(s)
```

The syntax for a nested for loop statement in Python programming language is as follows-

```
for iterating_var in sequence:
        for iterating_var in sequence:
                 statements(s)
       statements(s)
```

A final note on loop nesting is that you can put any type of loop inside any other type of loop. For example a for loop can be inside a while loop or vice versa.

```
while expression:
        for iterating_var in sequence:
                 statements(s)
        statement(s)
```

```
for iterating_var in sequence:
        while expression:
              statement(s)
       statements(s)
```

Example Code

The following program uses a nested-for loop to display multiplication tables from 1-10.

```
#!/usr/bin/python3
import sys
for i in range(1,11):
        for j in range(1,11):
        k=i*j
        print (k, end=' ')
print()
```

The print() function inner loop has **end=' '** which **appends a space instead of default newline**. Hence, the numbers will appear in one row. Last **print() will be executed at the end of inner for loop**. When the above code is executed, it produces the **following result**

```
1 2 3 4 5 6 7 8 9 10
2 4 6 8 10 12 14 16 18 20
3 6 9 12 15 18 21 24 27 30
4 8 12 16 20 24 28 32 36 40
```

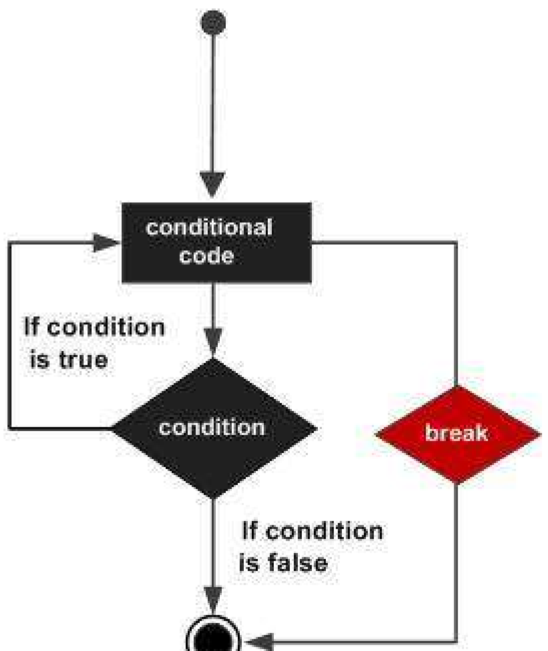
```
5 10 15 20 25 30 35 40 45 50
6 12 18 24 30 36 42 48 54 60
7 14 21 28 35 42 49 56 63 70
8 16 24 32 40 48 56 64 72 80
9 18 27 36 45 54 63 72 81 90
10 20 30 40 50 60 70 80 90 100
```

Loop Control Statements

The Loop control statements change the execution from its normal sequence. When the execution leaves a scope, all automatic objects that were created in that scope are destroyed. Python supports the following 3 control statements

Control Statement	Description
break statement	Terminates the loop statement and transfers execution to the statement immediately following the loop.
continue statement	Causes the loop to skip the remainder of its body and immediately retest its condition prior to reiterating.
pass statement	The pass statement in Python is used when a statement is required syntactically but you do not want any command or code to execute.

Break Statement



The break statement is used for premature termination of the current loop. After abandoning the loop, execution at the next statement is resumed, just like the traditional break statement in C.
The most common use of break is when some external condition is triggered requiring a hasty exit from a loop. The break statement can be used in both while and for loops.
If you are using nested loops, the break statement stops the execution of the innermost

loop and starts executing the next line of the code after the block.

Syntax

The syntax for a break statement in Python is as follows-

break

Code

```
#!/usr/bin/python3
for letter in 'Python':  # First Example
   if letter == 'h':
        break
   print ('Current Letter :', letter)

 # Second Example
var = 10
while var > 0:
   print ('Current variable value :', var)
   var = var -1
   if var == 5:
        break
print ("Good bye!")
```

```
Current Letter : P
Current Letter : y
Current Letter : t

Current variable value : 10
Current variable value : 9
Current variable value : 8
Current variable value : 7
Current variable value : 6
Good bye!
```

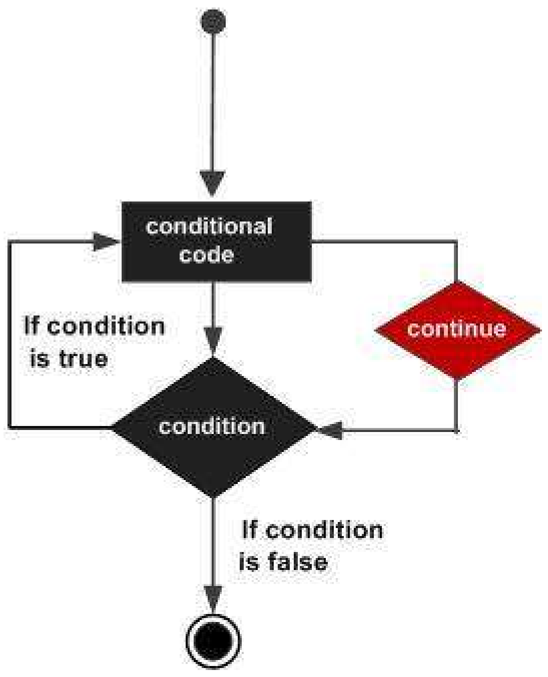
The following program demonstrates the use of break in a for loop iterating over a list. User inputs a number, which is searched in the list. If it is found, then the loop terminates with the 'found' message.

Code:	Output:

```
#!/usr/bin/python3
no=int(input('any number please : '))
numbers=[11,33,55,39,55,75,37,21,23,41,13]
for num in numbers:
    if num==no:
        print ('number found in list')
        break
else:
    print ('number not found in list')
```
	```
	any number please : 33
	number found in list

	any number please : 5
	number not found in list
	```

continue statement



The continue statement in Python returns the control to the beginning of the current loop. When encountered, the loop starts the next iteration without executing the remaining statements in the current iteration.
The continue statement can be used in both while and for loops.

Syntax
continue

Code:	Output:

```
#!/usr/bin/python3
for letter in 'Python':    # First Example
    if letter == 'h':
        continue
```
	```
	Current Letter : P
	Current Letter : y
	Current Letter : t
	Current Letter : o
	```

```
    print ('Current Letter :', letter)

var = 5      # Second Example
while var > 0:
    var = var -1
    if var == 3:
        continue
    print ('Current variable value :', var)
print ("Good bye!")
```

```
Current Letter : n
Current variable value : 5
Current variable value : 4
Current variable value : 2
Current variable value : 1
Current variable value : 0
Good bye!
```

```
# Prints out only odd numbers - 1,3,5,7,9
for x in range(10):
# Check if x is even
if x % 2 == 0:
continue
print(x)

Output : 0 1 2 3 4 1 3 5 7 9
```

pass statement

It is used when a statement is required syntactically but you do not want any command or code to execute.
The pass statement is a null operation; nothing happens when it executes. The pass statement is also useful in places where your code will eventually go, but has not been written.

Syntax
pass

Code:

```
#!/usr/bin/python3
for letter in 'Python':
    if letter == 'h':
        pass
        print ('This is pass block')
   print ('Current Letter :', letter)
print ("Good bye!")
```

Output:

```
Current Letter : P
Current Letter : y
Current Letter : t
This is pass block
Current Letter : h
Current Letter : o
Current Letter : n
Good bye!
```