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
```

WORKING WITH PYTHON ARRAYS
It is the collection of elements of a single data type, eg. array of int, array of string. In Python, there is no native array data structure. So, we use Python lists instead of an array.

Create an Array
We can create a Python array with comma separated elements between square brackets[].

How to create an array in Python?
We can make an integer array and store it to arr.
```
PyArr = [10, 20, 30, 40, 50]
```

Access elements of an Array
We can access individual elements of an array using index inside square brackets [].

Array Index
Index is the position of element in an array. In Python, arrays are zero-indexed. This means, the element's position starts with 0 instead of 1.

Example: Accessing elements of array using indexing
```
PyArr = [10, 20, 30, 40, 50]
print(PyArr[0])
```

```
print(PyArr[1])
print(PyArr[2])
```

Find length of an Array
Python arrays are just lists, so finding the length of an array is equivalent to finding length of a list in Python.

Example:
```
PyBrands=["Coke", "Apple", "Google", "Microsoft", "Toyota"]
NumBrands=len(PyBrands)
print(NumBrands)
```

Slicing of an Array
Python has a slicing feature,It allows to access pieces of an array.
[x : y]

Example:
```
PyFruits=["Apple", "Banana", "Mango", "Grapes", "Orange"]
print(PyFruits[1:4])
print(PyFruits[ :3])
print(PyFruits[-4:])
print(PyFruits[-3:-1])
```

Multi-Dimensional Arrays
It is an array within an array. This means an array holds different arrays inside it.

Example:
```
MultArr = [[1,2], [3,4], [5,6], [7,8]]
print(MultArr[0])
print(MultArr[3])
print(MultArr[2][1])
print(MultArr[3][0])
```

Python Matrix
A matrix is a two-dimensional data structure. In python, matrix is a nested list.

Example:
```
PyArr=[['Roy',80,75,85,90,95],
     ['John',75,80,75,85,100],
     ['Dave',80,80,80,90,95]]
print(PyArr[0])
print(PyArr[0][1])
print(PyArr[1][2])
print(PyArr[2][2])
```

WORKING WITH PYTHON TUPLE DATA STRUCTURE
Tuple is readonly List
OR
It is a collection that cannot be modified. A tuple is defined using parenthesis.

Advantages of Tuple over List
1 We generally use tuple for heterogeneous (different) datatypes and

list for homogeneous (similar) datatypes.
2 Since tuple are immutable, iterating through tuple is faster than with list.
3 Tuples that contain immutable elements can be used as key for a dictionary. With list, this is not possible.

More About Tuple:
1.You can't add elements to a tuple.
2.You can't remove elements from a tuple.

How to Create a tuple?
To create a tuple, just list the values within parenthesis separated by commas.

Creating Empty Tuple:
```
PyTuple = ()
print(type(PyTuple))#<class 'tuple'>
print(PyTuple)#()
```

Creating tuple having integers
```
PyTuple=(1, 2, 3)
print(type(PyTuple))#<class 'tuple'>
print(PyTuple)#(1, 2, 3)
```

Creating tuple having float values
```
PyTuple=(1.1, .1.2, 3.1)
print(type(PyTuple))#<class 'tuple'>
print(PyTuple)#(1.1, 1.2, 3.1)
```

Creating a tuple with mixed datatypes
```
PyTuple = (1, "Data Science", 3.4)
print(type(PyTuple))#<class 'tuple'>
print(PyTuple)#(1, 'Data Science', 3.4)
```

Creating a Nested tuple
```
PyTuple=("Data Science", (8, 4, 6), (1, 2, 3))
print(type(PyTuple))#<class 'tuple'>
print(PyTuple)#('Data Science', (8, 4, 6), (1, 2, 3))
```

Tuple Packing.
Creating tuple without parentheses, also called tuple packing.
```
PyTuple = 3, 4.6, "Data Science"
print(type(PyTuple))#<class 'tuple'>
print(PyTuple)#(3, 4.6, 'Data Science')
#Tuple Unpacking is also possible
a, b, c = PyTuple
print(a);print(b);print(c)
```

NOTE:
if a value is more, it dislays "Value Error",
if a variable is more, it displays "not enough values"

Creating a tuple with one element is a bit tricky.
Having one element within parentheses is not enough. We will need a trailing comma to indicate that it is in fact a tuple.

Example:

```
PyTuple = ("Data Science")

print(type(PyTuple)) #<class 'str'>
PyTuple = ("Data Science",)

print(type(PyTuple))#<class 'tuple'>
PyTuple = "Data Science",

print(type(PyTuple)) #<class 'tuple'>

Example: #List Inside Tuple
MyTuple=('a',[1,2])
print(type(MyTuple))
print(MyTuple[0])
print(MyTuple[1])
print(MyTuple[1][1])
MyTuple[1][1]='4'
print(MyTuple)

Example:
PyTuple=(1,2,3,[4,5])
print(type(PyTuple))
print(PyTuple[0])
print(PyTuple[3])
print(PyTuple[3][0])
PyTuple[3][0]=40
print(PyTuple[3])
print(PyTuple[3][1])
PyTuple[0]=100
print(PyTuple)

Tuple Membership Test:
We can test if an item exists in a tuple or not, using the keyword in.

Example:
PyTuple=(1,2,3,[4,5])
print(type(PyTuple))#<class 'tuple'>
print(1 in PyTuple)
print(10 not in PyTuple)
print(5 in PyTuple)

Iterating Through a Tuple
Using a for loop we can iterate through each item in a tuple.

Example:
for name in ('KSRaju','Dinesh',"NareshIT"):
     print("Hai",name)
```