List Data Structure:

- In any real time, project, creating, updating, retrieving, deleting elements are very common operations
- Few more real times operations are below,
    - Storing
    - Searching
    - Retrieving
    - Deleting
    - Processing
    - Duplicate
    - Ordered
    - Unordered
    - Size
    - Capacity
    - Sorting
    - Un-sorting
    - Random access
    - Keys
    - Values
    - Key – value pairs
- So, to understand above operations where to use and how to use then we need to learn about data structures.

Python Data structures:

- If you want to store a group of individual objects as a single entity, then you should go for data structures.

Sequence of elements:

- Data structure also called as sequence.
- A sequence is a datatype that can contains a group of elements.
- The purpose of any sequence is to store and process a group of elements.
- In python, strings, lists, tuples and dictionaries are very important sequence datatype.

list data structure:

- We can create list by using, square brackets [] symbols, list() predefined function.
- A list can store group of objects or elements.
- A list can store same (Homogeneous) type of elements.
- A list can store different type (Heterogeneous) of elements.
- A list size will increase dynamically
- In list insertion order is preserved or fixed.
- Duplicate elements are allowed.
- List having mutable nature.
- Store elements by using index. A list data structure supports both positive and negative indexes. Positive index means from left to right. Negative index

means right to left.
- Once if we create list object means internally object is creating for list class.

Creating list:

- We can create list by using square brackets []
- Inside list, elements will be separated by comma separator

demo.py

```
l = []
print(l)
print(type(l))
```

Output

```
[]

<class 'list'>
```

demo.py

```
numbers = [10, 20, 30, 40]
print(numbers)
```

Output

```
[10, 20, 30, 40]
```

Creating list by using list(parameter1) predefined function:

- We can create list by using list(parameter1) function
- list(parameter1) predefined function will take only one parameter.
- This parameter should be sequence (range, list, set, tuple, etc...) object otherwise we will get error.

demo.py

```
r=range(0, 10)
l=list(r)
print(l)
```

output

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Difference between list(parameter1) predefined function and list class

- list(parameter1)is predefined function in python.
- list(parameter1) predefined function is used to create a list of elements or objects.
- This function takes one parameter as sequence object.
- list is predefined class in python
- Once if we created a list then internally it represents as list class type
- represents as list class type.

demo.py

```
r=range(0, 10)
l=list(r)
print(l)
```

```
print(type(l))

output
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
<class 'list'>
```

list having mutable nature:

- Once we created a list object then we can change or modify the elements in the existing list object.
- So, list having mutable nature.

```
demo.py

l = [1, 2, 3, 4, 5]
print(l)
print("Before modifying l[0] : ",l[0])
l[0]=20
print("After modifying l[0] : ",l[0])
print(l)

output

[1, 2, 3, 4, 5]
Before modifying l[0] : 1
After modifying l[0] : 20
[20, 2, 3, 4, 5]
```

Accessing elements from list:

- We can access elements from list by using, Index , slice operator, loops.

By using index:

- index represents accessing the elements by their position numbers in the list.
- Indexing represents accessing the elements by their position numbers in the list.
- Index starts from 0 onwards.
- List supports both positive and negative indexes. Positive index represents from left to right direction. Negative index represents from right to left.
- If we are trying to access beyond the range of list index, then we will get error like IndexError

```
demo.py

names = ["Sachin", "Prasad", "Ramesh"]
print(names)
print(names[0])
print(names[1])
print(names[2])
print(type(names))
print(names[30])

output
['Sachin', 'Prasad', 'Ramesh']
Sachin
Prasad
```

```
Ramesh
<class 'list'>
IndexError: list index out of range
```

Slicing:

- Slicing represents extracting a piece of the list from already created list
- **syntax:**

```
listref[start: stop: stepsize]

 start: It indicates the index where slice can start.Default value is 0.
 stop: It indicates the index where slice can end.Default value is max allowed index of list i.e. length of the list.
Step size: increment value.Default value is 1.
```

```
demo.py

n = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
print(n)
print(n[:])
print(n[::])
print(n[0:5:])

output

[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
[1, 2, 3, 4, 5]
```

Reverse of list:

- We can get reverse of list.

```
demo
.py
l = [1, 2, 3, 4, 5]
print(l)
print(l[::-1])

output
[1, 2, 3, 4, 5]
[5, 4, 3, 2, 1]
```

Accessing list by using loops:

- We can access elements from list by using for and while loops.

```
demo.py

a = [100, 200, 300, 400]
for x in a:
    print(x)
```

```
output
100
200
300
400
```

len() function:

- By using len() predefined function we can find the length of list
- This function returns the number of elements present in the list.

```
demo.py

n = [1, 2, 3, 4, 5]
print(len(n))

Output
5
```

Methods in list data structure:

- As discussed, list is a predefined class.
- So, list class can contain methods because methods can be created inside of class only.
- We can check these methods by using dir(parameter1) predefined function.

```
demo.py

print(dir(list))

output
[
                '__add__', '__class__', '__contains__', '__delattr__',
'__delitem__', '__dir__',
          '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__',
'__getitem__',
               '__gt__', '__hash__', '__iadd__', '__imul__', '__init__',
'__init_subclass__',
          '__iter__', '__le__', '__len__', '__lt__', '__mul__', '__ne__',
'__new__',
               '__reduce__', '__reduce_ex__', '__repr__', '__reversed__',
'__rmul__',
                '__setattr__', '__setitem__', '__sizeof__', '__str__',
'__subclasshook__',
          'append', 'clear', 'copy', 'count', 'extend', 'index', 'insert',
'pop', 'remove',
          'reverse', 'sort'
          ]
```

count(parameter1)method:

- count(parameter1) is a method in list class, we should access this method by using list object.
- This method returns the number of occurrences of specified item in the list

```
demo.py

n = [1, 2, 3, 4, 5, 5, 5, 3]
```

```
print(n.count(5))
print(n.count(2))

output
3
1
```

copy(list object) method:

- Using copy method,we can get cloned list object, whose changes not reflected on original list object

```
demo.py

data1= [10,20,30]
data2=data1;
data2.push(40)
print(data1)
print(data2)

output
10,20,30,40
10,20,30,40
```

- In the above example, if you observe, we made change on data2 list object, it reflects on data1 also, it shouldn't.

```
demo.py

data1= [10,20,30]
data2=data1.copy()
data2.push(40)
print(data1)
print(data2)

output
10,20,30
10,20,30,40
```

- If you observe above example, using copy method we can get data2 object from data1, if we made changes on data2 list object, those changes never reflects on data1 list object.

append(parameter1)method:

- append(parameter1) is a method in list class, we should access this method by using list object.
- This method adds object or element to the existing list object.
- This method will add the elements to list at the end of the list.

```
demo.py

l=[]
l.append(10)
l.append(20)
l.append(30)
print(l)
```

```
output
[10, 20, 30]
```

insert(index, object) method: :

- insert(parameter1) is a method in list class, we should access this method by using list object.
- This method adds object or element to the existing list object.
- If we want to insert element to specific index position then we can insert by using insert(index, object) method

```
demo.py

n=[10,20,30,40,50]
n.insert(0, 76)
print(n)

output
[76, 10, 20, 30, 40, 50]
```

remove(parameter) method::

- remove(parameter1) is a method in list class, we should access this method by using list object.
- If we want remove a specific element from list then we can remove by using this method

```
demo.py

n=[1, 2, 3]
n.remove(1)
print(n)

output
[2, 3]
```

pop() method: :

- pop() is a method in list class, we should access this method by using list object
- This method removes and returns the last element of the list.

```
demo.py

n=[1, 2, 3]
print(n.pop())
print(n)

output
3
[1, 2]
```

reverse()::

- reverse() is a method in list class, we should access this method by using list object.
- This method reverse the order of list elements.

```
demo.py

n=[1, 2, 3, 4]
print(n)
n.reverse()
print(n)

output
[1, 2, 3, 4]
[4, 3, 2, 1]
```

sort() method::

- sort() is a method in list class, we should access this method by using list object
- In list by default insertion order is preserved.
- If we want to sort the elements of list according to default natural sorting order then we should go for sort() method.
- For numbers the default natural sorting order is ascending order
- For strings the default natural sorting order is alphabetical order

```
demo.py

n=[1, 4, 5, 2, 3]
n.sort()
print(n)
s=['Sachin', 'Ramesh', 'Arjun']
s.sort()
print(s)

output

[1, 2, 3, 4, 5]
['Arjun', 'Sachin', 'Ramesh']
```

Concatenation operator +:

- '+' operator concatenate two list objects to join them and returns single list.

```
demo.py

a= [1, 2, 3]
b= [4, 5, 6]
c = a + b
print(c)

output
[1, 2, 3, 4, 5, 6]
```

Repetition operator *:

- '*' operator works to repetition of elements in the list.

demo.py

```
a = [1, 2, 3]
print(a)
print(2*a)
```

output

```
[1, 2, 3]
[1, 2, 3, 1, 2, 3]
```

Membership operators:

- We can check if the element is a member of a list or not by using membership operators those are, in and not in.
- If the element is member of list, then in operator returns True otherwise False.
- If the element is not in the list, then not in operator returns True otherwise False

demo.py

```
x=[10, 20, 30, 40, 50]
print(20 in x) # True
print(20 not in x) # False
print(90 in x) # False
print(90 not in x) # True
```

output

```
True
False
False
True
```

Nested Lists:

- A list within another list is called nested list
- We can take a list as an element in another list.

demo.py

```
a = [80, 90]
b = [10, 20, 30, a]

print(b[0])
print(b[1])
print(b[2])
print(b[3])
```

output

```
10
20
30
[80, 90]
```

list comprehensions:

- List comprehensions represents creating new lists from Iterable object like a list, set, tuple, dictionary and range.
- List comprehensions code is very concise way.
- Here Iterable represents a list, set, tuple, dictionary or range object
- The result of list comprehension is new list based on the applying conditions.
- **syntax**list = [expression for item1 in iterable1 if statement]

```
demo.py

squares = [x**2 for x in range(1, 11)]
print(squares)

output
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```