1- Loops

Loops are a way to repeat a block of code for a specific number of times or until a condition is met. Python has two types of loops: for and while.

a- For loops iterate over a sequence (such as a list, tuple, string, etc.)

b- while loops execute as long as a condition is true. You can also create nested loops, which are loops inside another loop

1.1- For Loops

In [1]:
```
for number in range(5):
    print(number)
```

```
0
1
2
3
4
```

In [2]:
```
list(range(1,10,2))
```

Out[2]:
```
[1, 3, 5, 7, 9]
```

In [3]:
```
for number in [0,1,2,3,4]:
    print(number)
```

```
0
1
2
3
4
```

```
In [4]: languages = ['Swift', 'Python', 'Go', 'JavaScript', 'R']
        # run a loop for each item of the list
        for language in languages:
              print(language)
```

```
Swift
Python
Go
JavaScript
R
```

```
In [5]: a_dict = {"one":1, "two":2, "three":3}
        for key in a_dict:
            print(key)
```

```
one
two
three
```

```
In [6]: a_dict = {1:"one", 2:"two", 3:"three"}
        keys = a_dict.keys()
        sorted(keys)
        for key in keys:
            print(key)
```

```
1
2
3
```

```
In [7]: for number in range(10):
                if number % 2 == 0:
                    print(number)
```

```
0
2
4
6
8
```

2- While Loops

The while loop is also used to repeat sections of code, but instead of looping n number of times, it will only loop until a specific condition is met.

```
In [8]: i = 0
        while i < 0:
            print(i)
```

```
In [9]: i = 0
        while i < 10:
            print(i)
            i = i+1
```

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

```
In [10]: count = 0
         # run a loop until count reaches 10
         while count < 10:
             print(count)
             count = count + 1
```

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

In [ ]:
```
# while example with break
i = 5
while i < 10:
    print(i)
if i == 0:
    break
    i += 1
    # While running this code will print code infinite number of times  so beaware to run this code
```

In [ ]:
```
# while loop example with flow control
i=0
while i<10:
    if i==3:
    i+=1
continue
    print(i)
if i==5:
    break
    i+=1
    # While running this code will print code infinite number of times  so beaware to run this code
```

3- What else is for in loops

The else statement in loops only executes if the loop completes successfully. The primary use of the else statement is for searching for items

In Python, you can use an else statement after a for or while loop. The else block only runs if the loop finishes normally, without a break statement.

```
In [15]: for x in range(6):
                 print(x)
         else:
                 print("Finished!")
```

```
0
1
2
3
4
5
Finished!
```

```
In [13]: my_list = [1, 2, 3, 4, 5]
         for i in my_list:
             if i==3:
                 print("Items found!")
                 break
         else:
             print(i)
             print("Item not found!")
```

```
Items found!
```

4- Python Comprehensions

Comprehensions in Python are a way to create new sequences (such as lists, sets, dictionaries, etc.) from existing ones in a short and concise syntax.

```
In [16]: numbers = [1, 2, 3, 4]
         squares = [n**2 for n in numbers]
         print(squares)
```

```
[1, 4, 9, 16]
```

4.1- List Comprehensions

List comprehension offers a shorter syntax when you want to create a new list based on the values of anexisting list.

```
In [23]: x = [i for i in range(5)]
         x

Out[23]: [0, 1, 2, 3, 4]
```

```
In [35]: mylist = [x for x in range(0,11)]
         mylist

Out[35]: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

```
In [21]: fruits = ["apple", "banana", "cherry", "kiwi", "mango"]
         newlist = []

         for x in fruits:
             if "a" in x:
                 newlist.append(x)
         print(newlist)

         ['apple', 'banana', 'mango']
```

4.2- Casting strings into integers

```
In [26]: [int(x) for x in ["1", "2", "3"]]

Out[26]: [1, 2, 3]
```

```
In [28]: x = ['1', '2', '3', '4', '5']
         y = [int(i) for i in x]
         y

Out[28]: [1, 2, 3, 4, 5]
```

4.3- Nested List Comprehensions

```
In [30]: list = [[1,2,3], [4,5,6], [7,8,9]]
         [num for elem in list for num in elem]

Out[30]: [1, 2, 3, 4, 5, 6, 7, 8, 9]
```

4.4- Dictionary Comprehensions

Dictionary comprehension is a method for transforming one dictionary into another dictionary. During this transformation, items within the original dictionary can be conditionally included in the new dictionary and each item can be transformed as needed.

```
In [31]: print( {i: str(i) for i in range(5)} )
```

```
{0: '0', 1: '1', 2: '2', 3: '3', 4: '4'}
```

```
In [32]: my_dict = {1:"dog", 2:"cat", 3:"horse"}
         print( {value:key for key, value in my_dict.items()} )
```

```
{'dog': 1, 'cat': 2, 'horse': 3}
```

4.5- Set Comprehensions

Set comprehensions are created in much the same way as dictionary comprehensions. Now a Python set is much like a mathematical set in that it doesn't have any repeated elements.

```
In [33]: my_list = [1, 2, 2, 3, 4, 5, 5, 7, 8]
         my_set = set(my_list)
         my_set
```

```
Out[33]: {1, 2, 3, 4, 5, 7, 8}
```

```
In [34]: my_list = [1, 2, 2, 3, 4, 5, 5, 7, 8]
         my_set = {x for x in my_list}
         my_set
```

```
Out[34]: {1, 2, 3, 4, 5, 7, 8}
```

```
In [ ]:
```