Nested Loops in Python

In Python, **a loop inside a loop is known as a nested loop**.

What is a Nested Loop in Python?

A nested loop is a loop inside the body of the outer loop. The inner or outer loop can be any type, such as a while loop or for loop. For example, the outer `for` loop can contain a `while` loop and vice versa.

The outer loop can contain more than one inner loop. There is no limitation on the chaining of loops.

In the nested loop, the number of iterations will be equal to the number of iterations in the outer loop multiplied by the iterations in the inner loop.

In each iteration of the outer loop inner loop execute all its iteration. **For each iteration of an outer loop the inner loop re-start and completes its execution** before the outer loop can continue to its next iteration.

Nested loops are typically used for working with multidimensional data structures, such as printing two-dimensional arrays, iterating a list that contains a nested list.

A nested loop is a part of a control flow statement that helps you to understand the basics of Python.

Python Nested for Loop

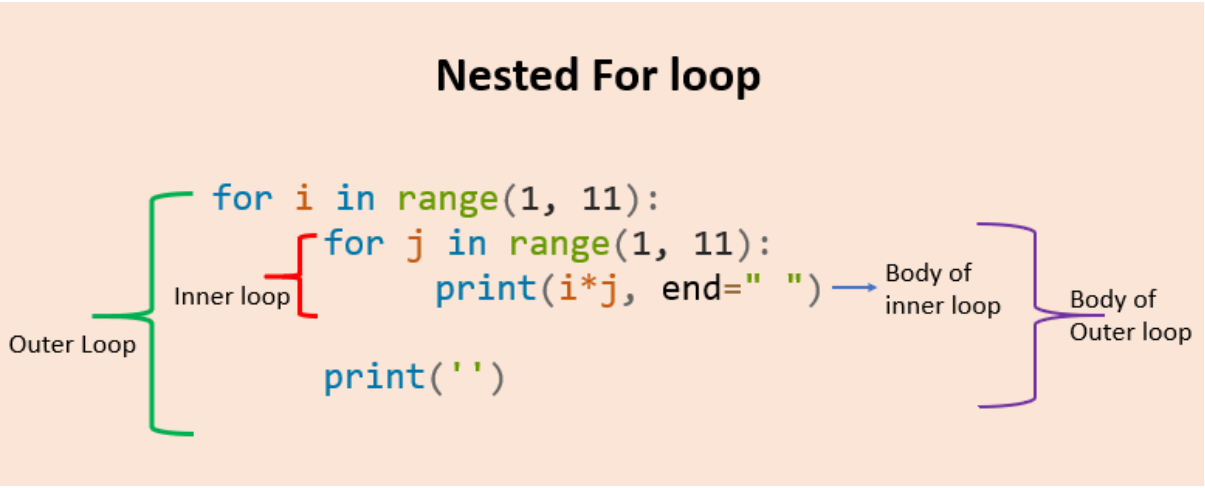
In Python, the for loop is used to iterate over a sequence such as a list, string, tuple, other iterable objects such as range.

Syntax of using a nested for loop in Python

```
# outer for loop
for element in sequence
   # inner for loop
   for element in sequence:
       body of inner for loop
   body of outer for loop
```

In this example, we are using a for loop inside a `for` loop. In this example, we are **printing a multiplication table** of the first ten numbers.

- The outer `for` loop uses the range() function to iterate over the first ten numbers
- The inner `for` loop will execute ten times for each outer number
- In the body of the inner loop, we will print the multiplication of the outer number and current number
- The inner loop is nothing but a body of an outer loop.



Python nested for loop

Example: Write a nested `for` loop program to print multiplication table in Python

```
# outer loop
for i in range(1, 11):
    # nested loop
    # to iterate from 1 to 10
    for j in range(1, 11):
        # print multiplication
        print(i * j, end=' ')
    print()
```


Output:

```
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
```

- In this program, the outer `for` loop is iterate numbers from 1 to 10. The `range()` return 10 numbers. So total number of iteration of the outer loop is 10.
- In the first iteration of the nested loop, the number is 1. In the next, it 2. and so on till 10.
- Next, For each iteration of the outer loop, the inner loop will execute ten times. The inner loop will also execute ten times because we are printing multiplication table up to ten.
- In each iteration of an inner loop, we calculated the multiplication of two numbers.

Nested Loop to Print Pattern

Another most common use of nested loop is to print various star and number patterns.

Let's see how to use a nested loop to print the following pattern in Python.

Pattern:

```
*

* *

* * *

* * * *

* * * * *
```


Program:

```
rows = 5
# outer loop
for i in range(1, rows + 1):
    # inner loop
    for j in range(1, i + 1):
        print("*", end=" ")
```

```
    print('')
```

- In this program, the outer loop is the number of rows print.
- The number of rows is five, so the outer loop will execute five times
- Next, the inner loop is the total number of columns in each row.
- For each iteration of the outer loop, the columns count gets incremented by 1
- In the first iteration of the outer loop, the column count is 1, in the next it 2. and so on.
- The inner loop iteration is equal to the count of columns.
- In each iteration of an inner loop, we print star

While loop inside a for loop

It is very common and helpful to use one type of loop inside another. we can put a while loop inside the `for` loop.

Assume we wanted to repeat each name from a list five times.

- Here we will iterate the list using an outer for loop
- In each iteration of outer for loop, the inner for loop execute five times to print the current name five times

```
names = ['Kelly', 'Jessa', 'Emma']
# outer loop
for name in names:
    # inner while loop
    count = 0
    while count < 5:
        print(name, end=' ')
        # increment counter
        count = count + 1
    print()
```

Output:

```
Kelly Kelly Kelly Kelly Kelly
```

```
Jessa Jessa Jessa Jessa Jessa

Emma Emma Emma Emma Emma
```

Practice: Print a rectangle Pattern with 5 rows and 3 columns of stars

Solve the below Python nested loop exercise.

Print following rectangle of stars

```
***

***

***

***

***
```


Solve more loop exercises.

Show Solution

Break Nested loop

The break statement is used inside the loop to exit out of the loop. If the break statement is used inside a nested loop (loop inside another loop), it will terminate the innermost loop.

In the following example, we have two loops. The outer `for` loop iterates the first four numbers using the `range()` function, and the inner `for` loop also iterates the first four numbers. If the **outer number and a current number of the inner loop** are the same, then break the inner (nested) loop.

Example:

```
for i in range(4):
    for j in range(4):
        if j == i:
            break
        print(i, j)
```

Output:

```
1 0

2 0

2 1

3 0

3 1

3 2
```


As you can see in the output, no rows contain the same number.

Continue Nested loop

The continue statement skip the current iteration and move to the next iteration. In Python, when the `continue` statement is encountered inside the loop, it skips all the statements below it and immediately jumps to the next iteration.

In the following example, we have two loops. The outer for loop iterates the first list, and the inner loop also iterates the second list of numbers.

If the outer number and the inner loop's current number are the same, then move to the next iteration of an inner loop.

Example:

```
first = [2, 4, 6]
second = [2, 4, 6]
for i in first:
```

```
    for j in second:
        if i == j:
            continue
        print(i, '*', j, '= ', i * j)
```

Output:

```
2 * 4 = 8

2 * 6 = 12

4 * 2 = 8

4 * 6 = 24

6 * 2 = 12

6 * 4 = 24
```


As you can see in the output, no same numbers multiplying to each other.

Single Line Nested Loops Using List Comprehension

For example, if you had two lists and want to get all combinations of them, To achieve this, you need to use two nested loops as mentioned below.

```
first = [2, 3, 4]
second = [20, 30, 40]
final = []
for i in first:
    for j in second:
        final.append(i+j)
print(final)
```

You can write more fast and compact code using the list compression and nested loop like as shown below.

```
first = [2, 3, 4]
```

```
second = [20, 30, 40]
final = [i+j for i in first for j in second]
print(final)
```

How to write it:

- First, Write an outer `for` loop that will iterate the first list like `[for i in first]`
- Next, Write an inner loop that will iterate the second list after the outer loop like `[for i in first for j in second]`
- Last, calculate the addition of the outer number and inner number like `[i+j for i in first for j in second]`
- At last, store result in a new list like `final = [i+j for i in first for j in second]`

Let's see more such examples.

In this example, we will use two `for` loops in list Comprehension and the final result would be a list of lists. we will not include the same numbers in each list. we will filter them using an if condition.

```
final = [[x, y] for x in [10, 20, 30] for y in [30, 10, 50] if x != y]
print(final)
```

Output:

```
[[10, 30], [10, 50], [20, 30], [20, 10], [20, 50], [30, 10], [30, 50]]
```

Nested while Loop in Python

In Python, The while loop statement repeatedly executes a code block while a particular condition is true. We use w a while loop when number iteration is not fixed.

In this section, we will see how to use a while loop inside another while loop.

The syntax to write a **nested while loop** statement in Python is as follows:

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

Example:

AD

In this example, we will print the first 10 numbers on each line 5 times.

```
i = 1
while i <= 5:
    j = 1
    while j <= 10:
        print(j, end='')
        j = j + 1
    i = i + 1
    print()
```

Output:

```
12345678910

12345678910

12345678910

12345678910

12345678910
```

for loop inside While loop

Sometimes it is helpful to use one type of loop inside another. we can put a `for` loop inside the `while` loop.

Assume we wanted to **print all perfect numbers from 1 to 100**

- Here we will iterate the first 100 numbers using a `while` loop
- In each iteration of the outer `while` loop, the inner `for` loop execute from 1 up to the current outer number to check if the current number is a perfect number.

```
print('Show Perfect number fom 1 to 100')
n = 2
# outer while loop
while n <= 100:
    x_sum = 0
    # inner for loop
    for i in range(1, n):
        if n % i == 0:
            x_sum += i
    if x_sum == n:
        print('Perfect number:', n)
    n += 1
```

When To Use a Nested Loop in Python?

- Nested loops are handy when you have nested arrays or lists that need to be looped through the same function.
- When you want to print different star and number patterns using rows can columns

Keep the time complexity in mind. Let's understand this with examples on how nested for loop work in Python.

We use for loop to iterates on the given elements of a sequence or iterable. like `for i in list`. Here time complexity is O(n) because we are iterating all items from a list.

The number of execution steps (iterations) determines the time complexity of a loop.

When you use a nested loop and both outer and inner loop runs without any if condition in it, the time complexity is `O(n^2)` because, for all of the n elements, the code is executed n times.

Example:

```
numbers = [[1, 2, 3], [4, 5, 6]]

cnt = 0
for i in numbers:
    for j in i:
        print('iteration', cnt, end=': ')
```

```
        print(j)
        cnt = cnt + 1
```

Output:

```
iteration 0: 1

iteration 1: 2

iteration 2: 3

iteration 3: 4

iteration 4: 5

iteration 5: 6
```


If you give a condition in the inner loop that will stop executing after some elements and not execute all n iterations of the inner loop or outer loop, it will have less time complexity.

Use nested loop when you don’t have any better alternatives, Remember writing efficient and compact code is far better than writing complex code.