Loops in Python

```
A loop in programming is a control structure that allows you to repeat a block of code a specified number of times
or until a certain condition is met. Loops are useful for tasks that involve repetitive operations, such as
processing elements in a list or repeating an action until a user input meets certain criteria. there are two main
types of loops in python.
1. For loop
2. white loop
```

For Loop

```
For loops are used to iterate over a sequence (such as a list, tuple, or string) and execute a block of code for
each item in the sequence.
It is also known as counter loop,because a for loop is often used to repeat a block of code for a specific number of
times, with the number of iterations being specified in advance.
for loop can be used for both reading and modifying purpose with given sequence if the sequene support modifying
objects.
```

```
syntax:
    for item in sequence:
    //iplace of item we can use any name we want, the sequence here will be our list,tupe,set,dictionary or string,
    or any other iterable if there is any.
```

In [2]:

```python
#lets read elements from a list using for loop
nameList = ["Nasir","Sabir","Jillani","Jibran","GM","Chan","Jahantab","Awais"]
#lets use for loop
for name in nameList:
    print(name)
```

```
Nasir
Sabir
Jillani
Jibran
GM
Chan
Jahantab
Awais
```

```
As shown above the each individual element is accessed individually and printed individually.
Keep in mind using this approach we can only access elements of list, but can't modify the list elements.
to modify list element we use another approach.
In case you want to modify items using this technique in any scanerio use this approach (although this is not
recommended)
```

In [5]:

```python
#lets read elements from a list using for loop
nameList = ["Nasir","Sabir","Jillani","Jibran","GM","Chan","Jahantab","Awais"]
print(nameList)
#lets use for loop
for name in nameList:
   index = nameList.index(name)
   nameList[index] = "{} 2.0".format(name)
print(nameList)
```

```
['Nasir', 'Sabir', 'Jillani', 'Jibran', 'GM', 'Chan', 'Jahantab', 'Awais']
['Nasir 2.0', 'Sabir 2.0', 'Jillani 2.0', 'Jibran 2.0', 'GM 2.0', 'Chan 2.0', 'Jahantab 2.0', 'Awais 2.0']
```

```
as shown above we modified the list using above loop as well but this is not a good way actually to get index
of element, the preffered way is given below,
```

In [6]:

```python
nameList = ["Nasir","Sabir","Jillani","Jibran","GM","Chan","Jahantab","Awais"]
print(nameList)
#lets use for loop
for index in range(len(nameList)):
   nameList[index] = nameList[index] + " 2.0"
print(nameList)
```

```
['Nasir', 'Sabir', 'Jillani', 'Jibran', 'GM', 'Chan', 'Jahantab', 'Awais']
['Nasir 2.0', 'Sabir 2.0', 'Jillani 2.0', 'Jibran 2.0', 'GM 2.0', 'Chan 2.0', 'Jahantab 2.0', 'Awais 2.0']
```

```
This above given way is preferred (in my thinking) way to perform the modification in list items, because we have
direct access to index the range function create a range from 0 to given number as we provide length of list so
range(8) means it will run loop for 0 including 7 but not 8 and these are the index of list containing 8 elements
now using index we can perform any action we want on list.
```

In [7]:

```python
#let see how actually range function works
for number in range(8):
    print(number)
```

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

```
As shown above it run loop from 0 to 7 because these are the values generated by range function
```

How to access index and item at same time using for loop

We can also access index along with its item using for loop with help of enumerate() function. The enumerate function in Python returns an object of type enumerate, which is an iterator that yields pairs of the form (index, item), where index is the index of the current item in the input sequence, and item is the value of the current item.

Even though the elements produced by enumerate are pairs of values, and pairs in Python are often represented as tuples, the enumerate object itself is not a tuple. Instead, it is an iterator, which means that it implements the **next** method and can be used in a for loop or with the next function to iterate over its elements one by one.

In [24]:
```python
string = "ABCDEFGH"
list1 = list(string) #i converted a string into list
var = enumerate(list1) #return an iterator which contains pair of (index,item)
for index,item in var:
    print(index,item)
    list1[index] = item +"**"
print(list1)
```

```
0 A
1 B
2 C
3 D
4 E
5 F
6 G
7 H
['A**', 'B**', 'C**', 'D**', 'E**', 'F**', 'G**', 'H**']
```

```
As shown above we simply modified the list using for loop with enumerate object.
```

For loop with list

In [32]:
```python
fruits = ['apple', 'banana', 'cherry']
for fruit in fruits:
    print(fruit)
```

```
apple
banana
cherry
```

For loop with tuple

In [33]:
```python
numbers = (1, 2, 3)
for number in numbers:
    print(number)
```

```
1
2
3
```

For loop with range

In [35]:
```python
for i in range(10):
    print(i)
```

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

For Loop with enumerate

In [36]:
```python
fruits = ['apple', 'banana', 'cherry']
for i, fruit in enumerate(fruits):
    print(i, fruit)
```

```
0 apple
1 banana
2 cherry
```

For Loop wtih dictionary

In [37]:
```python
d = {'apple': 1, 'banana': 2, 'cherry': 3}
for key in d:
    print(key, d[key])
```

```
apple 1
banana 2
cherry 3
```

For loop with item of dictionary

In [39]:
```
d = {'apple': 1, 'banana': 2, 'cherry': 3}
print(d.items())
#d.items returns alist of tuples which contains keys and value
for key, value in d.items():
    print(key, value)
```

```
dict_items([('apple', 1), ('banana', 2), ('cherry', 3)])
apple 1
banana 2
cherry 3
```

else with for loop

In [41]:
```
#we can use else with for loop so that when the condition of for loop evaluated to false or we can say when for
#loop is finished the else command will be executed
for num in range(6):
    print(num)
else:
    print("Loop ended")
```

```
0
1
2
3
4
5
Loop ended
```

NestedLoop

In [55]:
```
#Nested loop are when one loop used inside another loop for example,
list1 = [["first","second","third"],99,["fourth","fifth","sixth"]]
for index in range(len(list1)):#iteratin the loop for no of time equal to list items
    if isinstance(list1[index],list): #a conditional statment which will be executed only when the current
        #element is object of list
        for innerIndex in range(len(list1[index])): #create another loop to iterate through inner list
            print(list1[index][innerIndex]) #printing the inner loop items
    else: #if the inner element is not a instance of list simply print that element
        print(list1[index])
```

```
first
second
third
99
fourth
fifth
sixth
```

Continue and Break statment inside for loop

In Python, the continue and break statements are used inside loops to control the flow of execution of loop.

The continue statement is used to skip the current iteration of the loop and move on to the next iteration. When a continue statement is encountered inside a loop, the rest of the code in that iteration is skipped, and the loop moves on to the next iteration. Here's an example:

In [59]:
```
for i in range(10):
    if i == 5 or i == 6 or i == 7 :
        continue
    print(i)
```

```
0
1
2
3
4
8
9
```

when the value of i was 5,6,7 there is a continue statment inside if statment which means ignore other statment inside for loop and start new iteration.

Break Statment

The break statement is used to exit the loop prematurely. When a break statement is encountered inside a loop, the loop terminates and the rest of the code in the loop is skipped. Here's an example:

In [61]:
```
for i in range(10):
    if i == 5:
        break
    print(i)
print("outside loop")
```

```
0
1
```

```
2
3
4
outside loop
```

now instead of continue we use break statment which means the loop will be exited when it reaches break statment no matter how many iterations are left behind, this is like restricting or forcefully stopping loop when specific condition meet.this is the reason when i ==5 the control flop exit from loop and statment outside loop executed.

In [ ]:

While Loop

In Python, the while loop is used to repeatedly execute a block of code as long as a specified condition is met. The basic syntax of a while loop is as follows:

```
while condition:
    # code to be executed
```

The condition is a Boolean expression that is evaluated before each iteration of the loop. If the condition is True, the code inside the loop is executed. If the condition is False, the loop terminates and the rest of the code in the program is executed. Here's an example:

In [63]:
```
count = 0
while count < 5: #the loop will execute 5 times when the value of count is 0,1,2,3,4 and when value of
    #count increases to 5 the condition of loop becomes false and control flow will move to statment after loop
    print(count)
    count += 1
```

```
0
1
2
3
4
```

It's important to be careful when using while loops, as they can cause infinite loops if the condition never becomes False. To prevent infinite loops, make sure to include a way for the condition to change so that it eventually becomes False.

We can use if else and continue and break statments in for and while loops according to our needs.

In [ ]: