LIST

**we use '[]' to create list

**it can have duplicate values

**it is mutable that is changes are allowed

**indexing possible so slicing is also poosible

**if we want values without any restrictions we use list

In [1]:

```
name = []
type(name)
```

Out[1]:

```
list
```

1. append()

*The append() method adds an item to the end of the list.

In [4]:

```
material = ['social','science','english','maths']
print('list before append',material)
material.append('telugu')
print('list after append',material)
```

```
list before append ['social', 'science', 'english', 'maths']
list after append ['social', 'science', 'english', 'maths', 'telugu']
```

2.clear()

*The clear() method removes all items from the list.

In [5]:

```
list_items =['pencial','pen','eraser']
print('list before clear',list_items)
list_items.clear()
print('list after clear',list_items)
```

```
list before clear ['pencial', 'pen', 'eraser']
list after clear []
```

3.copy()

*we can also use '=' to copy but it will modify the old list if we modify the new list

In [8]:

```
movies = ['sye','dhee','abcd']
new_list=movies.copy()
print('old list',movies)
print('new list',new_list)
new = movies
new.append('RRR')
print('old list',movies)
print('new list',new)
```

```
old list ['sye', 'dhee', 'abcd']
new list ['sye', 'dhee', 'abcd']
old list ['sye', 'dhee', 'abcd', 'RRR']
new list ['sye', 'dhee', 'abcd', 'RRR']
```

4.count()

*The count() method returns the number of times the specified element appears in the list.

In [11]:

```
vegtables=['tomato','onion','potato','potato']
print('count',vegtables.count('potato'))
```

```
count 2
```

5.extend()

*The extend() method adds all the elements of an iterable (list, tuple, string etc.) to the end of the list.

In [13]:

```
fruits=['apple','mango','grape']
new=['banana','kiwi']
print('list before extend',fruits)
fruits.extend(new)
print('list after extend',fruits)
```

```
list before extend ['apple', 'mango', 'grape']
list after extend ['apple', 'mango', 'grape', 'banana', 'kiwi']
```

6.index()

*The index() method returns the index of the specified element in the list.

In [14]:

```
states = ['ap','up','goa']
print('index',states.index('up'))
```

```
index 1
```

7.insert()

*it inserts element at specified position

In [15]:

```
regions = ['guntur','anantapur','hyderabad']
print('list before insert',regions)
regions.insert(1,'godavari')
print('list after insert',regions)
```

```
list before insert ['guntur', 'anantapur', 'hyderabad']
list after insert ['guntur', 'godavari', 'anantapur', 'hyderabad']
```

8.pop()

*The pop() method removes the item at the given index from the list and returns the removed item.

In [16]:

```
pairs =['trousers','jeans','scissors']
print('list before pop',pairs)
pairs.pop(1)
print('list after pop',pairs)
```

```
list before pop ['trousers', 'jeans', 'scissors']
list after pop ['trousers', 'scissors']
```

9.remove()

*The pop() method removes the item at the given index from the list and returns the removed item.

In [17]:

```
food = ['idly','dosa','upma','idly']
print('list before remove',food)
food.remove('idly')
print('list after remove',food)
```

```
list before remove ['idly', 'dosa', 'upma', 'idly']
list after remove ['dosa', 'upma', 'idly']
```

10.reverse()

*The reverse() method reverses the elements of the list.

In [18]:

```
snacks=['pani puri','pav bajji','egg bajji']
print('list before reverse',snacks)
snacks.reverse()
print('list after reverse',snacks)
```

```
list before reverse ['pani puri', 'pav bajji', 'egg bajji']
list after reverse ['egg bajji', 'pav bajji', 'pani puri']
```

11.sort()

*The sort() method sorts the elements of a given list in a specific ascending or descending order.

In [20]:

```
soaps=['mysore sandal','lux','rexona']
soaps.sort()
print('sorted list',soaps)
```

```
sorted list ['lux', 'mysore sandal', 'rexona']
```

SET

**we use keyword set to create set()

**it is inialized with {}

**duplicate values are not allowed

**indexing is not possible so slicing is not possible

** values are arranges in alphabetical order

**it is mutable that is changes are allowed

**if we want values as unique without duplicate values we use set

In [21]:

```
items =set()
type(items)
```

Out[21]:

```
set
```

1.add()

*The add() method adds a given element to a set. If the element is already present, it doesn't add any element.

In [23]:

```
material_2 = {'social','science','english','maths'}
print('set before add',material_2)
material_2.add('hindi')
print('set after add',material_2)
```

```
set before add {'maths', 'english', 'social', 'science'}
set after add {'maths', 'english', 'hindi', 'social', 'science'}
```

2.clear()

*The clear() method removes all elements from the set.

*clear() method doesn't take any parameters.

*clear() method doesn't return any value and returns a None.

In [24]:

```
list_items_2 ={'pencial','pen','eraser'}
print('set before clear',list_items_2)
list_items_2.clear()
print('set after clear',list_items_2)
```

```
set before clear {'pencial', 'eraser', 'pen'}
set after clear set()
```

3.copy()

*The copy() method returns a shallow copy of the set.

In [26]:

```
movies_2 = {'sye','dhee','abcd'}
new_list1=movies_2.copy()
print('old set',movies_2)
print('new set',new_list1)
new2 = movies_2
```

```
old set {'dhee', 'sye', 'abcd'}
new set {'dhee', 'sye', 'abcd'}
```

4.difference()

*The difference() method returns the set difference of two sets.

In [4]:

```
vegtables2={'tomato','onion','potato'}
veg={'tomat0','onoin','carrot'}
print(vegtables2.difference(veg))
print(veg.difference(vegtables2))
print(vegtables2)
```

```
{'potato', 'tomato', 'onion'}
{'tomat0', 'carrot', 'onoin'}
{'potato', 'tomato', 'onion'}
```

5.difference_update()

*The difference_update() updates the set calling difference_update() method with the difference of sets.

In [3]:

```
fruits2={'apple','mango','grape'}
fru={'apple','kiwi'}
fruits2.difference_update(fru)
print(fruits2)
print(fru)
```

```
{'mango', 'grape'}
{'kiwi', 'apple'}
```

6.discard()

*The discard() method removes a specified element from the set (if present).

*discard() method takes a single element x and removes it from the set (if present).

In [5]:

```
states2 = {'ap','up','goa'}
states2.discard('up')
print('after discard',states2)
```

```
after discard {'ap', 'goa'}
```

7.intersection

*The intersection() method returns a new set with elements that are common to all sets.

In [7]:

```
regions2 = {'guntur','anantapur','hyderabad'}
reg={'guntur','hyderbad'}
print(regions2.intersection(reg))
```

```
{'guntur'}
```

8.intersection_update()

*The intersection_update() updates the set calling intersection_update() method with the intersection of sets.

In [9]:

```
regions2 = {'guntur','anantapur','hyderabad'}
reg={'guntur','hyderbad'}
regions2.intersection_update(reg)
print(regions2)
```

```
{'guntur'}
```

9.isdisjoint()

*The isdisjoint() method returns True if two sets are disjoint sets. If not, it returns False.

In [10]:

```
pairs2 ={'trousers','jeans','scissors'}
pai={'pyjamas'}
print(pairs2.isdisjoint(pai))
```

```
True
```

10.issubset()

*The issubset() method returns True if all elements of a set are present in another set (passed as an argument). If not, it returns False.

In [11]:

```
food2 = {'idly','dosa','upma','idly'}
new_food={'idly','dosa'}
print(food2.issubset(new_food))
print(new_food.issubset(food2))
```

```
False
True
```

11.superset()

*the issuperset() method returns True if all elements of a set are present in another set. If not, it returns False

In [12]:

```
snacks2={'pani puri','pav bajji','egg bajji'}
snac={'egg bajji','pav bajji','pani puri','masala puri'}
print(snacks2.issuperset(snac))
print(snac.issuperset(snacks2))
```

```
False
True
```

12.pop()

*The pop() method returns an arbitrary (random) element from the set. Also, the set is updated and will not contain the element (which is returned).

In [13]:

```
soaps2={'mysore sandal','lux','rexona'}
print(soaps2.pop())
```

```
rexona
```

13.remove()

*The remove() method removes the specified element from the set.

In [14]:

```
days={'monday','tuesday','wednesday'}
print('set before remove',days)
days.remove('monday')
print('set after remove',days)
```

```
set before remove {'monday', 'wednesday', 'tuesday'}
set after remove {'wednesday', 'tuesday'}
```

14.symmetric_difference()

*The Python symmetric_difference() method returns the symmetric difference of two sets.

*The symmetric difference of two sets A and B is the set of elements that are in either A or B, but not in their intersection.

In [19]:

```
brands={'levis','zara','allen solly'}
new_brand={'zara','h & m'}
print(brands.symmetric_difference(new_brand))
```

```
{'levis', 'allen solly', 'h & m'}
```

15.symmetric_difference_update()

*The symmetric_difference_update() method finds the symmetric difference of two sets and updates the set calling it.

In [22]:

```
brands={'levis','zara','allen solly'}
new_brand={'zara','h & m'}
brands.symmetric_difference_update(new_brand)
print(brands)
```

```
{'levis', 'allen solly', 'h & m'}
```

16.union()

*The Python set union() method returns a new set with distinct elements from all the sets.

In [23]:

```
brands={'levis','zara','allen solly'}
new_brand={'h & m','lee'}
brands.union(new_brand)
print(brands)
```

```
{'zara', 'levis', 'allen solly'}
```

17.update()

*The Python set update() method updates the set, adding items from other iterables.

In [24]:

```
brands={'levis','zara','allen solly'}
new_brand={'h & m','lee'}
brands.update(new_brand)
print(brands)
```

```
{'zara', 'levis', 'allen solly', 'h & m', 'lee'}
```

DICTIONARY

**it is created using "[]"

**it is key-value pair

**it can have duplicate values but not duplicate keys

**we use colon between key and value left to colon is called key and right is value

**it is mutable that is changes are allowed

**if want values in dataframe we use dictionary

In [25]:

```
univ={}
type(univ)
```

Out[25]:

```
dict
```

1.clear()

*The clear() method removes all items from the dictionary.

In [28]:

```
univ={'name':'jeshu','roll_n0' : 3,'address' : 'atp'}
print('before clear',univ)
univ.clear()
print('after clear',univ)
```

```
before clear {'name': 'jeshu', 'roll_n0': 3, 'address': 'atp'}
after clear {}
```

2.copy()

*They copy() method returns a copy (shallow copy) of the dictionary.

In [29]:

```
univ={'name':'jeshu','roll_n0' : 3,'address' : 'atp'}
print('before clear',univ)
new=univ.copy()
print('new',new)
print('old',univ)
```

```
before clear {'name': 'jeshu', 'roll_n0': 3, 'address': 'atp'}
new {'name': 'jeshu', 'roll_n0': 3, 'address': 'atp'}
old {'name': 'jeshu', 'roll_n0': 3, 'address': 'atp'}
```

3.fromkeys()

*The fromkeys() method creates a new dictionary from the given sequence of elements with a value provided by the user.

In [31]:

```
keys={'apple','mango','orange'}
value='fruit'
new_=univ.fromkeys(keys,value)
print('new',new_)
```

```
new {'orange': 'fruit', 'mango': 'fruit', 'apple': 'fruit'}
```

4.get()

*The get() method returns the value for the specified key if the key is in the dictionary.

In [33]:

```
univ={'name':'jeshu','roll_n0' : 3,'address' : 'atp'}
print(univ.get('address'))
```

```
atp
```

5.items()

*The items() method returns a view object that displays a list of dictionary's (key, value) tuple pairs.

In [35]:

```
univ={'name':'jeshu','roll_n0' : 3,'address' : 'atp'}
print(univ.items())
```

```
dict_items([('name', 'jeshu'), ('roll_n0', 3), ('address', 'atp')])
```

6.keys()

*The keys() method returns a view object that displays a list of all the keys in the dictionary

In [37]:

```
univ={'name':'jeshu','roll_n0' : 3,'address' : 'atp'}
print(univ.keys())
```

```
dict_keys(['name', 'roll_n0', 'address'])
```

7.pop()

*The pop() method removes and returns an element from a dictionary having the given key.

In [39]:

```
univ1={'name':'jeshu','roll_n0' : 3,'address' : 'atp'}
print('popped value',univ1.pop('roll_n0'))
```

```
popped value 3
```

8.popitem()

*The Python popitem() method removes and returns the last element (key, value) pair inserted into the dictionary.

In [44]:

```
univ3={'name':'jeshu','roll_n0' : 3,'address' : 'atp'}
print(univ3.popitem())
```

```
('address', 'atp')
```

9.setdefault()

*The setdefault() method returns the value of a key (if the key is in dictionary). If not, it inserts key with a value to the dictionary.

In [45]:

```
univ3={'name':'jeshu','roll_n0' : 3,'address' : 'atp'}
print(univ3.setdefault('age'))
print(univ3.setdefault('name'))
```

```
None
jeshu
```

10.update()

*The update() method updates the dictionary with the elements from another dictionary object or from an iterable of key/value pairs.

In [47]:

```
univ3={'name':'jeshu','roll_n0' : 3,'address' : 'atp'}
new1={'Age' :21,'clg':'srm'}
univ3.update(new1)
print(univ3)
```

```
{'name': 'jeshu', 'roll_n0': 3, 'address': 'atp', 'Age': 21, 'clg': 'srm'}
```

11.values()

*The values() method returns a view object that displays a list of all the values in the dictionary.

In [48]:

```
univ3={'name':'jeshu','roll_n0' : 3,'address' : 'atp'}
print(univ3.values())
```

```
dict_values(['jeshu', 3, 'atp'])
```

TUPLE

**it is created using "()"

**duplicate values are allowed

**indexing is possible so slicing is possible

**immutable that is changes are not allowed

**if we don't want change values furthur we use tuple

In [49]:

```
user_data=()
type(user_data)
```

Out[49]:

```
tuple
```

1.count()

*The count() method returns the number of times the specified element appears in the tuple.

In [51]:

```
names = ('mani','venu','priya','venu','mani')
print(names.count('venu'))
```

```
2
```

2.index()

*The index() method returns the index of the specified element in the tuple.

In [52]:

```
names = ('mani','venu','priya','venu','mani')
print(names.index('mani',2,5))
```

```
4
```

In [ ]:

```
```