1. Function in list

1. new_list.append(value)
   - adds the element at the end of the list
   - only one item can be add
   - if list passed, it returns the nested list e,g list = [1,2,3,[4,5,6,[7,8,9]]]
   - inplace command
2. new_list.extend((list))
   - mearge or extend the list size
   - can accept only one iterable item
   - when list passed in extend, it returns normal list e,g list = [1,2,3,4,5,6,7]

Note : - if extend list with dictionary then bydefault it will only extend keys() not values, in this case we have to extend it by dict_name.values()

3. list_name.insert(index, value)
   - values can be add or update in a spacified index position in the list
   - inplace command

2. Delete or Remove items from the list

1. list_name.remove(item)
   - inplace command
   - this method will remove that perticular item from the list
2. list_name.pop(index_num)
   - this method is not inplace command
   - we can store this modified list into the different variable
   - by default the value is -1
   - it is use for remove the item from the list

3. del list_name[index_num]

- this keyword works on the index values of items
- inplace command

4. list_name.clear()

- inplace command
- delete the all the items from the list and returns the blank list

In [ ]:
```
1
```

3. sorting a list in Assending and Dessending Order

1. list_name.sort()

- this method use for arrenge the items or values in alphabetical or accending order
- inplace command (means scope is limited)
- if the list is holdeing a int and str type values then there will be a an error called TypeError

2. list_name. sort((reverse = True))

- this method use for arreng the list of item in deccending order
- inplace command (means scope is limited)
- if the list is holdeing a int and str type values then there will be a an error called TypeError

3. sorted(list_name)

- not a inplace command (scope is no limited/ can save the items into different variable)
- if the list is holdeing a int and str type values then there will be a an error called TypeError
- sorted method tuple can also sort but it will automatically change the type of the tuple i.e it will convert tupel into list

In [17]:
```
1
```

4. Reversing a list

1. list_name[:: -1]
   - this will reverse the list
2. list_name.reverse()
   - this will reverse the string
   - inplace command
3. reversed(list_name)
   - this will reverse the string
   - NOT inplace command

5. countig and index

1. list_name.index(char/item)
   - The index() method returns the position at the first occurrence of the specified value.
   - this will return the INDEX NUMBER of that specified value
   - inplace command
2. list_name.count(chr/item)
   - returns the number of occurences of a item
   - this function will return 0 if the item is not present in the list

In [ ]:
```
1
```

6. Built in functions

iterator : - it is nothing but a set of values or set of data types e.g list, tuple, set etc...

1. max(iterator)
   - This function will return the maximum value which are present within a string
   - Not inplace function
2. min(iterator)

- This funciton will return the minumum value of the of the iterator
- Not inplace function

3. sum(iterator)

- this function will returns the addition of the iterator
- not inplace command

7. nested list

- it is a list within a list
- e.g = [1,2,3, [4,5,6], [7,8,9,[10,11,12]]]

8. copy

list_name.copy()

- The copy() method returns a copy of the specified list.

Shallow copy

- syntax
  - list_name.copy()
- Shallow Copy stores the references of objects to the original memory address.
- Shallow Copy reflects changes made to the new/copied object in the original object. if there is a nested list and its element.
- shallow copy dosen't reflects changes made to the new/copied objects in the original object. if it is not a nested element or list

DeepCopy

- syntax
  - var = copy.deepcopy(list1)
- Deep copy stores copies of the object's value.
- Deep copy doesn't reflect changes made to the new/copied object in the original object.

Difference between shallow copy and deep copy

- In Shallow copy, a copy of the original object is stored and only the reference address is finally copied.
- In Deep copy, the copy of the original object and the repetitive copies both are stored.

List comprehension

- syntax:-
  - new_list = [new_item for item in list]
  - new_list = [new_item for item in list if condition]
  - new_list = [new_ item if condition else item for item in list]

In [1]:

```
lis = [1,2,44,5,6,8]
a= max(lis)
a
```

```
44
```

In [2]:

```
li = [1,2,3, [4,5,6], [7,8,9,[10,11,12]]]


def access(li):
    for i in li:
        if type(i) != list:
            print(i)
        else:
            return access(i)
access(li)
```

```
1
2
3
4
5
6
```

In [ ]:

```

```

In [ ]:

```

```