List comprehension
In this method, we basically define the process through which the list needs to be created. We specify what operation needs to be done and on what elements it needs to done.

Syntax:
[expression for item in iterable]

Example:
```
result=[x**2 for x in [3,4,5]]
print(result)
```

Example:
```
#[expression for item in iterable]
print([x*1 for x in [1,2,3,4]])
print([x*2 for x in [1,2,3,4]])
print([x**2 for x in [1,2,3,4]])
print([x**2 for x in [1,2,3,4] if x%2==0])
print([x**2 for x in [1,2,3,4] if x%2!=0])
```

Syntax:
[expression if conditional else stmt for item in iterable]

Example:
```
print([x if x>3 else x+1 for x in [1,2,3,4,5,6]])#[2, 3, 4, 4, 5, 6]
print([x if x>3 else x+3 for x in [1,2,3,4,5,6]])#[4, 5, 6, 4, 5, 6]
print([x if x<3 else x+3 for x in [1,2,3,4,5,6]])#[1, 2, 6, 7, 8, 9]
```

Example:
```
num=[1,2,3,4]
result=[x*10 for x in num]
print(result)
```

Example:
```
print([x for x  in "Hello12345" if x.isdigit()])
print([x for x  in "Hello12345" if x.isalpha()])
```

NOTE: If any character other than alpha, nums will be ignore...!!

Example:
```
words=['Hello',"Hai",'python']
result=[x.upper() for x in words]
print(result)
```

Example:
```
Sample_List = ["Sample", "List"]
Result = [item[0] for item in Sample_List]
print(Result)
```

Example:
```
a=[1,2,3,4,5,6]
b=[2,3]
result=[x+y for x in a for y in b]
print(result )
```

Differences between Hardcopy, shallowCopy and DeepCopy in python (Copy or Clone)

Hard Copy an Object in Python
In Python, we use = operator to create a copy of an object. You may think that this creates a new object; it doesn't. It only creates a new variable that shares the reference of the original object.

Example:
```
myList=[1,2,3,4,5,6]
print(myList)
yourList=myList
print(yourList)
myList[0]="NareshIT"
print(myList)
print(yourList)
```

In Python, there are two ways to create copies:
1 Shallow Copy or COPY
2 Deep Copy

Copy Module
We use the copy module of Python for shallow and deep copy operations. Suppose, you need to copy the compound list say x.

Example:
```
import copy
print(dir(copy))
```

Shallow Copy (copy.copy())
A shallow copy creates a new object which stores the reference of the original elements. So, a shallow copy doesn't create a copy of nested objects, instead it just copies the reference of nested objects. This means, a copy process does not recurse or create copies of nested objects itself.

Example:
```
import copy
myList=[1,2,3,4,5,6]
print(myList)
yourList=myList
print(yourList)
myList=copy.copy(yourList)
myList[4]="PYTHON"
print(myList)
print(yourList)
```

Example:
```
import copy
old_list = [[1, 1, 1], [2, 2, 2], [3, 3, 3]]
new_list = copy.copy(old_list)
old_list.append([4, 4, 4])
print("Old list:", old_list)
print("New list:", new_list)
```

Deep Copy
A deep copy constructs a new compound object and then recursively

inserts the copies into it the objects found in the original.

Syntax:
copy.deepcopy(x)

Example:
```
import copy
yourList=[1,2,3,4,['a','b','c']]
myList=copy.deepcopy(yourList)
myList[4][1]="Hello"
print(myList)
print(yourList)
```

Example:
```
import copy
ys = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
zs = copy.deepcopy(ys)
ys[1][1] = 'X'
print(ys)#[[1, 2, 3], [4, 'X', 6], [7, 8, 9]]
print(zs)#[[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

zip() in Python
The purpose of zip() is to map the similar index of multiple containers so that they can be used just using as single entity.
Python Zip and Unzip lists, tuples.

Syntax :
```
zip(*iterators)
```

Parameters : Python iterables or containers (list, string etc)
Return Value :
Returns a single iterator object, having mapped values from all the containers.

Example:
```
PyList1=["Hadoop","Spark","PYTHON","DataScience"]
PyList2=["Bigdata","Teradata","Pandas","SKLearn"]
z1=zip(PyList1,PyList2)
print(list(z1))
```

Example:
```
Pylist1=[1,2]
Pylist2=[2,3]
ziplist=zip(Pylist1,Pylist2)
ziptuple=zip(Pylist1,Pylist2)
zipset=zip(Pylist1,Pylist2)
zipdict=zip(Pylist1,Pylist2)
print(list(ziplist))
print(tuple(ziptuple))
print(set(zipset))
print(dict(zipdict))
```

Example:
```
PyList1=["Hadoop","Spark","PYTHON","DataScience"]
PyList2=["Bigdata","Teradata","Pandas","SKLearn"]
for l1,l2 in zip(PyList1,PyList2):
    print(l1,l2)
```

Example:

```
PyList1=["Hadoop","Spark","PYTHON","DataScience"]
PyList2=["Bigdata","Teradata","Pandas","SKLearn"]
Dict_Zip=dict(zip(PyList1,PyList2))
print(Dict_Zip)
```

Example:

```
PyList1=[10,20,30,40]
PyList2=[50,60,70,80]
for a,b in zip(PyList1,PyList2):
    print('{} + {}={}'.format(a,b,a+b))
```

How to unzip?

Unzipping means converting the zipped values back to the individual self as they were. This is done with the help of “*” operator.

Example:

```
PyList1=["Hadoop","Spark","PYTHON","DataScience"]
PyList2=["Bigdata","Teradata","Pandas","SKLearn"]
z1=zip(PyList1,PyList2)
z2=zip(*z1)
print(*z2)
```

Difference between del and None Keywords:

del

The variable will be removed and we cannot access that variable(unbind operation)

Example:

```
PyStr="PYTHON"
del PyStr
print(PyStr)#NameError: name 'PyStr' is not defined.
```

None:

None assignment the variable will not be removed but the corresponding object is eligible for Garbage Collection(re-bind operation).

Example:

```
PyStr="PYHON"
print(PyStr)#PYTHON
PyStr=None
print(PyStr)#None
```