Solve the following problem statements to be solved by using Set Function

Team of Python Programmers ={"Rossum", "Mahaboob","Tara"} Team of Java Programmers={"Travis","Mahaboob","Sara"}
Q1).Find the programmers who are working in all Launguages?

In [4]:
```
pp={"Rossum", "Mahaboob","Tara"}
jp={"Travis","Mahaboob","Sara"}
```

In [6]:
```
allppjp=pp.union(jp)
print(allppjp,type(allppjp))
```

```
{'Sara', 'Tara', 'Mahaboob', 'Rossum', 'Travis'} <class 'set'>
```

Q2).Find the Programmers who are working on both Python and Java?

In [8]:
```
pp={"Rossum", "Mahaboob","Tara"}
jp={"Travis","Mahaboob","Sara"}
print(pp,type(pp))
print(jp,type(jp))
```

```
{'Rossum', 'Tara', 'Mahaboob'} <class 'set'>
{'Mahaboob', 'Sara', 'Travis'} <class 'set'>
```

In [10]:
```
bothppjp=pp.intersection(jp)
print(bothppjp,type(bothppjp))
```

```
{'Mahaboob'} <class 'set'>
```

Q3).Find the programmers who are working only on Python but not in Java?

In [12]:
```
pp={"Rossum", "Mahaboob","Tara"}
jp={"Travis","Mahaboob","Sara"}
onlypp=pp.difference(jp)
print(onlypp,type(onlypp))
```

```
{'Rossum', 'Tara'} <class 'set'>
```

Q4).Find all the Programmers who are working in Java not on Python?

In [14]:
```
pp={"Rossum", "Mahaboob","Tara"}
jp={"Travis","Mahaboob","Sara"}
onlyjp=jp.difference(pp)
print(onlyjp,type(onlyjp))
```

```
{'Sara', 'Travis'} <class 'set'>
```

Q5).Find all the Programmers who are Exclusively working on Phython as well as Java?

In [20]:
```
pp={"Rossum", "Mahaboob","Tara"}
jp={"Travis","Mahaboob","Sara"}
exclppjp=pp.symmetric_difference(jp)
print(exclppjp,type(exclppjp))
```

```
{'Rossum', 'Travis', 'Sara', 'Tara'} <class 'set'>
```

Inner OR Nested Set:

Case1:- set in set---> NOT POSSIBLE (apply indexing or modifying)

Set in set not possible to define one set to in another set because sets are unhashable type

In [3]:
```
s1={10,20,"Mahaboob Khan",{17,16,18},"MRIIRS"}
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[3], line 1
----> 1 s1={10,20,"Mahaboob Khan",{17,16,18},"MRIIRS"}

TypeError: unhashable type: 'set'
```

Case2:- list in set---> NOT POSSIBLE (apply indexing or modifying)

its not possible to define one list in other set because sets are unhashable type

In [7]:
```
s1={10,20,"Mahaboob Khan",[17,16,18],"MRIIRS"}
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[7], line 1
----> 1 s1={10,20,"Mahaboob Khan",[17,16,18],"MRIIRS"}

TypeError: unhashable type: 'list'
```

Case3:- tuple in set---> POSSIBLE

Its possible to define one tuple in other set because tuples are immutable

In [14]:
```
s1={10,20,"Mahaboob Khan",(17,16,18),"MRIIRS"}
print(s1,type(s1))
```

```
{'Mahaboob Khan', 20, 'MRIIRS', 10, (17, 16, 18)} <class 'set'>
```

In [29]:
```
for val in s1:
    print(val,type(val))# writing tuple in set is possible *
```

```
10 <class 'int'>
20 <class 'int'>
Mahaboob Khan <class 'str'>
(17, 16, 18) <class 'tuple'>
MRIIRS <class 'str'>
```

Case4:- set in list---> POSSIBLE

its possible to define one set to other list because list are mutable and allows us to locate set by using indices

In [79]:
```
lst=[10,20,"Mahaboob Khan",{17,16,18},"MRIIRS"]
print(lst,type(lst))
```

```
[10, 20, 'Mahaboob Khan', {16, 17, 18}, 'MRIIRS'] <class 'list'>
```

In [81]:
```
for val in lst:
    print(val,type(val),type(lst))
```

```
10 <class 'int'> <class 'list'>
20 <class 'int'> <class 'list'>
Mahaboob Khan <class 'str'> <class 'list'>
{16, 17, 18} <class 'set'> <class 'list'>
MRIIRS <class 'str'> <class 'list'>
```

In [85]:
```
lst[3]
```

Out[85]: {16, 17, 18}

In [89]:
```
lst[3].add("CDOE")
print(lst,type(lst))
```

```
[10, 20, 'Mahaboob Khan', {16, 17, 18, 'CDOE'}, 'MRIIRS'] <class 'list'>
```

In [7]:
```
lst=[10,20,"Mahaboob Khan",{17,16,18},"MRIIRS"]
print(lst,type(lst))
lst.append({19,20,21})
print(lst,type(lst))
```

```
[10, 20, 'Mahaboob Khan', {16, 17, 18}, 'MRIIRS'] <class 'list'>
[10, 20, 'Mahaboob Khan', {16, 17, 18}, 'MRIIRS', {19, 20, 21}] <class 'list'>
```

In [9]:
```
lst[3].remove(17)
print(lst,type(lst))
```

```
[10, 20, 'Mahaboob Khan', {16, 18}, 'MRIIRS', {19, 20, 21}] <class 'list'>
```

In [11]:
```
lst.pop(-1)
print(lst,type(lst))
```

```
[10, 20, 'Mahaboob Khan', {16, 18}, 'MRIIRS'] <class 'list'>
```

In [19]:
```
lst[3].clear() # index No:3 is empty set()
print(lst,type(lst))
```

```
[10, 20, 'Mahaboob Khan', set(), 'MRIIRS'] <class 'list'>
```

Case5:- set in tuple---> POSSIBLE

It possible to define one set in aother tuple because tuples are immutable and allow us to locate set objects by using indices

In [22]:
```
tpl=(10,"Khan",{20,30,40},"MRIIRS")
print(tpl,type(tpl))
```

```
(10, 'Khan', {40, 20, 30}, 'MRIIRS') <class 'tuple'>
```

In [24]:
```
for val in tpl:
    print(val,type(val),type(tpl))
```

```
10 <class 'int'> <class 'tuple'>
Khan <class 'str'> <class 'tuple'>
{40, 20, 30} <class 'set'> <class 'tuple'>
MRIIRS <class 'str'> <class 'tuple'>
```

In [28]:
```
tpl[2].add(100)
print(tpl,type(tpl))
```

```
(10, 'Khan', {40, 100, 20, 30}, 'MRIIRS') <class 'tuple'>
```

In [34]:
```
tpl.pop(-2)
print(tpl,type(tpl))
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[34], line 1
----> 1 tpl.pop(-2)
      2 print(tpl,type(tpl))

AttributeError: 'tuple' object has no attribute 'pop'
```

In [36]:
```
tpl[-2].clear()
print(tpl,type(tpl))
```

```
(10, 'Khan', set(), 'MRIIRS') <class 'tuple'>
```

** Frozenset **

Frozenset is one of the pre-defined class and treated as set data type The purpose of frozenset data type is that "To store multiple values either simlar type or different type or both the types in in single object with unique values only" (duplicates are not allowed) The elements of frozenset must be obtained from different objects like set,tuple and list etc.,

syntax: frozensetobj=frozenset(set/list/tuple/str/bytes/bytearray/range)

--> An object of frozenset never maintains insertion order because PVM can display any one of the possibilty of elements of frozenset object

--> On object of frozenset belongs to immutable because frozenset object does not support item assignment and not possible to modify,change and add

--> We can create two types of frozenset objects:

1. Empty frozenset

2. Non-Empty frozenset

In [40]:
```
s1={10,20,30,40,50}
print(s1,type(s1))
```

```
{50, 20, 40, 10, 30} <class 'set'>
```

In [42]:
```
fs=frozenset(s1)
print(fs,type(fs))
```

```
frozenset({50, 20, 40, 10, 30}) <class 'frozenset'>
```

In [48]:
```
s2=("MRIIRS")
print(s2,type(s2))
```

```
MRIIRS <class 'str'>
```

In [50]:
```
fs=frozenset(s2) # eliminated the duplicate elements
print(fs,type(fs))
```

```
frozenset({'R', 'M', 'S', 'I'}) <class 'frozenset'>
```

In [58]:
```
lst=["Mahaboob","CDOE","MRIIRS","khan"]
fs=frozenset(lst)
print(fs,type(fs))
```

```
frozenset({'Mahaboob', 'CDOE', 'MRIIRS', 'khan'}) <class 'frozenset'>
```

In [60]:
```
fs=frozenset((10,20,10,20,30,10,))
print(fs,type(fs))
```

```
frozenset({10, 20, 30}) <class 'frozenset'>
```

In [62]:
```
a=(10)
fs=frozenset((a,))
print(fs,type(fs))
```

```
frozenset({10}) <class 'frozenset'>
```

In [64]:
```
fs[10]=100
#frozenset object does not support item assignment and not possible to modify,chang
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[64], line 1
----> 1 fs[10]=100

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

In [68]:
```
fs.add(100)
#frozenset object does not support item assignment and not possible to modify,chang
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[68], line 1
----> 1 fs.add(100)

AttributeError: 'frozenset' object has no attribute 'add'
```