Python Programming Basics

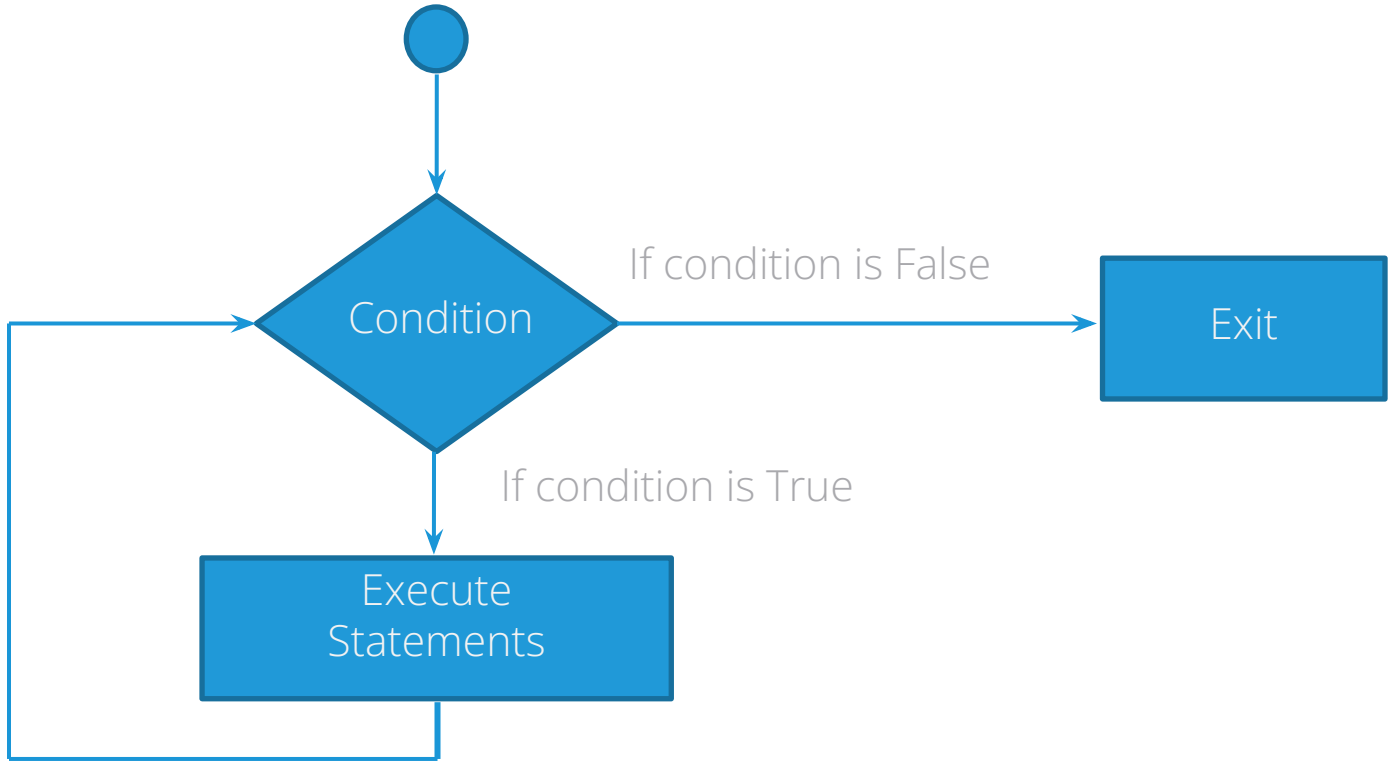
Loops in Python

Contents

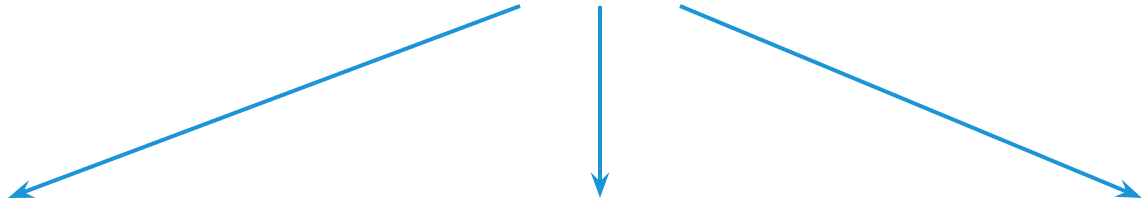
Introduction

Loops are used when a block of codes needs to be executed multiple times (called iteration)

Flow Chart to illustrate a loop statement



Types of Loops



for Loop

- Iterates over the elements of any sequence (vector) till the condition defined is true
- Number of iterations are fixed and known in advance

repeat Loop

- Infinite loop and used with break statement to exit the loop
- Number of iterations depends on the condition which is checked at the end of each iteration

while Loop

- Repeats a statement or group of statements until some condition is met
- Number of iterations depends on the condition which is checked at the beginning of each iteration

for Loop

Syntax:

```
for  Loop_Variable  in  Sequence:
        Statement 1
        Statement 2
         ....
```

Loop_Variable	sets a new value for each iteration of the loop.
Sequence	is a vector assigned to a Loop_Variable
Statement 1, Statement 2,...	Body of **for** consisting of block of program statements contained after :

for Loop

- In **for** loop, the number of times the loop should be executed is typically defined in the initialisation part.
- Loop variable is created when the **for** statement runs.
- Loop variable is assigned the next element in the vector with each iteration and then the statements of the body are executed.

```
# Print numbers 1 to 4
```

```
for i in range(1,5):
    print(i)
```

```
1
2
3
4
```

for Loop

```
# Print how many times Mumbai and Delhi are appearing in the vector

Location = ["Mumbai","Delhi","Delhi","Mumbai","Mumbai","Delhi"]
count = 0

for i in Location:
    if (i == "Mumbai"):
        count = count + 1

countdelhi = len(Location)-count
print("Mumbai:",count)
print("Delhi:",countdelhi)
```

```
Mumbai: 3
Delhi: 3
```

- Here the loop iterates 6 times as the vector **Location** has 6 elements.
- In each iteration, **i** takes on the value of corresponding element of **Location.**
- Counter object **count** is used to count **Mumbai** in **Location.**
- **if** statement checks for **Mumbai** in **Location** and increases the **count** by 1.
- **countdelhi** is a object which has count of **Delhi** in **Location**.
- **length()** returns the no. of elements in the object.
- count of **Mumbai** is subtracted from length of **Location**, giving the count of **Delhi.**

for Loop

- For loop is the most famous among all loops available in Python and its construct implies that the number of iterations is fixed and known in advance, as in cases like “generate the first 100 prime numbers” or “enlist the 10 most important clients”.
- But what if we do not know till when the loop should be iterated or control the number of iterations and one or several conditions may occur which are not predictable beforehand?
- In that case the while and next loops may come to the rescue.

Nested for Loop

Syntax:

```
for Loop_Variable1 in Sequence1 :
    Statement1
   for Loop_Variable2 in Sequence2:
          Statement2
          Statement3
```

Loop_Variable1, Loop_Variable2	sets a new value for each iteration of the loop.
Sequence1, Sequence2	is a vector assigned to their respective Loop_Variable
Statement 1, Statement 2, Statement 3,....	Body of **for** consisting of block of program statements contained after :

Nested for Loop

- The placing of one loop inside the body of another loop is called nesting.
- When you “nest” two loops, the outer loop takes control of the number of complete repetitions of the inner loop. Thus inner loop is executed N- times for every execution of Outer loop.

```
# Print numbers :
for i in range(1,6):
    for j in range(1,3):
        print(i*j)
```

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

Nested for Loop

Important Note :

- While writing a nested for loop, always practice indentation.
- Python creates an indentation automatically, it follows indentation rules strictly. Hence, if there is some wrong indentation then Python gives indentation error while execution.

while Loop

Syntax:

```
while  test_expression:
        Statement 1
        Statement 2
          ....
```

Statement 1, Statement 2,...	Body of while consisting of block of program statements contained after :

while Loop

- The initialization part defines a condition for the loop. The condition is checked every time at the beginning of the loop. The body of the loop is executed only if the condition evaluates to TRUE. This process is repeated until the condition evaluates to False.

```
# Print odd numbers between 1 to 10
i = 1
while(i < 11):
    print(i)
    i = i+2
```

Object **i** is incremented by 2 each time inside the loop.

```
1
3
5
7
9
```

while Loop

```
# Take input as a number from user
# Print the sum of natural numbers upto that number.

number = int(input("Enter a number: "))
if number < 0:
    print("Enter a positive number"
else:
    sum = 0
    while(number > 0):
        sum = sum + number
        number = number - 1
    print("The sum is", sum)
```

```
Enter a number: 3
The sum is 6
```

- **if** condition checks whether the number is less than zero or not; if returns TRUE, it tells user to print a positive number.
- **sum** is a counter object; set to 0.
- **while** loop is used to iterate until the **number** becomes 0 till then the value of **number** is added to **sum** and **number** is decremented by 1 with each iteration and when it becomes 0, the loop will stop and result will be printed.

Here the input is 3, hence the sum of 5 natural number is 6

Note : input() is explained in if..else Conditional Statements tutorial.

repeat using while Loop

- In Python there is no separate function for repeat loop. This loop is similar to previous **`while`** loop, but it is used when we want the blocks of statements to be executed at least once, no matter what the result of the condition.

- Note that you had to set a condition within the loop with a **`break`** statement to exit otherwise the loop would have executed infinite times. This statement introduces us to the notion of exiting or interrupting cycles within loops.

repeat using while Loop

Syntax:

```
while True:
  Statement 1
  Statement 2
  if (condition):
 break
```

break	Terminates the loop statement and transfers execution to the statement immediately following the loop.
Statement 1, Statement 2,...	Body of repeat consisting of block of program Statements contained after :

repeat using while Loop

- The condition is placed at the end of the loop body, so the statements inside the loop body are executed at least once, no matter what the result of condition and are repeated until the condition evaluates to FALSE.

```
# Print even numbers between 0 to 10
total = 0
while True:
    print(total)
    total = total + 2
    if(total > 8):
        break
```

total is a counter object; set to 0.
total will be incremented by 2 with each iteration and if it becomes greater than 8 the loop will stop

```
0
2
4
6
8
```

Interruption and Exit Statements in Python

How do you exit from a loop?

Can you stop or interrupt the loop, aside from the “natural” end, which occurs either because you reached the given number of iterations (for) or because you met a condition (while, repeat)?

And if yes how?

- **`break`** statement responds to the first question. It passes the control to the statement immediately after the end of the loop (if any). We have already seen how to use **`break`** statement in the last example.
- **`next`** statement discontinues the current iteration of the loop without terminating it and starts next iteration of the loop

Interruption and Exit Statements in Python

```
# print numbers between 1 to 15 which are not divisible by 4
```

```
for i in range(1,15):
    if(i%4!=0):
        next
    else:
        print(i)
```

if statement checks whether value in **i** is divisible by 4 or not; if it returns TRUE then it skips all the statements after that and starts the next iteration of the loop.

```
4
8
12
```

Interruption and Exit Statements in Python

```
# break the loop if a number in a range of 1 to 15 is divisible 4
for i in range(1,15):
    if(i%4==0):
        break
    else:
        print(i)
```

if statement checks whether value in **i** is divisible by 4 or not; if it returns TRUE then it exits the for loop and stops further iteration of the loop.

```
1
2
3
```

Quick Recap

In this session, we learnt how different types of loops are used in programming to repeat a specific block of code. Here is a quick recap .

for loop	• Number of iterations are defined in the beginning and runs till the condition defined is TRUE • Counter is automatically incremented • Nested for loop is used for multiple conditions
while loop	• Condition is checked in the beginning of the loop • Counter is incremented inside the loop
Interruption and exit statements	• **`break`**: exits the loop • **`next`**: discontinues the current iteration