Set methods

add() Adds an element to the set

In [1]:
```
#The add() method adds an element to the set.
#If the element already exists, the add() method does not add the element.
fruits = {"apple", "banana", "cherry"}

fruits.add("orange")

print(fruits)
```

```
{'cherry', 'orange', 'apple', 'banana'}
```

clear() Removes all the elements from the set

In [2]:
```
#The clear() method removes all elements in a set.
fruits = {"apple", "banana", "cherry"}

fruits.clear()

print(fruits)
```

```
set()
```

copy() Returns a copy of the set

In [3]:
```
#The copy() method copies the set.
fruits = {"apple", "banana", "cherry"}

x = fruits.copy()

print(x)
```

```
{'cherry', 'apple', 'banana'}
```

difference() Returns a set containing the difference between two or more sets

In [4]:
```
#The difference() method returns a set that contains the difference between two sets.

#Meaning: The returned set contains items that exist only in the first set, and not in both sets.
x = {"apple", "banana", "cherry"}
y = {"google", "microsoft", "apple"}

z = x.difference(y)

print(z)
```

```
{'cherry', 'banana'}
```

difference_update() Removes the items in this set that are also included in another, specified set

The difference_update() method removes the items that exist in both sets. The difference_update() method is different from the difference() method, because the difference() method returns a new set, without the unwanted items, and the difference_update() method removes the unwanted items from the original set.

In [5]:
```
x = {"apple", "banana", "cherry"}
y = {"google", "microsoft", "apple"}

x.difference_update(y)

print(x)
```

```
{'cherry', 'banana'}
```

discard() Remove the specified item

In [6]:
```
fruits = {"apple", "banana", "cherry"}

fruits.discard("banana")

print(fruits)
```

```
{'cherry', 'apple'}
```

intersection() Returns a set, that is the intersection of two or more sets

In [7]:
```
x = {"apple", "banana", "cherry"}
y = {"google", "microsoft", "apple"}

z = x.intersection(y)

print(z)
```

```
{'apple'}
```

intersection_update() Removes the items in this set that are not present in other, specified set(s)

The intersection_update() method removes the items that is not present in both sets (or in all sets if the comparison is done between more than two sets). The intersection_update() method is different from the intersection() method, because the intersection() method returns a new set, without the unwanted items, and the intersection_update() method removes the unwanted items from the original set.

In [8]:
```
x = {"apple", "banana", "cherry"}
y = {"google", "microsoft", "apple"}

x.intersection_update(y)

print(x)
```

```
{'apple'}
```

isdisjoint() Returns whether two sets have a intersection or not

In [10]:
```
#The isdisjoint() method returns True if none of the items are present in both sets,
#otherwise it returns False.
x = {"apple", "banana", "cherry"}
y = {"google", "microsoft", "facebook"}

z = x.isdisjoint(y)

print(z)
```

```
True
```

issubset() Returns whether another set contains this set or not

The issubset() method returns True if all items in the set exists in the specified set, otherwise it returns False.

In [11]:
```
x = {"a", "b", "c"}
y = {"f", "e", "d", "c", "b", "a"}

z = x.issubset(y)

print(z)
```

```
True
```

issuperset() Returns whether this set contains another set or not

In [12]:
```
x = {"f", "e", "d", "c", "b", "a"}
y = {"a", "b", "c"}

z = x.issuperset(y)

print(z)
```

```
True
```

pop() Removes an element from the set

In [13]:
```
fruits = {"apple", "banana", "cherry"}

fruits.pop()

print(fruits)
```

```
{'banana', 'apple'}
```

remove() Removes the specified element

In [14]:
```
fruits = {"apple", "banana", "cherry"}

fruits.remove("banana")

print(fruits)
```

```
{'cherry', 'apple'}
```

symmetric_difference() Returns a set with the symmetric differences of two sets

The symmetric_difference() method returns a set that contains all items from both set, but not the items that are present in both sets. Meaning: The returned set contains a mix of items that are not present in both sets.

In [15]:
```
x = {"apple", "banana", "cherry"}
y = {"google", "microsoft", "apple"}

z = x.symmetric_difference(y)

print(z)
```

```
{'cherry', 'microsoft', 'banana', 'google'}
```

symmetric_difference_update() inserts the symmetric differences from this set and another

The symmetric_difference_update() method updates the original set by removing items that are present in both sets, and inserting the other items.

In [16]:
```
x = {"apple", "banana", "cherry"}
y = {"google", "microsoft", "apple"}

x.symmetric_difference_update(y)

print(x)
```

```
{'cherry', 'microsoft', 'banana', 'google'}
```

union() Return a set containing the union of sets

The union() method returns a set that contains all items from the original set, and all items from the specified set(s). You can specify as many sets you want, separated by commas. It does not have to be a set, it can be any iterable object. If an item is present in more than one set, the result will contain only one appearance of this item.

In [17]:
```
x = {"apple", "banana", "cherry"}
y = {"google", "microsoft", "apple"}

z = x.union(y)

print(z)
```

```
{'cherry', 'microsoft', 'banana', 'google', 'apple'}
```

update() Update the set with another set, or any other iterable

In [ ]:
```
The update() method updates the current set, by adding items from another set (or any other iterable).
If an item is present in both sets, only one appearance of this item will be present in the updated set.
```

In [ ]:
```
x = {"apple", "banana", "cherry"}
y = {"google", "microsoft", "apple"}

x.update(y)

print(x)
```