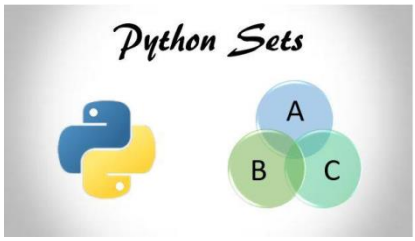

Unit 5

What is set?

- A Python set is an unordered collection of comma separated elements inside curly braces '{ }'.
- Each element of set is separated by comma.
- Elements of set enclosed within { } curly braces.
- Set in python is mutable but element of it is immutable mean we cannot change its element by passing index value.
- Like list, tuple or string element of set is not access by using index. We can only print whole set or we can iterate set using loop.
- **Unlike tuple or list, Python sets don't allow multiple occurrences of the same element. Hence, each element in Python set is unique.**

There are two types of built-in set types:

- **set**: Python set type is mutable
- **frozen set**: Python frozen set type is immutable

How to declare set?

- By placing element inside {} curly braces. Each element must separate by comma or you can also declare set by using set() built in function.

```
>>> myset={1,2,3,4,5,6,1,4}
>>> print(myset)
{1, 2, 3, 4, 5, 6}
```

In above example you can see how we create set. But as per its definition it not allow duplicate element.

```
>>> myset2=set("hello")
>>> print(myset2)
{'l', 'e', 'h', 'o'}
```

In above example you can see by using set() function we create set and again as per its rule no repetition of element. You can also see the surprising order while you print set as its unordered as per definition.

- **creating set from string**
  >>> py_set = ('Python Sets')
  >>> print (py_set)
  {'e','y','S','P','s','h','n','o',' ','t'} **// here you can see it take white space as element and also check order of it.**

PLEASE TRY EVERY EXAMPLE ON YOUR MACHINE ALSO.

- **You cannot declare set by using mutable item like in below example we declare set by enclosing list.**

  >>> myset={1,2,[10,20,30]} **//it gives below error**
  Traceback (most recent call last):
  File "<pyshell#0>", line 1, in <module>
  myset={1,2,[10,20,30]}
  TypeError: unhashable type: 'list'

- **Even you cannot declare set within set. Mean nested set is not allowed please see the below example.**
  >>> myset={1,2,{10,20,30}} **//it gives below error**
  Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
  myset={1,2,{10,20,30}}
  TypeError: unhashable type: 'set'

- **You can declare set from list by using set() in built function like**
  >>> myset=set([1,2,3,4])
  >>> print(myset)
  {1, 2, 3, 4}

- **creating set from a tuple**
  >>> py_set = set(('a','b','c','d'))
  >>> print (py_set)
  {'a','b','c','d'}

Declare Empty set

- You can declare empty set by using set() inbuilt function like,
  >>> myset=set()
  >>> print(myset)

set()
>>> type(myset)
<class 'set'> **when you print set type it gives class set**

- You cannot declare empty set like below,
  >>> myset={}
  >>> print(myset)
  {}
  >>> type(myset)
  <class 'dict'> **when you print set type it gives class dict.**

Accessing Element of python set

- As set is unordered collection of element so we cannot access its element by using its index value like list, tuple or string.
- So, we can access list by using loop.
  >>> myset={1,2,4,12,3,10,15,11}
  >>> print(myset)
  {1, 2, 3, 4, 10, 11, 12, 15} **//output**
  >>> for i in myset:
  print(i, end=" ")
  1 2 3 4 10 11 12 15 **//output (also see order of element while we print it.)**
- **We cannot do slicing operation on set as it's not access by using index value.**

Add new items in a Python set

- Indexing have no meaning. We cannot access or change an element of set using indexing or slicing.
- We can add single elements using the method **add() method.**
- Multiple elements can be added using **update() method.**
- **Duplicate items are automatically avoided while adding new items.**
  >>> myset={'a','v',10,2,20,5,'b','x',10}
  >>> print(myset)
  {2, 5, 'a', 10, 'v', 20, 'b', 'x'}
  >>> myset.add('y') **adding single element using add() method**
  >>> print(myset)
  {2, 5, 'a', 'y', 10, 'v', 20, 'b', 'x'}
  >>> myset.update([1,2,'n',5]) **adding multiple element using update() method**
  >>> print(myset)
  {1, 2, 5, 'a', 'y', 10, 'n', 'v', 20, 'b', 'x'} **[PLEASE CHECK ORDER OF ELEMENT, ITS RANDOM]**

Remove or Delete Element from Python set

- To remove or delete items from set here we have three methods **discard()**, **remove()** and **pop().**
- But pop() method is remove arbitrary element.
- Where by using discard() and remove() method you can remove specific element.
- To remove all the element from set use **clear() method.**
- Let's see it by using example.

```
>>> my_set = {1, 3, 4, 5, 6}
>>> my_set.discard(4)
>>> my_set
{1, 3, 5, 6}
>>> my_set.remove(6)
>>> my_set
{1, 3, 5}
>>> my_set.discard(2)
>>> my_set
{1, 3, 5}
>>> my_set.remove(2)
...
KeyError: 2
```

- **Example of pop() and clear() method**

```
>>> my_set = set("HelloWorld")
>>> my_set.pop()
'r'
>>> my_set.pop()
'W'
>>> my_set
{'d', 'e', 'H', 'o', 'l'}
>>> my_set.clear()
>>> my_set
set()
```

Copy a set in Python

- To copy a Python set or its items, Python provides built-in function copy( ). It creates a shallow copy and returns it.

```
>>> py_set1 = set([1,2,3])
>>> #to copy items of py_set1 to py_set
>>> py_set = py_set1.copy()
>>> print (py_set)
```

Frozen set in Python

- The frozen sets are the immutable form of the normal sets, i.e., the items of the frozen set cannot be changed and therefore it can be used as a key in dictionary.

- The elements of the frozen set cannot be changed after the creation. We cannot change or append the content of the frozen sets by using the methods like add() or remove().
- The frozenset() method is used to create the frozenset object. The iterable sequence is passed into this method which is converted into the frozen set as a return type of the method.
- supports methods like copy(), difference(), intersection(), isdisjoint(), issubset(), issuperset(), symmetric_difference() and union().

```
>>>set1 = frozenset([1,2,3,4])
>>>set2 = frozenset(['a',1,2,4,'b'])
 >>>set1|set2
Output : frozenset(['a', 1, 2, 3, 4, 'b'])
>>>set1.isdisjoint(set2)
Output : False
>>>set1.difference(set2)
Output : frozenset([3])
```

Try this : set1.add('j') ????

Build in Set Methods in Python

Method	Description
** Here set is your set name like myset.add() and so on…	
set.add( )	Add single element to set.
set.clear( )	Remove all the items from the set.
set.copy( )	Returns a shallow copy of the set. Used for copying.
set.difference( )	Returns the difference of two or more sets.
set.discard( )	Removes an element from a set if it exist in the set.
set.intersection( )	Returns the set of common elements between sets i.e. intersection.
set.disjoint( )	Returns true if the sets don't have elements in common i.e. null intersection.
set.issubset( )	Returns true if another set contains all the elements of this set.
set.issuperset( )	Returns true if this set contains all the elements of another set.
set.pop( )	Returns and removes an arbitrary item from the set. Raises error if the set is empty.
set.remove(x)	Removes the element x from the set. If x doesn't exist in the set, it raises error.
set.symmetric_difference( )	Performs symmetric difference and returns the set of all the members of the sets involved excluding the common elements.

set.union(x)	Performs and the returns the union of the sets as a set.
set.update( )	Adds multiple items in the set.

Operator for Python Set

A | B **A.union(B)**	Returns a set which is the union of sets `A` and `B`.
A |= B **A.update(B)**	Adds all elements of array `B` to the set `A`.
A & B **A.intersection(B)**	Returns a set which is the intersection of sets `A` and `B`.
A &= B **A.intersection_update(B)**	Leaves in the set `A` only items that belong to the set `B`.
A - B **A.difference(B)**	Returns the set difference of `A` and `B` (the elements included in `A`, but not included in `B`).
A -= B **A.difference_update(B)**	Removes all elements of `B` from the set `A`.
A ^ B **A.symmetric_difference(B)**	Returns the symmetric difference of sets `A` and `B` (the elements belonging to either `A` or `B`, but not to both sets simultaneously).
A ^= B **A.symmetric_difference_update(B)**	Writes in `A` the symmetric difference of sets `A` and `B`.
A <= B **A.issubset(B)**	Returns `true` if `A` is a subset of `B`.
A >= B **A.issuperset(B)**	Returns `true` if `B` is a subset of `A`.
A < B	Equivalent to `A <= B and A != B`
A > B	Equivalent to `A >= B and A != B`

Mathematical Operation on Python Set

- In python set operations like union, intersection, difference and symmetric difference can do this with operators or methods.

Union Operation

- The union of two sets are calculated by using the **or (|)** operator.
- The union of the two sets contains the all the items that are present in both the sets.
- Same can be accomplished using the **union() method.**
- Check out below example with and without using **union() method.**

```
>>> seta={1,2,3}
>>> setb={3,5,6}
>>> print(seta|setb)  #using or operator
```

```
{1, 2, 3, 5, 6}
>>> print(seta.union(setb))
{1, 2, 3, 5, 6}
>>> print(setb.union(seta)) # using union() method
{1, 2, 3, 5, 6}
```

Note: you can also see while doing union operation there is no repetition of element.

Intersection Operation

- This operation on python set gives elements that are common in both sets. Intersection is performed using **& operator.**
- Same can be accomplished using the method **intersection().**

```
>>> seta={1,2,3}
>>> setb={3,5,6}
>>> print(seta&setb) using & operator
{3}
>>> print(seta.intersection(setb))   using intersection() method
{3}
>>> print(setb.intersection(seta))
{3}
>>> seta&setb
{3}
```

Set difference operation

- (A - B) is a set of elements that are only in A but not in B. Similarly, B - A is a set of element in B but not in A.
- Difference is performed using - operator.
- Same can be accomplished using the method **difference()**.

```
>>> seta={1,2,3}
>>> setb={3,5,6}
>>> print(seta-setb)
{1, 2}
>>> print(setb-seta)
{5, 6}
>>> print(seta.difference(setb))
{1, 2}
>>> print(setb.difference(seta))
{5, 6}
```

Let's Discuss some method of set in Python

intersection_update() method

- Removes the items from the original set that are not present in both the sets (all the sets if more than one are specified).
- **Intersection_update()** method is different from **intersection()** method since it modifies the original set by removing the unwanted items, on the other hand, **intersection()** method returns a new set.

```
>>> set1={1,2,4,5}
>>> set2={2,6,7,8}
>>> set3={10,5,2,20}
>>> set1.intersection_update(set2,set3)
>>> print(set1)
{2}
```

- **You can in above example only 2 is common for all three set so while you perform intersection_update() method it will remove all other element which not in other two sets. And only 2 is remain in set1**
- **One more example**

```
a = {"ayush", "bob", "castle"}
b = {"castle", "dude", "emyway"}
c = {"fuson", "gaurav", "castle"}
a.intersection_update(b, c)
print(a)   //what is output here?
```

diffrence_update() method

- It modifies this set by removing all the items that are also present in the specified sets.
- Mean it will modify the actual set by only those item which is not present in other two sets.

```
>>> set1={1,2,4,5}
>>> set2={2,6,7,8}
>>> set3={10,5,2,20}
>>> set1.difference_update(set2,set3)
>>> print(set1)
{1, 4}
```

- **You can see in above example only 1 and 4 remain present because its not present in other two sets. So it modify actual set1.**

```
a = {"ayush", "bob", "castle"}
b = {"castle", "dude", "emyway"}
c = {"fuson", "gaurav", "castle","bob"}
```

```
a.difference_update(b, c)
print(a)   //what is output here?
```

isdisjoint() method

- It will return Boolean value true or false if intersection of two set is empty set.
- Mean there no common element present in two set than true else false

```
>>> set1={1,2,3,5}
>>> set2={7,6,10,4}
>>> set1.isdisjoint(set2)
True
```

issubset() method

- Return Boolean value true or false if all element is present in second set.
- Mean it will return true if all element of set A is present in set B.

```
>>> set1={1,2,4,5}
>>> set2={10,2,6,1,33,5,4}
>>> set1.issubset(set2)
True
>>> set2.issubset(set1)
False
```

- **Order of element not consider it only check present of element into other set. So here in above example all element or set1 is present in set2.**

issuperset() method

- The issuperset() method returns True if a set has every elements of another set.
- True if A is a superset of B
- False if A is not a superset of B

```
>>>A = {1, 2, 3, 4, 5}
>>>B = {1, 2, 3}
>>>C = {1, 2, 3}
# Returns True
>>>print(A.issuperset(B))
# Returns False
>>>print(B.issuperset(A))
# Returns True
>>>print(C.issuperset(B))
```

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.

```
>>> set1={'a','b','c'}
>>> set2={'e','g','a'}
>>> set1.symmetric_difference(set2)
{'e', 'c', 'g', 'b'}
>>> print(set1)
{'a', 'c', 'b'}
>>> print(set2)
{'e', 'a', 'g'}
```

For more method with example please visit below link
https://www.programiz.com/python-programming/methods/set

Difference between discard() and remove()

- Despite the fact that discard() and remove() method both perform the same task, There is one main difference between discard() and remove().
- If the key to be deleted from the set using discard() doesn't exist in the set, the python will not give the error. The program maintains its control flow.
- On the other hand, if the item to be deleted from the set using remove() doesn't exist in the set, the python will give the error.
- Please see the below example.

```
Months = set(["January","February", "March", "April", "May", "June"])
print("\nprinting the original set ... ")
print(Months)
print("\nRemoving items through discard() method...");
Months.discard("Feb"); #will not give an error although the key feb is not available in
 the set
print("\nprinting the modified set...")
print(Months)
print("\nRemoving items through remove() method...");
Months.remove("Jan") #will give an error as the key jan is not available in the set.
print("\nPrinting the modified set...")
print(Months)
```