Tuples

- Tuples are ordered collection of data items.
- They store multiple items in a single variable.
- Tuple items are separated by commas and enclosed within round brackets ().
- Tuples are unchangeable meaning we can not alter them after creation.

In [2]:
```
t = (1,2,3)
t
```

Out[2]:
```
(1, 2, 3)
```

In [3]:
```
t[0]
```

Out[3]:
```
1
```

In [4]:
```
# replace first element with new

t[0] = 'NEW'
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Input In [4], in <cell line: 3>()
      1 # replace first element with new
----> 3 t[0] = 'NEW'

TypeError: 'tuple' object does not support item assignment
```

In [5]:
```
#Tuples with different data types

details = ("Abhijeet", 18, "Ashish", 9.8)
print(details)
```

```
('Abhijeet', 18, 'Ashish', 9.8)
```

In [6]:
```
animals = ("cat", "dog", "bat", "mouse", "pig", "horse", "donkey", "goat", "c
```

In [7]:
```
# print alternate values in each tuple

print(animals[::2])
```

```
('cat', 'bat', 'pig', 'donkey', 'cow')
```

In [8]:
```
# concatenate tuples

countries = ("Pakistan", "Afghanistan", "Bangladesh", "ShriLanka")
countries2 = ("Vietnam", "India", "China")
southEastAsia = countries + countries2
print(southEastAsia)
```

```
('Pakistan', 'Afghanistan', 'Bangladesh', 'ShriLanka', 'Vietnam', 'India',
'China')
```

Unpack Tuples

In [9]:
```
info = ("Himani", 40, "MIT")
(name, age, university) = info
print("Name:", name)
print("Age:",age)
print("Studies at:",university)
```

```
Name: Himani
Age: 40
Studies at: MIT
```

In [13]:
```
#But what if we have more number of items then the variables?

fauna = ("cat", "dog", "horse", "pig", "parrot", "salmon")
(animals,*bird, fish) = fauna
print("Animals:", animals)
print("Bird:", bird)
print("Fish:", fish)
```

```
Animals: cat
Bird: ['dog', 'horse', 'pig', 'parrot']
Fish: salmon
```

Sets

- Sets are unordered collection of data items.
- They store multiple items in a single variable.
- Sets items are separated by commas and enclosed within curly brackets {}.
- Sets are unchangeable, meaning you cannot change items of the set once created.
- Sets do not contain duplicate items.

In [16]:
```
s={1, 2, 3}
s
```

Out[16]:
```
{1, 2, 3}
```

In [17]:
```
s={1,2,3,1,2,1,2,3,3,3,3,2,2,2,1,1,2}
s
```

Out[17]:
```
{1, 2, 3}
```

In [18]:
```
info = {"Carla", 19, False, 5.9, 19}
print(info)
```

```
{False, 'Carla', 19, 5.9}
```

In [19]:
```
# Add items to set

cities = {"Pathankot", "Mohali", "Dharmpur", "Shimla"}
cities.add("Kullu")
print(cities)
```

```
{'Dharmpur', 'Mohali', 'Shimla', 'Kullu', 'Pathankot'}
```

In [20]:
```
#  use the update() method to add it into the existing set.

cities = {"Pathankot", "Mohali", "Dharmpur", "Shimla"}
cities2 = {"Moga", "Solan", "Kasuli"}
cities.update(cities2)
print(cities)
```

```
{'Dharmpur', 'Solan', 'Mohali', 'Shimla', 'Moga', 'Pathankot', 'Kasuli'}
```

In [21]:
```
# the union() method returns a new set

cities3 = cities.union(cities2)
print(cities3)
```

```
{'Dharmpur', 'Solan', 'Mohali', 'Shimla', 'Moga', 'Pathankot', 'Kasuli'}
```

The union() and update() methods prints all items that are present in the two sets. The union() method returns a new set whereas update() method adds item into the existing set from another set.

intersection and intersection_update():

The intersection() and intersection_update() methods prints only items that are similar to both the sets. The intersection() method returns a new set whereas intersection_update() method updates into the existing set from another set.

In [22]:
```
cities = {"Pathankot", "Mohali", "Dharmpur", "Shimla"}
cities2 = {"Shimla", "Solan", "Kasuli","Mohali"}
cities3 = cities.intersection(cities2)
print(cities3)
```

```
{'Mohali', 'Shimla'}
```

In [23]:
```
cities = {"Pathankot", "Mohali", "Dharmpur", "Shimla"}
cities2 = {"Shimla", "Solan", "Kasuli","Mohali"}
cities.intersection_update(cities2)
print(cities)
```

```
{'Mohali', 'Shimla'}
```

symmetric_difference and symmetric_difference_update()

The symmetric_difference() and symmetric_difference_update() methods prints only items that are not similar to both the sets. The symmetric_difference() method returns a new set whereas symmetric_difference_update() method updates into the existing set from another set.

In [24]:
```
cities = {"Pathankot", "Mohali", "Dharmpur", "Shimla"}
cities2 = {"Shimla", "Solan", "Kasuli","Mohali"}
cities3 = cities.symmetric_difference(cities2)
print(cities3)
```

```
{'Dharmpur', 'Solan', 'Pathankot', 'Kasuli'}
```

In [25]:
```
cities.symmetric_difference_update(cities2)
print(cities)
```

```
{'Dharmpur', 'Solan', 'Pathankot', 'Kasuli'}
```

difference() and difference_update()

The difference() and difference_update() methods prints only items that are only present in the original set and not in both the sets. The difference() method returns a new set whereas difference_update() method updates into the existing set from another set.

In [26]:
```
cities = {"Pathankot", "Mohali", "Dharmpur", "Shimla"}
cities2 = {"Shimla", "Solan", "Kasuli","Mohali"}
cities3 = cities.difference(cities2)
print(cities3)
```

```
{'Dharmpur', 'Pathankot'}
```

In [27]:
```
cities = {"Pathankot", "Mohali", "Dharmpur", "Shimla"}
cities2 = {"Shimla", "Solan", "Kasuli","Mohali"}
cities.difference_update(cities2)
print(cities)
```

```
{'Dharmpur', 'Pathankot'}
```

In [28]:
```
# use remove() and discard() methods to remove items form list.

cities = {"Pathankot", "Mohali", "Dharmpur", "Shimla","Kullu"}
cities.remove("Kullu")
print(cities)
```

```
{'Dharmpur', 'Mohali', 'Shimla', 'Pathankot'}
```

In [29]:
```
cities = {"Pathankot", "Mohali", "Dharmpur", "Shimla"}
cities.discard("Mohali")
print(cities)
```

```
{'Shimla', 'Dharmpur', 'Pathankot'}
```

The main difference between remove and discard is that, if we try to delete an item which is not present in set, then remove() raises an error, whereas discard() does not raise any error.

In [30]:
```
cities.discard("Mohali")
print(cities)
```

```
{'Shimla', 'Dharmpur', 'Pathankot'}
```

In [31]:
```
cities.remove("Mohali")
print(cities)
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
Input In [31], in <cell line: 1>()
----> 1 cities.remove("Mohali")
      2 print(cities)

KeyError: 'Mohali'
```

isdisjoint()

The isdisjoint() method checks if items of given set are present in another set. This method returns False if items are present, else it returns True.

In [2]:
```
#check if items of a particular set are present in another set

cities = {"Pathankot", "Mohali", "Dharmpur", "Shimla"}
cities2 = {"Moga", "Solan", "Kasuli","Pathankot"}
print(cities.isdisjoint(cities2))
```

```
False
```

issubset()

The issubset() method checks if all the items of the original set are present in the particular set. It returns True if all the items are present, else it returns False.

In [1]:
```
cities = {"Pathankot", "Mohali", "Dharmpur", "Shimla"}
cities2 = {"Pathankot","Shimla"}
print(cities2.issubset(cities))
```

```
True
```

In [ ]:
```

```