SETS

A set in python is a data structure that contains an unordered collection of unique and immutable objects.

Set in Python is a data structure equivalent to sets in mathematics. It may consist of various elements; the order of elements in a set is undefined. You can add and delete elements of a set, you can iterate the elements of the set, you can perform standard operations on sets (union, intersection, difference). Besides that, you can check if an element belongs to a set.

Creating a Set:

A set is created by placing all the items (elements) inside curly braces {}, separated by comma or by using the built-in function set().

```
>>> my_set = {1, 2, 3}   # set of integers

>>> my_set

set([1, 2, 3])

>>> my_set2 = {1.0, "Hello", (1, 2, 3)}    # set of mixed datatypes

>>> my_set2

set([1.0, 'Hello', (1, 2, 3)])
```

 # a set removes duplicates and outputs only unique values

```
>>> my_set3 = {1,2,3,4,3,2}
>>> my_set3
set([1, 2, 3, 4])

>>> cities = set(("Hyderabad", "Mumbai",
"Delhi","Bangalore","Hyderabad","Chennai"))
>>> cities
set(['Chennai', 'Hyderabad', 'Bangalore', 'Delhi', 'Mumbai'])
```

 # In the following example, a string is singularized into its characters to build the resulting set x:

```
>>> x = set("A Python Tutorial")
>>> x
set(['A', ' ', 'i', 'h', 'l', 'o', 'n', 'P', 'r', 'u', 't', 'a', 'y', 'T'])
```

```
>>> my_string = 'foobar'
>>> setA = set(my_string)
>>> setA
set(['a', 'r', 'b', 'o', 'f'])
```

Sets are implemented in a way, which doesn't allow mutable objects.

set cannot have mutable items. Here [3, 4] is a mutable list:

```
>>> my_set4 = {1, 2, [3, 4]}
```

```
Traceback (most recent call last):
  File "<pyshell#10>", line 1, in <module>
    my_set4 = {1, 2, [3, 4]}
TypeError: unhashable type: 'list'
```

 # we can make set from a list

```
>>> my_set5 = set([1,2,3,2])
>>> my_set5
set([1, 2, 3])
```

#creating an empty set:

```
>>> a = {}
>>> a
{}

>>> a = set()
>>> a
set([])
```

Frozen sets

Frozenset 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.

```
>>> cities = frozenset(["Hyderabad", "Bangalore","Chennai"])
>>> cities.add("Delhi")

Traceback (most recent call last):
  File "<pyshell#29>", line 1, in <module>
    cities.add("Delhi")
AttributeError: 'frozenset' object has no attribute 'add'
```

Changing a Set
Adding elemens to a Set:

Though sets can't contain mutable objects, sets are mutable. But since they are unordered, indexing have no meaning. We cannot access or change an element of set using indexing or slicing. Set does not support it.

We can add single element using the add() method and multiple elements using the update() method. The update() method can take tuples, lists, strings or other sets as its argument. In all cases, duplicates are avoided.

```
>>> my_set6 = {1,2,3}
>>> my_set6
set([1, 2, 3])

>>> my_set6[0]

Traceback (most recent call last):
  File "<pyshell#33>", line 1, in <module>
    my_set6[0]
TypeError: 'set' object does not support indexing

>>> SetX = {1,2,3,4,5,6,7,8}
>>> SetX
set([1, 2, 3, 4, 5, 6, 7, 8])

>>> SetX.add((9,10))  # adding a tuple (9,10) to SetX
>>> SetX
set([1, 2, 3, 4, 5, 6, 7, 8, (9, 10)])

>>> SetX.update([11,12])   # adding multiple elements
>>> SetX
set([1, 2, 3, 4, 5, 6, 7, 8, 11, 12, (9, 10)])

#adding a list and set to an existing set:
>>> SetX.update(["Hello", "Python"], {1.5, 10.0})
>>> SetX
set([1.5, 1, 2, 3, 4, 5, 6, 7, 8, 10.0, 11, 12, (9, 10), 'Python', 'Hello'])
```

Removing elements from a set

A particular item can be removed from set using methods, discard() and remove().

The only difference between the two is that, while using discard() if the item does not exist in the set, it remains unchanged. But remove() will raise an error in such condition.

```
>>> my_set7 = {1, 3, 4, 5, 6}
```

```
>>> my_set7
set([1, 3, 4, 5, 6])

>>> my_set7.discard(4)
>>> my_set7
set([1, 3, 5, 6])

>>> my_set7.remove(6)
>>> my_set7
set([1, 3, 5])
```

removing an element not present in the set will result in an error If the value is not present, discard() does not do anything. Whereas, remove will raise a KeyError exception.

```
>>> my_set7.remove(2)

Traceback (most recent call last):
  File "<pyshell#56>", line 1, in <module>
    my_set7.remove(2)
KeyError: 2
```

Similarly, we can remove and return an item using the pop() method. Set being unordered, there is no way of determining which item will be popped. It is completely arbitrary.

```
>>> my_set8 = set("HelloWorld")
>>> my_set8
set(['e', 'd', 'H', 'l', 'o', 'r', 'W'])

>>> print(my_set8.pop())
e
>>> my_set8.pop()
'd'
>>> my_set8.clear()   # clearing the set
>>> my_set8
set([])
```

Set Methods

Copy():

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

A set can be copied using = operator in Python.

```
numbers = {1, 2, 3, 4}

new_numbers = numbers
```

```
new_numbers.add('5')

print('numbers: ', numbers)  ----------------------------→ {1,2,3,4,'5'}
print('new_numbers: ', new_numbers) --------------→ {1,2,3,4,'5'}
```

The problem with copying the set in this way is that if you modify the numbers set, the new_numbers set is also modified. This is called deep copy.

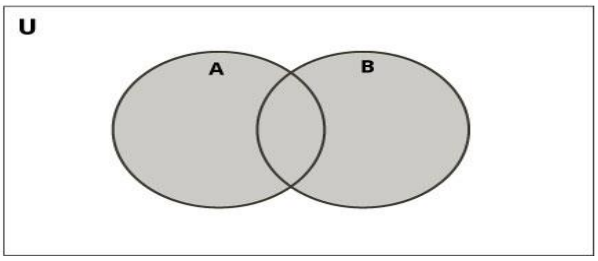
However, if you need the original set unchanged when the new set is modified, you can use copy() method. This is called shallow copy.

```
numbers = {1, 2, 3, 4}
new_numbers = numbers.copy()


new_numbers.add('5')

print('numbers: ', numbers) ----------------------------------→ {1,2,3,4}
print('new_numbers: ', new_numbers) --------------------→ {1,2,3,4,'5'}
```

Python SET Operations

Sets can be used to carry out mathematical set operations like union, intersection, difference and symmetric difference. We can do this with operators or methods.

SET Union:



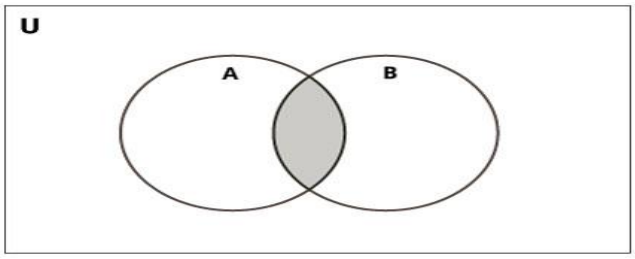
Let us consider the following two sets for the following operations.

Union of A and B is a set of all elements from both sets.
Union is performed using | operator. Same can be accomplished using the method union().

```
>>> print(A|B)
set([1, 2, 3, 4, 5, 6, 7, 8])

>>> A.union(B)
set([1, 2, 3, 4, 5, 6, 7, 8])
>>> B.union(A)
```

set([1, 2, 3, 4, 5, 6, 7, 8])

SET Intersection:



Intersection of A and B is a set of elements that are common in both sets. Intersection is performed using & operator. Same can be accomplished using the method intersection().

```
>>> print(A&B)
set([4, 5])
>>> A.intersection(B)
set([4, 5])
>>> B.intersection(A)
set([4, 5])
```

SET Difference:

Difference of A and B (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().

```
>>> print(A-B)
set([1, 2, 3])
>>> print(B-A)
set([8, 6, 7])

>>> A.difference(B)
set([1, 2, 3])
>>> B.difference(A)
set([8, 6, 7])
```

SET Symmetric Difference:

Symmetric Difference of A and B is a set of elements in both A and B except those that are common in both.

Symmetric difference is performed using ^ operator. Same can be accomplished using the method symmetric_difference().

```
>>> print(A^B)
set([1, 2, 3, 6, 7, 8])

>>> A.symmetric_difference(B)
set([1, 2, 3, 6, 7, 8])
>>> B.symmetric_difference(A)
set([1, 2, 3, 6, 7, 8])
```

issubset():

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

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

Issuperset():

The issuperset() method returns True if a set has every elements of another set. If not, it returns False.

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

Isdisjoint():

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

Two sets are said to be disjoint sets if they have no common elements. For example:

```
A = {1, 5, 9, 0}
B = {2, 4, -5}

>>> A = {1, 2, 3, 4}
>>> B = {5, 6, 7}
>>> C = {4, 5, 6}
>>> print('Are A and B disjoint?', A.isdisjoint(B))
('Are A and B disjoint?', True)
>>> print('Are A and C disjoint?', A.isdisjoint(C))
```

('Are A and C disjoint?', False)

Update():

The update() adds elements from a set to the set (calling the update() method).

Syntax:
A.update(B)

Here, A and B are two sets. The elements of set B are added to the set A.

```
>>> A = {'a', 'b'}
>>> B = {1, 2, 3}
>>> result = A.update(B)
>>> print('A =',A)
('A =', set(['a', 1, 2, 'b', 3]))
>>> print('B =',B)
('B =', set([1, 2, 3]))
>>> print('result =',result)
('result =', None)
```

Iterating Through a Set

Using a for loop, we can iterate though each item in a set.

```
>>> for letter in set("apple"):
        print(letter)
```

a

p

e

l