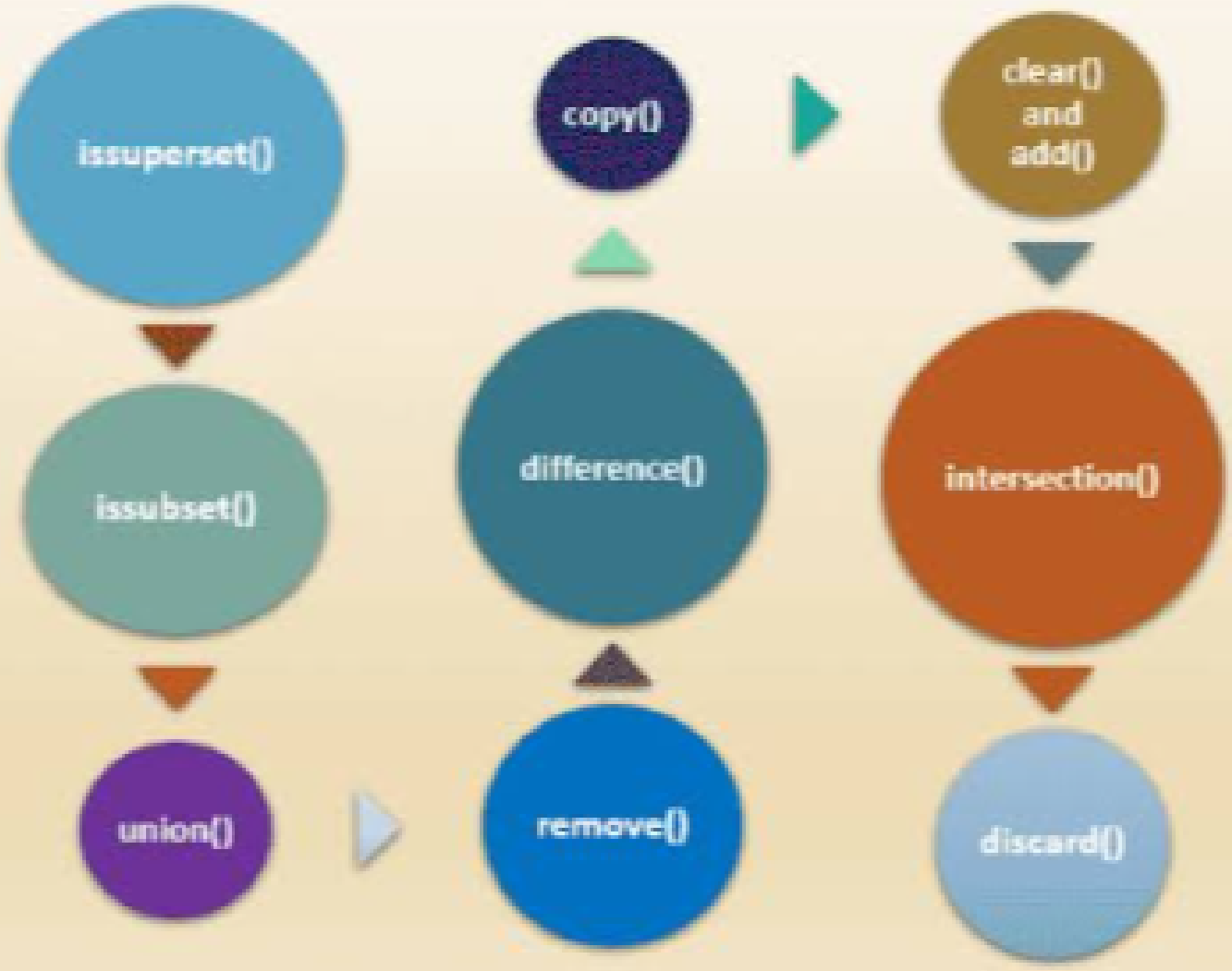
Python

Python
Sets

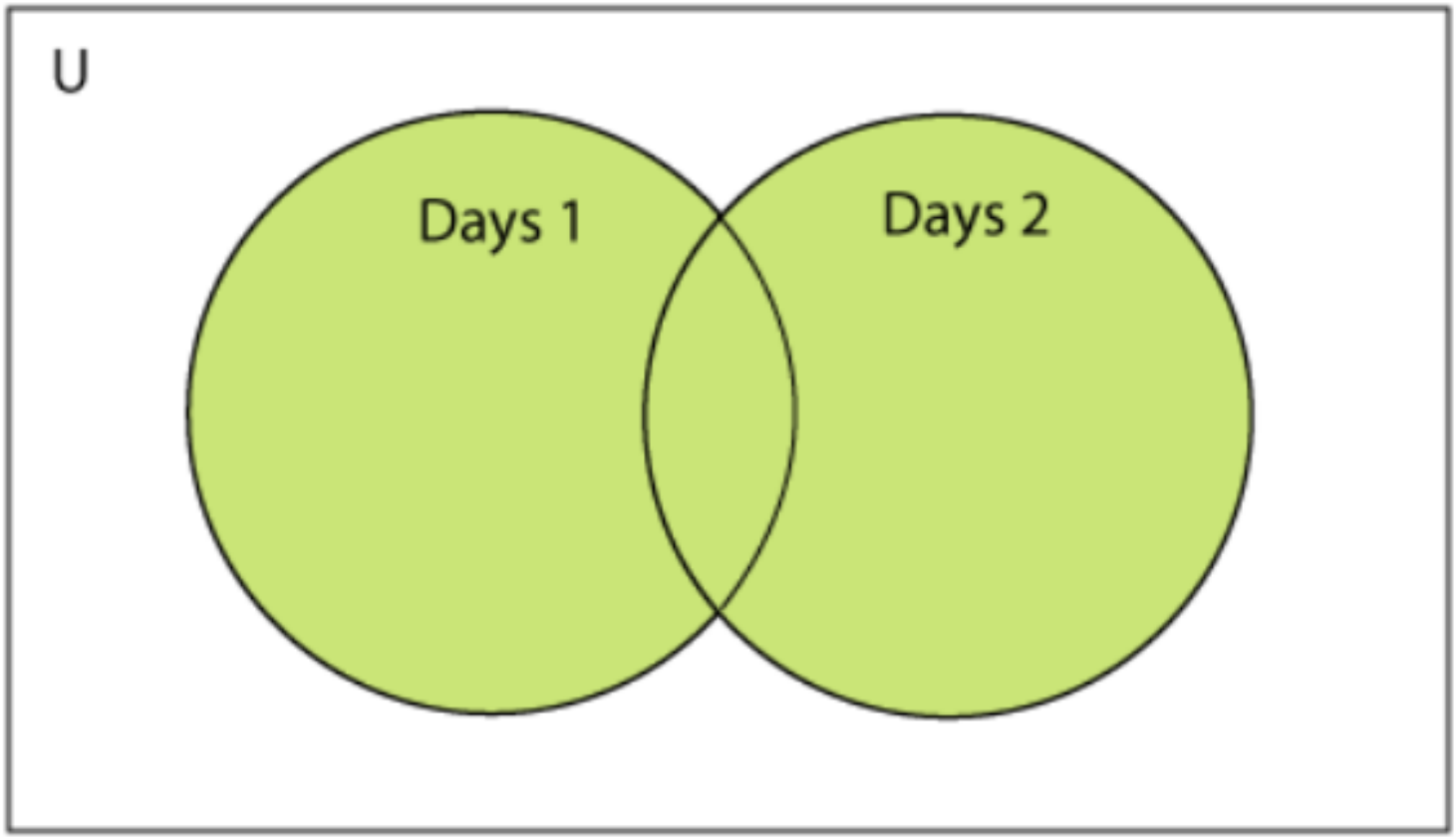

issuperset()
copy()
clear() and add()
issubset()
difference()
intersection()
union()
remove()
discard()

Python Set Operations

Set can be performed mathematical operation such as union, intersection, difference, and symmetric difference. Python provides the facility to carry out these operations with operators or methods. We describe these operations as follows.

Union of two Sets

The union of two sets is calculated by using the pipe (|) operator. The union of the two sets contains all the items that are present in both the sets.



Consider the following example to calculate the union of two sets.

Example 1: using union | operator

```
Days1 = {"Monday","Tuesday","Wednesday","Thursday", "Sun
Days2 = {"Friday","Saturday","Sunday"}
print(Days1|Days2) #printing the union of the sets
```

Output:

```
{'Friday', 'Sunday', 'Saturday', 'Tuesday', 'Wednesday', 'Monday', 'Thursday'}
```

Python also provides the **union()** method which can also be used to calculate the union of two sets. Consider the following example.

Example 2: using union() method

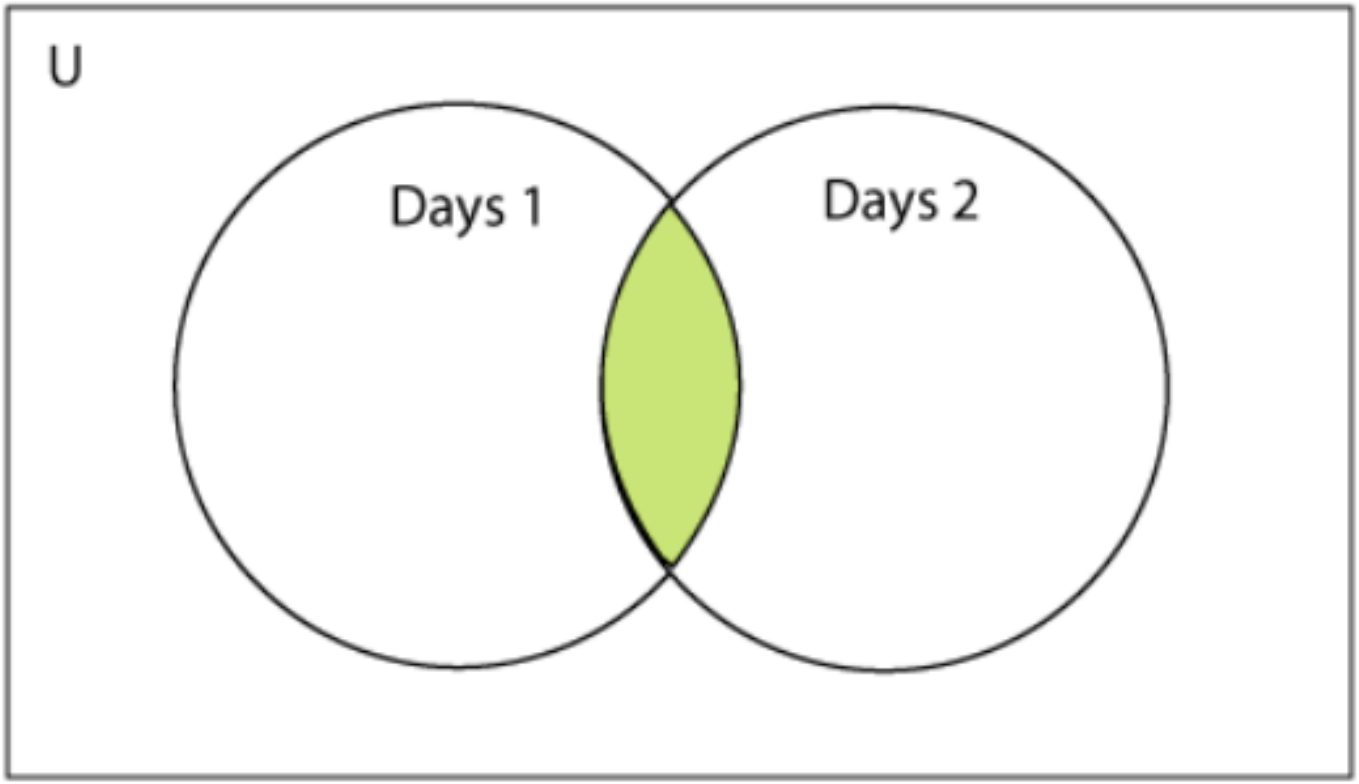
```
Days1 = {"Monday","Tuesday","Wednesday","Thursday"}
Days2 = {"Friday","Saturday","Sunday"}
print(Days1.union(Days2)) #printing the union of the sets
```

Output:

```
{'Friday', 'Sunday', 'Saturday', 'Tuesday', 'Wednesday', 'Monday', 'Thursday'}
```

Intersection of two sets

The intersection of two sets can be performed by the **and &** operator or the **intersection() function**. The intersection of the two sets is given as the set of the elements that common in both sets.



Consider the following example.

Example 1: Using & operator

```
Days1 = {"Monday","Tuesday", "Wednesday", "Thursday"}
Days2 = {"Monday","Tuesday","Sunday", "Friday"}
print(Days1&Days2) #prints the intersection of the two set:
```

Output:

```
{'Monday', 'Tuesday'}
```

Example 2: Using intersection() method

```
set1 = {"Devansh","John", "David", "Martin"}
set2 = {"Steve", "Milan", "David", "Martin"}
print(set1.intersection(set2)) #prints the intersection of the
```

Output:

```
{'Martin', 'David'}
```

Example 3:

```
set1 = {1,2,3,4,5,6,7}
set2 = {1,2,20,32,5,9}
set3 = set1.intersection(set2)
print(set3)
```

Output:

```
{1 , 2 , 5}
```

The intersection_update() method

The **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).

The **intersection_update()** method is different from the intersection() method since it modifies the original set by removing the unwanted items, on the other hand, the intersection() method returns a new set.

Consider the following example.

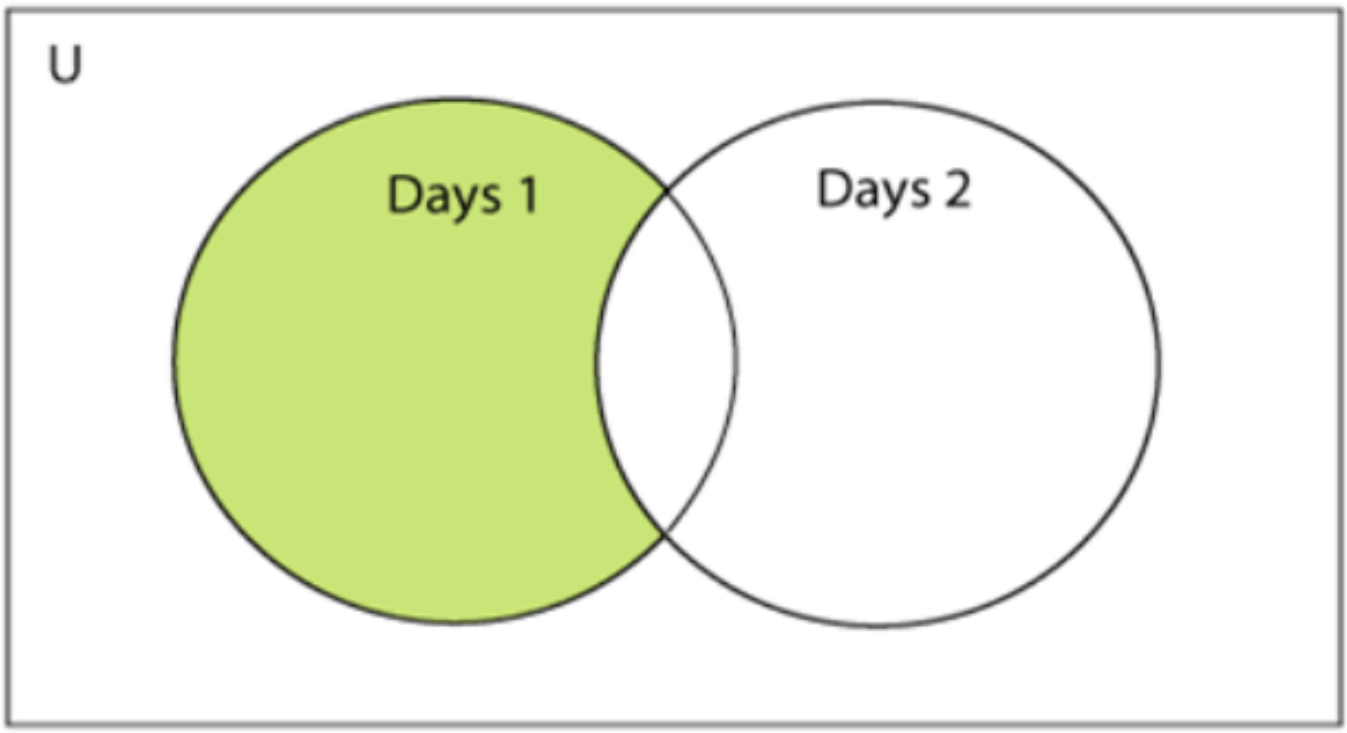
```
a = {"Devansh", "bob", "castle"}
b = {"castle", "dude", "emyway"}
c = {"fuson", "gaurav", "castle"}

a.intersection_update(b, c)

print(a)
```

Output:

```
{'castle'}
```

Difference between the two sets

The difference of two sets can be calculated by using the subtraction (-) operator or **intersection()** method. Suppose there are two sets A and B, and the difference is A-B that denotes the resulting set will be obtained that element of A, which is not present in the set B.



Consider the following example.

Example 1 : Using subtraction (-) operator

```
Days1 = {"Monday", "Tuesday", "Wednesday", "Thursday"}
Days2 = {"Monday", "Tuesday", "Sunday"}
print(Days1-Days2)                                    #
{"Wednesday", "Thursday" will be printed}
```

Output:

```
{'Thursday', 'Wednesday'}
```

Example 2 : Using difference() method

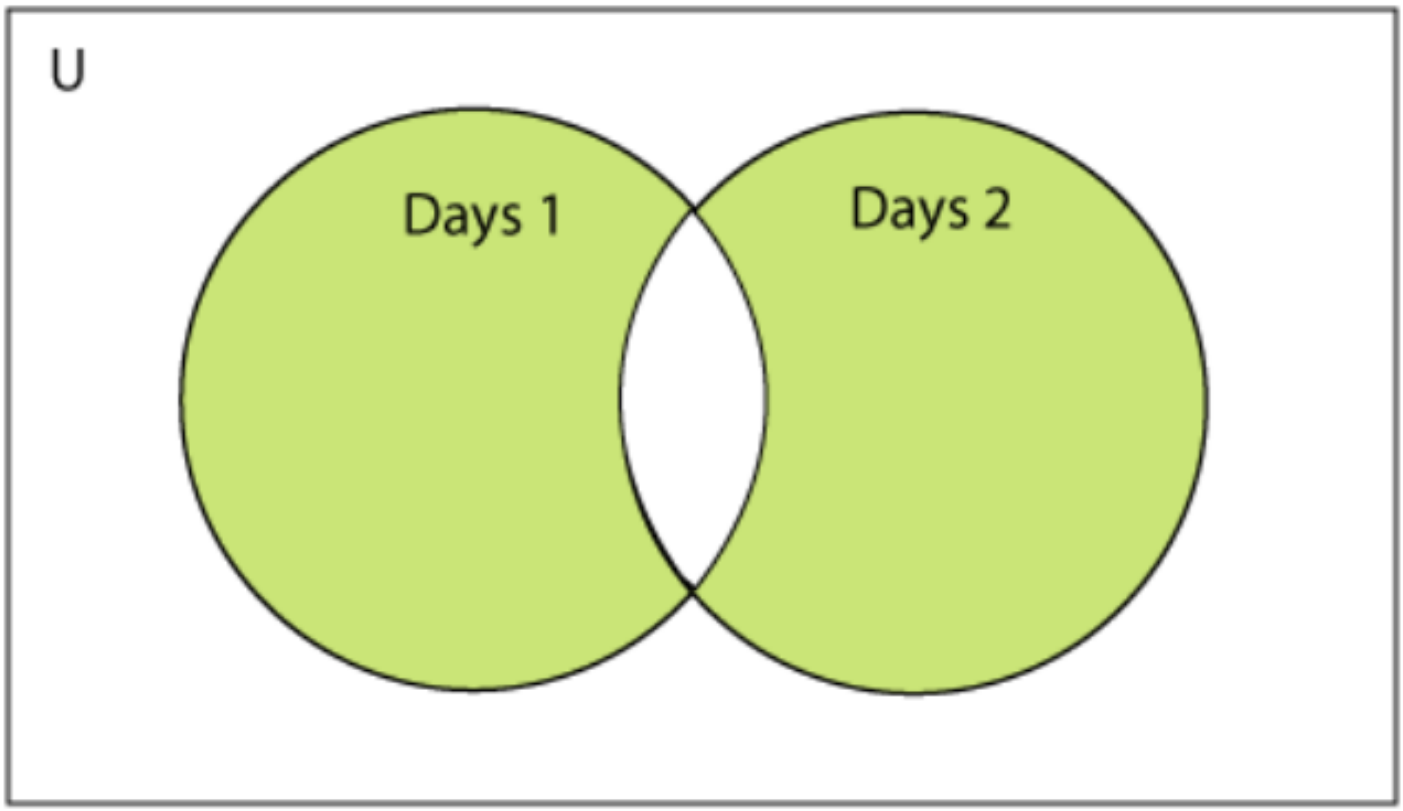
```
Days1 = {"Monday", "Tuesday", "Wednesday", "Thursday"}
Days2 = {"Monday", "Tuesday", "Sunday"}
print(Days1.difference(Days2)) # prints the difference of th
```

Output:

```
{'Thursday', 'Wednesday'}
```

Symmetric Difference of two sets

The symmetric difference of two sets is calculated by ^ operator or **symmetric_difference()** method. Symmetric difference of sets, it removes that element which is present in both sets. Consider the following example:



Example - 1: Using ^ operator

```
a = {1,2,3,4,5,6}
b = {1,2,9,8,10}
c = a^b
print(c)
```

Output:

```
{3, 4, 5, 6, 8, 9, 10}
```

Example - 2: Using symmetric_difference() method

```
a = {1,2,3,4,5,6}
b = {1,2,9,8,10}
c = a.symmetric_difference(b)
print(c)
```

Output:

```
{3, 4, 5, 6, 8, 9, 10}
```

Set comparisons

Python allows us to use the comparison operators i.e., <, >, <=, >= , == with the sets by using which we can check whether a set is a subset, superset, or equivalent to other set. The boolean true or false is returned depending upon the items present inside the sets.

Consider the following example.

```
Days1 = {"Monday",  "Tuesday", "Wednesday", "Thursday"}
Days2 = {"Monday", "Tuesday"}
Days3 = {"Monday", "Tuesday", "Friday"}

#Days1 is the superset of Days2 hence it will print true.
print (Days1>Days2)

#prints false since Days1 is not the subset of Days2
print (Days1<Days2)

#prints false since Days2 and Days3 are not equivalent
print (Days2 == Days3)
```

Output:

```
True
False
False
```

FrozenSets

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 the 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.

Consider the following example to create the frozen set.

```
Frozenset = frozenset([1,2,3,4,5])
print(type(Frozenset))
print("\nprinting the content of frozen set...")
for i in Frozenset:
    print(i);
Frozenset.add(6) #gives an error since we cannot change the content of Frozens
```

Output:

```
<class 'frozenset'>

printing the content of frozen set...
1
2
3
4
5
Traceback (most recent call last):
  File "set.py", line 6, in <module>
    Frozenset.add(6) #gives an error since we can change the content of Frozenset after creation
AttributeError: 'frozenset' object has no attribute 'add'
```

Python Built-in set methods

Python contains the following methods to be used with the s

SN	Method	Description
1	add(item)	It adds an item to the set. It has no effect if the item is already present in the set.
2	clear()	It deletes all the items from the set.
3	copy()	It returns a shallow copy of the set.
4	difference_update(....)	It modifies this set by removing all the items that are also present in the specified sets.
5	discard(item)	It removes the specified item from the set.

6	intersection()	It returns a new set that contains only the common elements of both the sets. (all the sets if more than two are specified).
7	intersection_update(....)	It removes the items from the original set that are not present in both the sets (all the sets if more than one are specified).
8	Isdisjoint(....)	Return True if two sets have a null intersection.
9	Issubset(....)	Report whether another set contains this set.
10	Issuperset(....)	Report whether this set contains another set.

11	pop()	Remove and return an arbitrary set element that is the last element of the set. Raises KeyError if the set is empty.
12	remove(item)	Remove an element from a set; it must be a member. If the element is not a member, raise a KeyError.
13	symmetric_difference(....)	Remove an element from a set; it must be a member. If the element is not a member, raise a KeyError.

14	symmetric_difference_update(....)	Update a set with the symmetric difference of itself and another.
15	union(....)	Return the union of sets as a new set. (i.e. all elements that are in either set.)
16	update()	Update a set with the union of itself and others.