005_Python_for_Loop

March 9, 2023

1 Loops in Python

Loops in Python programming function similar to loops in C, C++, Java or other languages. Python loops are used to repeatedly execute a block of statements until a given condition returns to be `False`. In Python, we have **two types of looping statements**, namely:

2 Python `for` Loop

In this class, you'll learn to iterate over a sequence of elements using the different variations of `for` loop. We use a `for` loop when we want to repeat a code block for a **fixed number of times**.

2.1 What is `for` loop in Python?

The for loop in Python is used to iterate over a sequence ([**string**], [**list**], [**dictionary**], [**set**], or [**tuple**]). Iterating over a sequence is called traversal.

2.2 Why use `for` loop?

Let's see the use `for` loop in Python.

- **Definite Iteration:** When we know how many times we wanted to run a loop, then we use count-controlled loops such as `for` loops. It is also known as definite iteration. For example, Calculate the percentage of 50 students. here we know we need to iterate a loop 50 times (1 iteration for each student).
- **Reduces the code's complexity:** Loop repeats a specific block of code a fixed number of times. It reduces the repetition of lines of code, thus reducing the complexity of the code. Using `for` loops and while loops we can automate and repeat tasks in an efficient manner.
- **Loop through sequences:** used for iterating over lists, strings, tuples, dictionaries, etc., and perform various operations on it, based on the conditions specified by the user.

2.2.1 Syntax :

```
for element in sequence:
    body of for loop
```

1. First, `element` is the variable that takes the value of the item inside the sequence on each iteration.
2. Second, all the `statements` in the body of the for loop are executed with the same value. The body of for loop is separated from the rest of the code using indentation.

3. Finally, loop continues until we reach the last item in the **sequence**. The body of for loop is separated from the rest of the code using indentation.

[1]:
```
# Example 1: For loop

words = ['one', 'two', 'three', 'four', 'five']

for i in words:
    print(i)
```

```
one
two
three
four
five
```

[2]:
```
# Example 2: Calculate the average of list of numbers

numbers = [10, 20, 30, 40, 50]

# definite iteration
# run loop 5 times because list contains 5 items
sum = 0
for i in numbers:
    sum = sum + i
list_size = len(numbers)
average = sum / list_size
print(average)
```

```
30.0
```

2.3 `for` loop with `range()` function

The [**range()**] function returns a sequence of numbers starting from 0 (by default) if the initial limit is not specified and it increments by 1 (by default) until a final limit is reached.

The **`range()`** function is used with a loop to specify the range (how many times) the code block will be executed. Let us see with an example.

We can generate a sequence of numbers using **`range()`** function. **`range(5)`** will generate numbers from 0 to 4 (5 numbers).

The **`range()`** function is "lazy" in a sense because it doesn't generate every number that it "contains" when we create it. However, it is not an iterator since it supports [**len()**] and **`__getitem__`** operations.

This **`range()`** function does not store all the values in memory; it would be inefficient. So it remembers the start, stop, step size and generates the next number on the go.

We can also define the start, stop and step size as **`range(start, stop,step_size)`**. **`step_size`** defaults to 1 if not provided.

[3]:
```
# Example 1: How range works in Python?

# empty range
print(list(range(0)))

# using range(stop)
print(list(range(10)))

# using range(start, stop)
print(list(range(1, 10)))
```

```
[]
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

[4]:
```
# Example 2:

for num in range(10):
    print(num)
```

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

[5]:
```
# Example 3:

for i in range(1, 11):
    print(i)
```

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

[6]:
```
# Example 4:

for i in range (2, 12, 2):  # beginning 2 with distance of 2 and stop before 12
    print (i)
```

```
2
4
6
8
10
```

[7]:
```
# Example 5:

num=2

for a in range (1,6):  # range (1,6) means numbers from 1 to 5, i.e.,␣
 ↪(1,2,3,4,5)
    print (num * a)
```

```
2
4
6
8
10
```

[8]:
```
# Example 6: Find Sum of 10 Numbers

sum=0
for n in range(1,11):  # range (1,11) means numbers from 1 to 5, i.e.,␣
 ↪(1,2,3,4,5,6,7,8,9,10)
    sum+=n
    print (sum)

'''
0+1 = 1
1+2 = 3
3+3 = 6
6+4 = 10
10+5 =15
21
28
36
45
45+10 = 55
'''
```

```
1
3
```

```
6
10
15
21
28
36
45
55
```

[8]:
```
'\n0+1 = 1\n1+2 = 3\n3+3 = 6\n6+4 = 10\n10+5 =15\n21\n28\n36\n45\n45+10 = 55\n'
```

[9]:
```
# Example 7: printing a series of numbers using for and range

print("Case 1:")
for i in range(5):  # Print numbers from 0 to 4
    print (i)

print("Case 2:")
for i in range(5, 10):  # Print numbers from 5 to 9
    print (i)

print("Case 3:")
for i in range(5, 10, 2):  # Print numbers from 5 with distace 2 and stop␣
 ↪before 10
    print (i)
```

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

2.4 `for` loop with `if-else`

A **`for`** loop can have an optional **[if-else]** block. The **`if-else`** checks the condition and if the condition is **`True`** it executes the block of code present inside the **`if`** block and if the condition is **`False`**, it will execute the block of code present inside the **`else`** block.

[10]:
```python
# Example 1: Print all even and odd numbers

for i in range(1, 11):
    if i % 2 == 0:
        print('Even Number:', i)
    else:
        print('Odd Number:', i)
```

```
Odd Number: 1
Even Number: 2
Odd Number: 3
Even Number: 4
Odd Number: 5
Even Number: 6
Odd Number: 7
Even Number: 8
Odd Number: 9
Even Number: 10
```

2.5 `for` loop with `else`

A **`for`** loop can have an optional **`else`** block as well. The **`else`** part is executed if the items in the sequence used in for loop exhausts.

`else` block will be skipped/ignored when:

- **`for`** loop terminate abruptly
- the [**break statement**] is used to break the **`for`** loop.

[11]:
```python
# Example 1:

digits = [0, 1, 5]

for i in digits:
    print(i)
else:
    print("No items left.")
```

```
0
1
5
No items left.
```

Explanation:

Here, the for loop prints items of the list until the loop exhausts. When the for loop exhausts, it executes the block of code in the **`else`** and prints **`No items left`**.

[12]:
```python
# Example 2:
```

```
for number in range(11):
    print(number)    # prints 0 to 10, not including 11
else:
    print('The loop stops at', number)
```

```
0
1
2
3
4
5
6
7
8
9
10
The loop stops at 10
```

[13]:
```
# Example 3: Else block in for loop

for i in range(1, 6):
    print(i)
else:
    print("Done")
```

```
1
2
3
4
5
Done
```

This **for-else** statement can be used with the **break** keyword to run the **else** block only when the **break** keyword was not executed. Let's take an example:

[14]:
```
# Example 4:

student_name = 'Arthur'

marks = {'Alan': 99, 'Bill': 55, 'Cory': 77}

for student in marks:
    if student == student_name:
        print(marks[student])
        break
else:
    print('No entry with that name found.')
```

```
No entry with that name found.
```

[15]:
```
# Example 5:

count = 0
for i in range(1, 6):
    count = count + 1
    if count > 2:
        break
    else:
        print(i)
else:
    print("Done")
```

```
1
2
```

2.6 Using Control Statement in `for` loops in Python

[**Control statements**] in Python like **`break`**, **`continue`**, etc can be used to control the execution flow of **`for`** loop in Python. Let us now understand how this can be done.

It is used when you want to exit a loop or skip a part of the loop based on the given condition. It also knows as **transfer statements**.

2.6.1 a) `break` in `for` loop

Using the **`break`** statement, we can exit from the **`for`** loop before it has looped through all the elements in the sequence as shown below. As soon as it breaks out of the **`for`** loop, the control shifts to the immediate next line of code. For example,

[16]:
```
# Example 1:

numbers = (0,1,2,3,4,5)
for number in numbers:
    print(number)
    if number == 3:
        break
```

```
0
1
2
3
```

Explanation:

In the above example, the loop stops when it reaches 3.

[17]:
```
# Example 2:
```

```
color = ['Green', 'Pink', 'Blue']
for i in color:
    if(i == 'Pink'):
        break
print (i)
```

```
Pink
```

Explanation:

Here, in the second iteration, the **`if`** condition becomes **`True`**. Hence the loop beaks out of the for loop and the immediate next line of code i.e **`print (i)`** is executed and as a result, pink is outputted.

[18]:
```
# Example 3:

numbers = [1, 4, 7, 8, 15, 20, 35, 45, 55]
for i in numbers:
    if i > 15:
        # break the loop
        break
    else:
        print(i)
```

```
1
4
7
8
15
```

[19]:
```
# Example 4:

for i in range(5):
    for j in range(5):
        if j == i:
            break
        print(i, j)
```

```
1 0
2 0
2 1
3 0
3 1
3 2
4 0
4 1
4 2
4 3
```

Explanation:

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.

2.6.2 b) continue in for loop

The **continue** statement is used to stop/skip the block of code in the loop for the current iteration only and continue with the next iteration. For example,

[20]:
```
# Example 1:

color = ['Green', 'Pink', 'Blue']
for i in color:
    if(i == 'Pink'):
        continue
print (i)
```

```
Blue
```

Explanation:

Here, in the second iteration, the condition becomes **True**. Hence the interpreter skips the **print (i)** statement and immediately executes the next iteration.

[21]:
```
# Example 2:

numbers = (0,1,2,3,4,5)
for number in numbers:
    print(number)
    if number == 3:
        continue
    print('Next number should be ', number + 1) if number != 5 else␣
 ↪print("loop's end") # for short hand conditions need both if and else␣
 ↪statements
print('outside the loop')
```

```
0
Next number should be  1
1
Next number should be  2
2
Next number should be  3
3
4
Next number should be  5
5
loop's end
outside the loop
```

Explanation:

In the example above, if the number equals 3, the step **after** the condition (but inside the loop) is skipped and the execution of the loop continues if there are any iterations left.

[22]:
```
# Example 3:

first = [3, 6, 9]
second = [3, 6, 9]
for i in first:
    for j in second:
        if i == j:
            continue
        print(i, '*', j, '= ', i * j)
```

```
3 * 6 =  18
3 * 9 =  27
6 * 3 =  18
6 * 9 =  54
9 * 3 =  27
9 * 6 =  54
```

Explanation:

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.

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

[23]:
```
# Example 4:

name = "mariya mennen"
count = 0
for char in name:
    if char != 'm':
        continue
    else:
        count = count + 1

print('Total number of m is:', count)
```

```
Total number of m is: 2
```

2.6.3 c) pass in for loop

The **pass** statement is a null statement, i.e., nothing happens when the statement is executed. Primarily it is used in empty functions or classes. When the interpreter finds a pass statement in the program, it returns no operation.

[24]:
```
# Example 1:

for number in range(6):
    pass
```

[25]:
```
# Example 2:

num = [1, 4, 5, 3, 7, 8]
for i in num:
    # calculate multiplication in future if required
    pass
```

2.7 Reverse for loop

Till now, we have learned about forward looping in **for** loop with various examples. Now we will learn about the backward iteration of a loop.

Sometimes we require to do reverse looping, which is quite useful. For example, to reverse a list.

There are three ways to iterating the **for** loop backward:

- Reverse **for** loop using [**range()**] function
- Reverse **for** loop using the [**reversed()**] function

2.7.1 Backward Iteration using the `reversed()` function

We can use the built-in function **reversed()** with **for** loop to change the order of elements, and this is the simplest way to perform a reverse looping.

[26]:
```
# Example 1: Reversed numbers using `reversed()` function

list1 = [10, 20, 30, 40]
for num in reversed(list1):
    print(num)
```

```
40
30
20
10
```

2.7.2 Reverse for loop using `range()`

We can use the built-in [**range()**] function with the **for** loop to reverse the elements order. The **range()** generates the integer numbers between the given start integer to the stop integer.

[27]:
```
# Example 1:

print("Reverse numbers using for loop")
num = 5
# start = 5
```

```
# stop = -1
# step = -1
for num in (range(num, -1, -1)):
    print(num)
```

```
Reverse numbers using for loop
5
4
3
2
1
0
```

There are many helper functions that make **for** loops even more powerful and easy to use. For example [**enumerate()**], [**zip()**], [**sorted()**], [**reversed()**]

[28]:
```
# Example 2:

print("reversed: \t",end="")
for ch in reversed("abc"):
    print(ch,end=";")

print("\nenuemerated:\t",end="")
for i,ch in enumerate("abc"):
    print(i,"=",ch,end="; ")

print("\nzip'ed: ")
for a,x in zip("abc","xyz"):
    print(a,":",x)
```

```
reversed:       c;b;a;
enuemerated:    0 = a; 1 = b; 2 = c;
zip'ed:
a : x
b : y
c : z
```

2.8 Nested `for` loops

Nested `for` loop is a **`for`** loop inside another **`for`** a loop.

A nested loop has one loop inside of another. In Python, you can use any loop inside any other loop. For instance, a **`for`** loop inside a **while loop**, a **`while`** inside **`for`** in and so on. It is mainly used with two-dimensional arrays.

In nested loops, the inner loop finishes all of its iteration for each iteration of the outer loop. i.e., For each iteration of the outer loop inner loop restart and completes all its iterations, then the next iteration of the outer loop begins.

Syntax:

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

2.8.1 `for` loop inside `for` loop

Example: Nested `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.

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

[29]:
```
# Example 1: printing a multiplication table of the first ten numbers

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

```
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
```

Explanation:

- 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, the inner loop will also execute ten times because we rea printing multiplication table up to ten. For each iteration of the outer loop, the inner loop will execute ten times.
- In each iteration of an inner loop, we calculated the multiplication of two numbers.

[30]:
```
# Example 1:

person = {
    'first_name':'Milaan',
    'last_name':'Parmar',
    'age':96,
    'country':'Finland',
    'is_marred':True,
    'skills':['Python', 'Matlab', 'R', 'C', 'C++'],
    'address':{
        'street':'Space street',
        'zipcode':'02210'
    }
}
for key in person:
    if key == 'skills':
        for skill in person['skills']:
            print(skill)
```

```
Python
Matlab
R
C
C++
```

[31]:
```
# Example 2: Write a code to add all the prime numbers between 17 to 53 using␣
 ↪while loop
# 17, 19, 23, 29, 31, 37, 41, 43, 47, 53

sum=0
for i in range(17,54):
    for j in range(2,i):
        if i%j ==0:
            break
    else:
        sum=sum+i
        print(i)
print(sum)
```

```
17
19
23
29
31
37
41
43
```

```
47
53
340
```

[32]:
```
# Example 3: iterating through nested for loops

color = ['Red', 'Pink']
element = ['flower', 'watch']
for i in color:
    for j in element:
        print(i, j)
```

```
Red flower
Red watch
Pink flower
Pink watch
```

[33]:
```
# Example 4: A use case of a nested for loop in `list_of_lists` case would be

list_of_lists = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
total=0
for list1 in list_of_lists:
    for i in list1:
        total = total+i
print(total)
```

```
45
```

[34]:
```
# Example 5:

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
```

```
iteration 0: 1
iteration 1: 2
iteration 2: 3
iteration 3: 4
iteration 4: 5
iteration 5: 6
```

Example: Nested `for` loop to print pattern

```
*
* *
* * *
* * * *
* * * * *
```

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

Explanation:

- 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

[35]:
```
# Example 1: Method 1

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

```
*
* *
* * *
* * * *
* * * * *
```

[36]:
```
# Example 1: Method 2 - Print Floyd Triangle with user input


ran = input("How many rows? ")

rang = int(ran)
k = 1
for i in range(1, rang+1):
    for j in range(1, i+1):
        print("*", end=" ")
        k = k + 1
```

```
    print()
```

```
How many rows? 5
*
* *
* * *
* * * *
* * * * *
```

[37]:
```
# Example 2: Method 1

for i in range(1,6):        # numbers from 0,1,2,3,4,5
    for j in range(1, i+1):
        print(i, end=" ")  # print number
    print(" ")             # line after each row to display pattern correctly
```

```
1
2 2
3 3 3
4 4 4 4
5 5 5 5 5
```

[38]:
```
# Example 2: Method 2

rows = 6
for num in range(rows):     # from 0,1,2,3,4,5
    for i in range(num):
        print(num,end=" ")  # print the number
    print(" ")              # line after each row to print
```

```
1
2 2
3 3 3
4 4 4 4
5 5 5 5 5
```

[39]:
```
# Example 3: Method 3

n=5  # giving number of rows i want
x = 0
for i in range(0 , n):          # from 0 to 4
    x += 1                      # equivalent to x=x+1
    for j in range(0, i + 1):  # 0,1,2,3,4,5
        print(x , end=" ")
    print(" ")
```

```
1
```

```
2 2
3 3 3
4 4 4 4
5 5 5 5 5
```

[40]:
```
# Example 4: Method 1

rows = 5
for row in range(1, rows+1):        # from 1
    for column in range(1, row+1): # from 1,2,3,4,5
        print(column, end=" ")
    print(" ")
```

```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
```

[41]:
```
# Example 4: Method 2

for i in range (1, 6):  # rows from 1 to 5
    for j in range(i):  # column range(i)
        print (j + 1, end = ' ')
    print ()
```

```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
```

[42]:
```
# Example 5: Method 1

for i in range (1,6):
    for j in range (i,0,-1):
        print(j, end=" ")
    print(" ")

"""
i = 1 2 3 4 5

# loop 1
for i = 1, range (1,0,-1): j=1
i = 1, print: 1

# loop 2
for  i =2, range (2,0,-1): j = 2,1
```

```
i = 2, print: 2,1
"""
```

```
1
2 1
3 2 1
4 3 2 1
5 4 3 2 1
```

[42]:
```
'\ni = 1 2 3 4 5 \n\n# loop 1\nfor i = 1, range (1,0,-1): j=1\ni = 1, print: 1
\n\n# loop 2\nfor  i =2, range (2,0,-1): j = 2,1\ni = 2, print: 2,1\n'
```

[43]:
```
# Example 5: Method 2

for i in range(0,5):
    for j in range(i+1,0,-1):
        print(j, end=" ")
    print()
```

```
1
2 1
3 2 1
4 3 2 1
5 4 3 2 1
```

[44]:
```
# Example 6: Example 1 reverse pyramid


for i in range (1,6):             # rows from 1 to 5
    for j in range (5,i-1,-1):    # column range(5,0,-1) = 54321
        print ("*", end=" "),
    print (" ")
```

```
* * * * *
* * * *
* * *
* *
*
```

[45]:
```
# Example 7: Example 2 reverse pyramid Method 1

rows = 5
#        range(1,10,2)  # from 1,3,5,7,9
for i in range(rows,0,-1):  # from 3,2,1
    num = i
    for j in range(0,i):
        print(num, end=" ")
```

```
    print(" ")
```

```
5 5 5 5 5
4 4 4 4
3 3 3
2 2
1
```

[46]:
```
# Example 7: Example 2 reverse pyramid Method 2

for i in range(5,0,-1):  # range from 5 4 3 2 1
    for j in range(0,i): # range(0,5)=0 1 2 3 4
        print(i, end=" ")
    print(" ")

"""
i = 5 4 3 2 1

# loop 1
for i = 5, range (0,5): j=5 4 3 2 1
i = 5, print: 5 5 5 5 5

# loop 2
for i = 4, range (0,4): j=4 3 2 1
i = 4, print: 4 4 4 4
"""
```

```
5 5 5 5 5
4 4 4 4
3 3 3
2 2
1
```

[46]:
```
'\ni = 5 4 3 2 1 \n\n# loop 1\nfor i = 5, range (0,5): j=5 4 3 2 1\ni = 5,
print: 5 5 5 5 5\n\n# loop 2\nfor i = 4, range (0,4): j=4 3 2 1\ni = 4, print: 4
4 4 4 \n'
```

[47]:
```
# Example 8: Method 1

for i in range(5,0,-1):  # rows range = 5 4 3 2 1
    for j in range(1,i+1): # column range
        print(j, end =" ")
    print()
```

```
1 2 3 4 5
1 2 3 4
1 2 3
1 2
```

```
1
```

[48]:
```python
# Example 8: Method 2

for i in range(1,6):
    for j in range(6-i):
        print(j+1, end=" ")
    print(" ")

"""
i = 1 2 3 4 5

# loop 1
for i = 1, range (5): j=0 1 2 3 4
i = 1, print: 1 2 3 4 5

# loop 2
for i = 2, range (4): j=0 1 2 3
i = 2, print: 1 2 3 4

# loop 3
for i = 3, range (3): j=0 1 2
i = 3, print: 1 2 3

# loop 4
for i = 4, range (2): j=0 1
i = 4, print: 1 2

# loop 5
for i = 5, range (1): j=0
i = 5, print: 1
"""
```

```
1 2 3 4 5
1 2 3 4
1 2 3
1 2
1
```

[48]:
```
'\ni = 1 2 3 4 5 \n\n# loop 1\nfor i = 1, range (5): j=0 1 2 3 4\ni = 1, print:
1 2 3 4 5\n\n# loop 2\nfor i = 2, range (4): j=0 1 2 3\ni = 2, print: 1 2 3
4\n\n# loop 3\nfor i = 3, range (3): j=0 1 2\ni = 3, print: 1 2 3\n\n# loop
4\nfor i = 4, range (2): j=0 1\ni = 4, print: 1 2\n\n# loop 5\nfor i = 5, range
(1): j=0 \ni = 5, print: 1 \n'
```

[49]:
```python
# Example 9: Print following rectangle of stars of 6 rows and 3 columns
```

2.8.2 `while` loop inside `for` loop

The **while loop** is an entry-controlled loop, and a **`for`** loop is a count-controlled loop. We can also use a **`while`** loop under the for loop statement. For example,

[50]:
```
# Example 1: Print Multiplication table of a first 5 numbers using `for` loop␣
↪and `while` loop

# outer loop
for i in range(1, 6):
    print('Multiplication table of:', i)
    count = 1
    # inner loop to print multiplication table of current number
    while count < 11:
        print(i * count, end=' ')
        count = count + 1
    print('\n')
```

```
Multiplication table of: 1
1 2 3 4 5 6 7 8 9 10

Multiplication table of: 2
2 4 6 8 10 12 14 16 18 20

Multiplication table of: 3
3 6 9 12 15 18 21 24 27 30

Multiplication table of: 4
4 8 12 16 20 24 28 32 36 40

Multiplication table of: 5
5 10 15 20 25 30 35 40 45 50
```

Explanation:

- In this program, we iterate the first five numbers one by one using the outer loop and range function
- Next, in each iteration of the outer loop, we will use the inner while loop to print the multiplication table of the current number

[51]:
```
# Example 2:

names = ['Amy', 'Bella', 'Cathy']

for name in names:  # outer loop
    count = 0       # inner while loop
    while count < 5:
        print(name, end=' ')
```

```
        # increment counter
        count = count + 1
    print()
```

```
Amy Amy Amy Amy Amy
Bella Bella Bella Bella Bella
Cathy Cathy Cathy Cathy Cathy
```

2.9 `for` loop in one line

We can also formulate the **`for`** loop statement in one line to reduce the number of lines of code. For example:

[52]:
```
# Example 1: regular `for` loop code

first = [3, 6, 9]
second = [30, 60, 90]
final = []
for i in first:
    for j in second:
        final.append(i+j)
print(final)
```

```
[33, 63, 93, 36, 66, 96, 39, 69, 99]
```

[53]:
```
# Example 1: single line `for` loop code

first = [3, 6, 9]
second = [30, 60, 90]
final = [i+j for i in first for j in second]
print(final)
```

```
[33, 63, 93, 36, 66, 96, 39, 69, 99]
```

[54]:
```
# Example 2: Print the even numbers by adding 1 to the odd numbers in the list

odd = [1, 5, 7, 9]
even = [i + 1 for i in odd if i % 2 == 1]
print(even)
```

```
[2, 6, 8, 10]
```

[55]:
```
# Example 3:

final = [[x, y] for x in [30, 60, 90] for y in [60, 30, 90] if x != y]
print(final)
```

```
[[30, 60], [30, 90], [60, 30], [60, 90], [90, 60], [90, 30]]
```

2.10 Accessing the index in for loop

The **enumerate()** function is useful when we wanted to access both value and its index number or any sequence such as list or string. For example, a list is an ordered data structure that stores each item with its index number. Using the item's index number, we can access or modify its value.

Using enumerate function with a loop, we can access the list's items with their index number. The **enumerate()** adds a counter to iteration and returns it in the form of an enumerable object.

There three ways to access the index in **for** loop let's see each one by one

[56]:
```
# Example 1: Print elements of the list with its index number using the␣
 ↪`enumerate()` function

#In this program, the for loop iterates through the list and displays the
#elements along with its index number.

numbers = [4, 2, 5, 7, 8]
for i, v in enumerate(numbers):
    print('Numbers[', i, '] =', v)
```

```
Numbers[ 0 ] = 4
Numbers[ 1 ] = 2
Numbers[ 2 ] = 5
Numbers[ 3 ] = 7
Numbers[ 4 ] = 8
```

[57]:
```
# Example 2: Printing the elements of the list with its index number using the␣
 ↪`range()` function

numbers = [1, 2, 4, 6, 8]
size = len(numbers)
for i in range(size):
    print('Index:', i, " ", 'Value:', numbers[i])
```

```
Index: 0   Value: 1
Index: 1   Value: 2
Index: 2   Value: 4
Index: 3   Value: 6
Index: 4   Value: 8
```

2.11 Iterate String using `for` loop

By looping through the **string** using **for** loop, we can do lots of string operations. Let's see how to perform various string operations using a **for** loop.

[58]:
```
# Example 1: For loop with string

# Method 1:
language = 'Python'
```

```
for letter in language:
    print(letter)

# Method 2: using range() function

for i in range(len(language)):
    print(language[i])
```

```
P
y
t
h
o
n
P
y
t
h
o
n
```

[59]:
```
# Example 2: Printing the elements of a string using for loop

for i in 'Hello World':
    print(i)
```

```
H
e
l
l
o

W
o
r
l
d
```

[60]:
```
# Example 3: Access all characters of a string

name = "Alan"
for i in name:
    print(i, end=' ')
```

```
A l a n
```

[61]:
```
# Example 4: Iterate string in reverse order

name = "Alan"
for i in name[::-1]:
    print(i, end=' ')
```

```
n a l A
```

[62]:
```
# Example 5: Iterate over a particular set of characters in string

name = "Alan Watson"
for char in name[2:7:1]:
    print(char, end=' ')
```

```
a n   W a
```

[63]:
```
# Example 6: Iterate over words in a sentence using the `split()` function.

dialogue = "Remember, Red, hope is a good thing, maybe the best of things, and␣
 ↪no good thing ever dies"
# split on whitespace
for word in dialogue.split():
    print(word)
```

```
Remember,
Red,
hope
is
a
good
thing,
maybe
the
best
of
things,
and
no
good
thing
ever
dies
```

[64]:
```
# Example 7:

for ch in 'abc':
    print(ch)
```

```
total = 0
for i in range(5):  # from 0 to 4
    total += i  # total = 0+1+2+3+4 = 10
for i,j in [(1,2),(3,1)]:  # [(1),(3)]
    total += i**j  # total = 1+3 = 4
print("total =",total)
```

```
a
b
c
total = 14
```

2.12 Iterate List using `for` loop

Python list is an ordered collection of items of different data types. It means Lists are ordered by index numbers starting from 0 to the total items-1. List items are enclosed in square `[]` brackets.

Below are the few examples of Python list.

```
>>> numers = [1,2,4,6,7]
>>> players = ["Messi", "Ronaldo", "Neymar"]
```

Using a loop, we can perform various operations on the list. There are ways to iterate through elements in it. Here are some examples to help you understand better.

[65]:
```
# Example 1: Iterate over a list Method 1

numbers = [1, 2, 3, 6, 9]
for num in numbers:
    print(num)
```

```
1
2
3
6
9
```

[66]:
```
# Example 2: Iterate over a list Method 2 (list comprehension)

numbers = [1, 2, 3, 6, 9]
[print(i) for i in numbers]
```

```
1
2
3
6
9
```

[66]:
```
[None, None, None, None, None]
```

[67]:
```
# Example 3: Iterate over a list using a for loop and range.

numbers = [1, 2, 3, 6, 9]
size = len(numbers)
for i in range(size):
    print(numbers[i])
```

```
1
2
3
6
9
```

[68]:
```
# Example 4: printing the elements of a list using for loop

even_numbers = [2, 4, 6, 8, 10]  # list with 5 elements
for i in even_numbers:
    print(even_numbers)
```

```
[2, 4, 6, 8, 10]
[2, 4, 6, 8, 10]
[2, 4, 6, 8, 10]
[2, 4, 6, 8, 10]
[2, 4, 6, 8, 10]
```

[69]:
```
# Example 5: printing the elements of a list using for loop

list = [60, "HelloWorld", 90.96]
for i in list:
    print(i)
```

```
60
HelloWorld
90.96
```

[70]:
```
# Example 6: Program to find the sum of all numbers stored in a list

# List of numbers
numbers = [6, 5, 3, 8, 4, 2, 5, 6, 11]  # list with 9 elements

# variable to store the sum
sum = 0

# iterate over the list
for val in numbers:
    sum = sum+val

print("The sum is", sum)
```

```
The sum is 50
```

[71]:
```
# Example 7: Calculate the square of each number using for loop.

numbers = [1, 2, 3, 4, 5]
# iterate over each element in list num
for i in numbers:
    # ** exponent operator
    square = i ** 2
    print("Square of:", i, "is:", square)
```

```
Square of: 1 is: 1
Square of: 2 is: 4
Square of: 3 is: 9
Square of: 4 is: 16
Square of: 5 is: 25
```

Explanation: `i` iterates over the 0,1,2,3,4. Every time it takes each value and executes the algorithm inside the loop. It is also possible to iterate over a nested list illustrated below.

[72]:
```
# Example 8: Calculate the average of list of numbers

numbers = [10, 20, 30, 40, 50]

# definite iteration
# run loop 5 times because list contains 5 items
sum = 0
for i in numbers:
    sum = sum + i
list_size = len(numbers)
average = sum / list_size
print(average)
```

```
30.0
```

[73]:
```
# Example 9: Printing a list using range function

color = ['Green', 'Pink', 'Blue']  # list with total 3 elements
print(len(color))  # print length of color
for i in range(len(color)):
    print(color[i])
```

```
3
Green
Pink
Blue
```

[74]:
```
# Example 10: Printing a list using range function

list_of_lists = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]  # list with 3 elements
for list1 in list_of_lists:
        print(list1)
```

```
[1, 2, 3]
[4, 5, 6]
[7, 8, 9]
```

2.13 Iterate Tuple using `for` loop

Python list Only the difference is that list is enclosed between square bracket `[]`, tuple between parentheses `()` and we cannot change the elements of a tuple once it is assigned, i.e., **immutable** whereas we can change the elements of a list i.e., **mutable**.

Below are the few examples of Python list.

```
>>> numers = (1,2,4,6,7)
>>> players = ("Messi", "Ronaldo", "Neymar")
```

Using a loop, we can perform various operations on the tuple. There are ways to iterate through elements in it. Here are some examples to help you understand better.

[75]:
```
# Example 1: For loop with tuple

numbers = (0, 1, 2, 3, 4, 5)
for number in numbers:
    print(number)
```

```
0
1
2
3
4
5
```

2.14 Iterate Dictionary using `for` loop

Python dictionary is used to store the items in the format of **key-value** pair. It doesn't allow duplicate items. It is enclosed with `{}`. Here are some of the examples of dictionaries.

```
>>> dict1 = {1: "Apple", 2: "Ball", 3: "Cat"}
>>> dict2 = {"Antibiotics": "Penicillin", "Inventor": "Fleming", "Year": 1928}
```

There are ways to iterate through **key-value** pairs. Here are some examples to help you understand better.

[76]:
```
# Example 1: Access only the keys of the dictionary.

dict1 = {"Antibiotics": "Penicillin", "Inventor": "Fleming", "Year": 1928}
```

```
for key in dict1:
    print(key)
```

```
Antibiotics
Inventor
Year
```

[77]:
```
# Example 2: Iterate keys and values of the dictionary

dict1 = {"Vaccine": "Polio", "Inventor": "Salk", "Year": 1953}
for key in dict1:
    print(key, "->", dict1[key])
```

```
Vaccine -> Polio
Inventor -> Salk
Year -> 1953
```

[78]:
```
# Example 3: Iterate only the values the dictionary

dict1 = {"Vaccine": "Smallpox ", "Inventor": "Jenner", "Year": 1796}
for value in dict1.values():
    print(value)
```

```
Smallpox
Jenner
1796
```

[79]:
```
# Example 4: For loop with dictionary
#Looping through a dictionary gives you the key of the dictionary.

person = {
    'first_name':'Milaan',
    'last_name':'Parmar',
    'age':96,
    'country':'Finland',
    'is_marred':True,
    'skills':['Python', 'Matlab', 'R', 'C', 'C++'],
    'address':{
        'street':'Space street',
        'zipcode':'02210'
    }
}
for key in person:
    print(key)

for key, value in person.items():
    print(key, value) # this way we get both keys and values printed out
```

```
first_name
last_name
age
country
is_marred
skills
address
first_name Milaan
last_name Parmar
age 96
country Finland
is_marred True
skills ['Python', 'Matlab', 'R', 'C', 'C++']
address {'street': 'Space street', 'zipcode': '02210'}
```

2.15 Iterate Set using for loop

Python sets is an unordered collection of items. Every set element is unique (no duplicates) and must be immutable (cannot be changed).

However, a set itself is **mutable**. We can add or remove items from it.

```
>>> my_set = {1, 2, 3}
>>> my_vaccine = {"Penicillin", "Fleming", "1928"}
```

Sets can also be used to perform mathematical set operations like **union**, **intersection**, **symmetric difference**, etc.

[80]:
```
# Example 6: For loop with set

mix_fruits = {'Banana', 'Apple', 'Mango', 'Orange', 'GUava', 'Kiwi', 'Grape'}
for fruits in mix_fruits:
    print(fruits)
```

```
GUava
Banana
Grape
Kiwi
Orange
Apple
Mango
```

2.16 Exercises List

2.16.1 Exercises Level 1

1. Iterate 0 to 10 using **for** loop, do the same using **while** loop.
2. Iterate 10 to 0 using **for** loop, do the same using **while** loop.
3. Write a loop that makes seven calls to **print()**, so we get on the output the following triangle:

```
# # # # # # # #
# # # # # # # #
# # # # # # # #
# # # # # # # #
# # # # # # # #
# # # # # # # #
# # # # # # # #
# # # # # # # #
```

4. Use nested loops to create the following:

```
    #
   ###
  #####
 #######
#########
###########
#############
```

5. Print the following pattern using loops

```
0 x 0 = 0
1 x 1 = 1
2 x 2 = 4
3 x 3 = 9
4 x 4 = 16
5 x 5 = 25
6 x 6 = 36
7 x 7 = 49
8 x 8 = 64
9 x 9 = 81
10 x 10 = 100
```

6. Iterate through the list, ['Python', 'Numpy','Pandas','Scikit', 'Pytorch'] using a **for** loop and print out the items.
7. Use **for** loop to iterate from 0 to 100 and print only even numbers
8. Use **for** loop to iterate from 0 to 100 and print only odd numbers

2.16.2 Exercises Level 2

1. Use **for** loop to iterate from 0 to 100 and print the sum of all numbers.

```
The sum of all numbers is 5050.
```

1. Use **for** loop to iterate from 0 to 100 and print the sum of all evens and the sum of all odds.

```
The sum of all evens is 2550. And the sum of all odds is 2500.
```

2.16.3 Exercises Level 3

1. Go to the data folder and use the [**countries.py**] file. Loop through the countries and extract all the countries containing the word **land**.

2. This is a fruit list, [‘banana’, ‘orange’, ‘mango’, ‘lemon’] reverse the order using loop.
3. Go to the data folder and use the **[countries_data.py]** file.
    1. What are the total number of languages in the data
    2. Find the ten most spoken languages from the data
    3. Find the 10 most populated countries in the world

[ ]: