List Comphersion

In [ ]:
```
List Compherension --> A Python list comprehension consists of brackets containing the expression,
                       which is executed for each element along with the for loop to iterate over each
                       element in the Python list.
                   --> Python List comprehension provides a much more short syntax for creating a new
                       list based on the values of an existing list.

Syntax:

    list = [expression for item in list if condition]
```

Example

Creation of list without list Compherension

In [1]:
```
list=[]
for i in range(1,11):
    list.append(i**2)
list
```
Out[1]:
```
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

Creation of list with list Compherension

In [2]:
```
list = [i**2 for i in range(1,11)]
list
```
Out[2]:
```
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

Creation of list without list Compherension

In [3]:
```
#[2,4,6,8,10]
list = []
for i in range(1,6):
    list.append(i*2)
list
```
Out[3]:
```
[2, 4, 6, 8, 10]
```

Creation of list with list Compherension

In [4]:
```
list=[i*2 for i in range(1,6)]
list
```
Out[4]:
```
[2, 4, 6, 8, 10]
```

Creation of list with list Compherension along with if Condition

In [1]:
```
list = [i for i in range(1,11) if i%2==0]
list
```
Out[1]:
```
[2, 4, 6, 8, 10]
```

Creation of list without list Compherension

In [8]:
```
list = ["Vijay","Abhishek","Harry"]
lis1=[]
for i in list:
    lis1.append(i[0])
lis1
```
Out[8]:
```
['V', 'A', 'H']
```

Creation of list with list Compherension

In [9]:
```
list=[i[0] for i in list]
print(list)
```
```
['V', 'A', 'H']
```

Creation of list without list Compherension along with if Condition

In [10]:
```
num1=[10,20,30,40]
num2=[30,40,50,60]
list=[]
for i in num1:
    if i in num2:
        list.append(i)
print(list)
```
```
[30, 40]
```

Creation of list without list Compherension along with if Condition

In [12]:
```
list1 = [i for i in num1 if i in num2]
list1
```
Out[12]:
```
[30, 40]
```

Creation of list without list Compherension along with loops

In [35]:
```
x=[]
y=[]
for i in range(1,4):
    for j in range(1,4):
        pass
    x.append(i)
    y.append(x)
print(y)
```
```
[[1, 2, 3], [1, 2, 3], [1, 2, 3]]
```

Creation of list with list Compherension

In [36]:
```
list1 = [[i for i in range(1,4)] for i in range(1,4) if i%2==0 ]
list1
```
Out[36]:
```
[[1, 2, 3]]
```

Taking input of a list using list compherension

In [41]:
```
list1 = [int(i) for i in input().split()]
print(list1)
```
```
10 20 30 40 50
[10, 20, 30, 40, 50]
```

EVAL Function

In [ ]:
```
Eval Function ---> The eval() function evaluates the specified expression, if the expression is a
                   legal Python statement, it will be executed.
              ---> take argument as a string it evaluate with argument and return the resptive object.

              ---> First eval fucntion is evaluating whatever things are given inside the string.
                   after evaluating whatever object is present in that eval function will return
                   answer with respect to that object.
```

Example of Eval Function

In [3]:
```
x = eval("10+20")
print(type(x))
```
```
<class 'int'>
```

In [4]:
```
x=eval("10+20j")
print(type(x))
```
```
<class 'complex'>
```

In [7]:
```
x = eval("55")
print(x)
```
```
55
```

In [10]:
```
eval("10+200//23456+9999//2334")
```
Out[10]:
```
14
```

Eval function is also used for taking input

In [9]:
```
l=eval(input("Enter : "))   #eval("[10,20,30,40,50]")
print(type(l))
```
```
Enter : [10,20,30]
<class 'list'>
```

In [46]:
```
l=eval(input("Enter : "))   #eval("{1:2,3:4,5:6}")
print(type(l))
```
```
Enter : {1:2,3:4,5:6}
<class 'dict'>
```

In [48]:
```
x= eval(input("Enter :")) #eval("{2,3:4,5:6}")
print(type(x))
```
```
Enter :{2,3:4,5:6}
Traceback (most recent call last):

  File ~\anaconda3\lib\site-packages\IPython\core\interactiveshell.py:3369 in run_code
    exec(code_obj, self.user_global_ns, self.user_ns)

  Input In [48] in <cell line: 1>
    x= eval(input("Enter :"))

  File <string>:1
    {2,3:4,5:6}
       ^
SyntaxError: invalid syntax
```

In [49]:
```
x= eval(input("Enter :")) #eval("{2,3:4,5:6}")
print(type(x))
```
```
Enter :[10,20,"Python"]
<class 'list'>
```

Copying of a list

In [ ]:
```
We can copy one list to another with the help of Aliasing and Cloning.
```

Aliasing

In [ ]:
```
Giving a new name to an existing list is called 'aliasing'.
```

Example

In [52]:
```
x=[10,20,30,40,50]
y=x
print(id(y))
print(id(x))
y[1]=200
print(y)
print(x)

Note --> The major disadvantage of aliasing is if we are going to perfvorm any
         change in the list object with one variable then because of that change the second
         variable object will also affected.
```
```
2563219441472
2563219441472
[10, 200, 30, 40, 50]
[10, 200, 30, 40, 50]
```

Cloning

In [ ]:
```
Cloning can be done by Two Ways:
    1.By slicing
    2.Copy Functions

Note --> In case of cloning a new object will be created and the content of old list will be copied
         to that new object.
```

By slicing

In [11]:
```
#By slicing
x=[10,20,30,40,50]
y=x[:]
print(id(x))
print(id(y))
x[1]=200
print(y)
print(x)
```
```
2299102206208
2299101882880
[10, 20, 30, 40, 50]
[10, 200, 30, 40, 50]
```

By Cloning

In [59]:
```
#By Cloning
x=[10,20,30,40,50]
y=x.copy()
print(id(x))
print(id(y))
y[2]=500
print(y)
print(x)
```
```
2563191501824
2563211910976
[10, 20, 500, 40, 50]
[10, 20, 30, 40, 50]
```

In [ ]:
```
aliasing --> simply creating a refernce or a variable that is pointiing to the existing object.
cloning -->  you are creating a new object with the old object content
```

Practice Question

In [ ]:
```
which compherension is not possible?
1.list compherension
2.set compherension
3.tuple compherension  #Correct
4.dictionary compherension
```

Set Compherension

In [ ]:
```
set compherension --> set compherension is also possible in Python. Set compherension is same as list
                      compherension but curly brackets we need to use for set compherension.


Syntax:
    set = {expression for item in list if condition}
```

Example

In [61]:
```
square = {x**2 for x in range(1,6)}
print(square)
```
```
{1, 4, 9, 16, 25}
```

Dictionary Compherension

In [ ]:
```
dict comphersion --> dict comphersion is also possible in Python. dict comphersion is same as list
                     compherension but curly brackets and colon we need to use for dictionary
                     compherension.
```

Example

In [12]:
```
d={}
for i in range(1,6):
    d[i]=i**2
d
```
Out[12]:
```
{1: 1, 2: 4, 3: 9, 4: 16, 5: 25}
```

In [65]:
```
dictionary = {x:x**x  for x in range(1,6)}
dictionary
```
Out[65]:
```
{1: 1, 2: 4, 3: 27, 4: 256, 5: 3125}
```

Tuple Compherension

In [ ]:
```
Tuple Compherension --> Tuple Compherension is not possible in Python. If you are trying to do
                        tuple comphersion then you will not get any error but the resultant object
                        is not the tuple object . You will get Generator Object.
```

Example

In [68]:
```
t=(x**2 for x in range(1,6))
t
```
Out[68]:
```
<generator object <genexpr> at 0x00000254CBD11CF0>
```