WORKING WITH PYTHON SET DATASTRUCTURE..!!

Python Sets
SET is an unordered collection of unique elements

How to create a set?
1 Insertion order is not preserved.But we can sort the elements.
2 Indexing and slicing not allowed for the set.
3 Heterogeneous elements are allowed.
4 We can represent set elements within curly braces and with comma seperation
5 We can apply mathematical operations like union,intersection,difference etc

There are currently two built-in set types:
1 set ==> The set type is mutable
2 frozenset. ==> It is immutable and hashable

Example:
```
FruitBasket={"Apple","Mango","orange","banana","Apple","Mango"}
print(type(FruitBasket))
print(FruitBasket)
```

Example:
We should not use empty curly brackets, treat as dict.!
```
PySet={}
print(type(PySet)) #<class 'dict'>
```

Set not allowed duplicates but list allows:
```
Numbers=[1,2,3,4,1,2,3,4,5]
UniqueNumbers=set(Numbers)
print(UniqueNumbers) #{1,2,3,4,5}
```

Iteration Over Sets
We can move over each of the items in a set using a loop. However, since sets are unorded

Example:
```
NumSet = set([0, 1, 2, 3, 4, 5])
for n in NumSet:
  print(n)
```

Different Python Set Methods

1 add()	2 clear()
3 copy()	4 difference()
5 difference_update()	6 discard()
7 intersection()	8 intersection_update()
9 isdisjoint()	10 issubset()
11 issuperset()	12 pop()
13 remove()	14 symmetric_difference()
15 symmetric_difference_update()	
16 union()	17 update()

Python Set add():
It adds a given element to a set. If the element is already present, it doesn't add that element.

Syntax:

```
set.add(elem)
```

Example:

```
PySet={}
print(PySet)
PySet.add(1)
print(PySet)#AttributeError: 'dict' object has no attribute 'add'
```

Example:

```
PySet={1}
print(PySet)#{1}
PySet.add(1)
print(PySet)#{1}
```

Example:

```
PySet={1}
print(PySet)
PySet.add(1,2)
print(PySet)
#TypeError: add() takes exactly one argument (2 given)
```

Example:

```
PySet={1}
print(PySet)
PySet.add([1,2])
print(PySet)
#TypeError: unhashable type: 'list'
```

Example:

```
PySet={1}
print(PySet)#{1}
PySet.add((1,2))
print(PySet)#{(1, 2), 1}
```

clear():
It is used to clear all elements from a set.

Syntax:

```
set.clear()
```

Example:

```
PySet={1}
print(PySet)#{1}
PySet.clear()
print(PySet)#set()
```

Python Set update():
It adds elements from a set to the set.
OR
A |= B or A.update(B)
Adds all elements of array B to the set A.

Syntax

```
A.update(B)
```

NOTE:
A and B are two sets. The elements of set B are added to the set A.

Example:
```
DataSet=set()
DataSet.update(["Data Science", "Big Data"])
print(DataSet)
```

Example:
```
A={10,20,30}; B=[40,50,60,10]
A.update(B,range(5))
print(A)
```

Python Set pop()
It removes an arbitrary element from the set and returns the element removed.

Syntax:
```
set.pop()
```

Example:
```
A ={'a', 'b', 'c', 'd'}
print('Return Value is', A.pop())
print(A)
```

Example:
```
NumSet = set([0, 1, 2, 3, 4, 5])
NumSet.pop()
print(NumSet)
NumSet.pop()
print(NumSet)
```

Python Set remove()
It searches for the given element in the set and removes it, but not a member of the set, a KeyError will be raised.

Syntax:
```
set.remove(element)
```

Example:
```
NumSet = set([0, 1, 2, 3, 4, 5])
NumSet.remove(0)
print(NumSet)
```

Trying to Delete Element That Doesn't Exist
Example:
```
PyPets = {'cat', 'dog', 'rabbit'}
PyPets.remove('cow')
print(PyPets)#KeyError
```

Python Set discard()
An element will be removed from the set, if it is contained in the set. If not a member of the set, nothing will be done.

Syntax:
```
s.discard(x)
```

Example:

```
NumSet = set([0, 1, 2, 3, 4, 5])
NumSet.discard(3)
print(NumSet)
```

Example:

```
PyNums={2, 3, 4, 5}
PyNums.discard(3)
print(PyNums)
PyNums.discard(10)
print(PyNums)
```

Example:

```
# Removing the elements of the set
# remove(), discard() and pop() can be used to remove the elements of
a set
# If the element to be removed is not present in the set, remove()
will raise an error while discard() will not
# pop() removes the last item from the set. A set is unordered, so the
element which will be removed is not known
Colour = {"Black", "Blue", "Red", "Green", "Orange", "Violet"}
Colour.remove("Blue")
Colour.discard("Green")
print(Colour)
x = Colour.pop()
print(x)
print(Colour)
```

intersection(s) (AND)
A set with all the elements which are contained in both sets is returned.

Syntax:

```
set1.intersetction(set2)
```

Example:

```
x = {"a","b","c","d","e"}
y = {"c","d","e","f","g"}
print(x.intersection(y))
print(x & y) #ampersand operator "&":
```

A &= B or A.intersection_update(B)
The intersection of two or more sets is the set of elements which are common to all sets.

Sytnax:

```
A.intersection_update(*Other_sets)
```

Example:

```
A = {1, 2, 3, 4}
B = {2, 3, 4, 5}
A.intersection_update(B)
print(A)
print(B)
```

Union of sets (OR)
A | B or A.union(B)   Returns a set which is the union of sets A and

B.

Syntax:
```
set1.union(set2)
```

Example:
```
PySetx=set(["Big Data", "Data Science"])
PySety=set(["Data Science", "Spark"])
PySeta = PySetx |PySety  #Union
print (PySeta)
```

Set difference() In set1 but not set2

Syntax
```
A.difference(B)
```

Example:
```
A = {1, 2, 3, 4}
B = {2, 3, 9}
print(A.difference(B))
print(B.difference(A))
```

Example:
```
A = {'a', 'b', 'c', 'd'}
B = {'c', 'f', 'g'}
print(A.difference(B))# Equivalent to A-B
print(B.difference(A))# Equivalent to B-A
```

A -= B or A.difference_update(B)
Removes all elements of B from the set A.

Syntax:
```
A.difference_update(B)
```

Example:
```
A = {'a', 'c', 'g', 'd'}
B = {'c', 'f', 'g'}
A.difference_update(B)
print(A)
```

Set symmetric_difference() (XOR)
It returns a new set which is the symmetric difference of two sets.
The symmetric difference of two sets A and B is the set of elements which are in either of the sets A or B but not in both. (A-B)UNION(B-A)

Syntax:
```
A.symmetric_difference(B)
```

Example:
```
A = {'a', 'b', 'c', 'd'}
B = {'c', 'd', 'e' }
print(A.symmetric_difference(B))
print(B.symmetric_difference(A))
```

Example:
```
PySetx=set(["Big Data", "Data Science"])
```

```
PySety=set(["Data Science", "Spark"])
PySetc=PySetx.symmetric_difference(PySety)
print(PySetc)
PySetd=PySetx^PySety
print(PySetd)
```

A ^= B or A.symmetric_difference_update(B)
Writes in A the symmetric difference of sets A and B.

Syntax:
```
A.symmetric_difference_update(B)
```

Example:
```
A = {'a', 'c', 'd'}
B = {'c', 'd', 'e' }
A.symmetric_difference_update(B)
print(A)
```

Set issuperset(): set1 contains set2

Syntax:
```
A.issuperset(B) or A >= B
```

Example:
```
A = {1, 2, 3, 4, 5}
B = {1, 2, 3}
C = {1, 2, 3}
print(A.issuperset(B))
print(B.issuperset(A))
print(C.issuperset(B))
```

Set issubset() set2 contains set1

Syntax:
```
A.issubset(B) or A <= B
```

Example:
```
A = {1, 2, 3, 4, 5}
B = {1, 2, 3}
C = {1, 2, 3}
print(A.issubset(B))
print(B.issubset(A))
print(C.issubset(B))
```

isdisjoint()
It returns True, if two sets have a null intersection.

Syntax:
```
set_a.isdisjoint(set_b)
```

Example:
```
A = {1, 2, 3, 4}
B = {5, 6, 7}
C = {4, 5, 6}
print(A.isdisjoint(B))
print(A.isdisjoint(C))
```

Set copy(): It returns a shallow copy of the set.

Syntax:
```
set.copy()
```

Example:
```
PySetx = set(["Big Data", "Data Science"])
PySetd = PySetx.copy()
print(PySetd)
```

Special Operators with set (Membership operators)
```
x={"a","b","c","d","e"}
print("a" in x)
print("d" in x)
```

Iterations with for loop:
```
x={"a","b","c","d","e"}
for i in x:
    print(i)
```

Built-in Functions with Set
```
all()   any()           enumerate()     len()
max()   min()           sorted()
```

Example:
```
PySet={4,5,8,2.2}
print(PySet)#{8, 2.2, 4, 5}
print(sorted(PySet))#[2.2, 4, 5, 8]
```

Python Frozenset
It is a new class that has the characteristics of a set, but its elements cannot be changed once assigned. While tuples are immutable lists, frozensets are immutable sets. Frozensets can be created using the function frozenset().

Syntax:
```
PyFSet=frozenset(iterable)
```

Example:
```
PyFSet1=frozenset([1,2,3,4,5])
print(type(PyFSet1))
print(PyFSet1)
PyFSet2=frozenset({2,3,4,5,6})
print(type(PyFSet2))
print(PyFSet2)
print(PyFSet1.copy())
print(PyFSet2.copy())
print(PyFSet1.difference(PyFSet2))
print(PyFSet1.intersection(PyFSet2))
print(PyFSet1.isdisjoint(PyFSet2))
print(PyFSet1.issubset(PyFSet2))
print(PyFSet1.issuperset(PyFSet2))
print(PyFSet2.symmetric_difference(PyFSet1))
print(PyFSet2.union(PyFSet1))
```

frozensets have less methods than sets.
There are some operators similar to sets intersection(), union(),

symmetric_difference(), difference(), issubset(), isdisjoint(), issuperset()) and a copy() method.

Example:

```
a = frozenset([1, 2, 3])
b = frozenset([2, 3, 4])
print(a.union(b))
print(a & b)
print(a.intersection(b))
print(a.symmetric_difference(b))
print(a.issubset(b))
print(a.issuperset(b))
```

Set Comprehension: Set comprehension is possible.
Example:

```
PySet={x*x for x in range(5)}
print (PySet)
PySet={2**x for x in range(2,10,2)}
print (PySet)
```

NOTE: Set Objects won't support indexing and slicing: