

Sets

A set is an unordered collection of items. Every set element is unique (no duplicates).

However, a set itself is mutable. We can add or remove items from it.

Sets can also be used to perform mathematical set operations like union, intersection, symmetric difference, etc.

Creating Python Sets

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

It can have any number of items and they may be of different types (integer, float, tuple, string etc.). But a set cannot have mutable elements like lists, sets or dictionaries as its elements.

In [11]:

```
# Different types of sets in Python
# set of integers
my_set = {1, 2, 3}
print(my_set)

# set of mixed datatypes
my_set = {1.0, "Hello", (1, 2, 3)}
print(my_set)
```

```
{1, 2, 3}
{1.0, 'Hello', (1, 2, 3)}
```

In [12]:

```
# set cannot have duplicates
# Output: {1, 2, 3, 4}
my_set = {1, 2, 3, 4, 3, 2}
print(my_set)

# we can make set from a list
# Output: {1, 2, 3}
my_set = set([1, 2, 3, 2])
print(my_set)

# set cannot have mutable items
# here [3, 4] is a mutable list
# this will cause an error.

my_set = {1, 2, [3, 4]}
```

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

```
-----
TypeError                                Traceback (most recent call last)
<ipython-input-12-40880666f5f2> in <module>
    13 # this will cause an error.
    14
--> 15 my_set = {1, 2, [3, 4]}
```

```
TypeError: unhashable type: 'list'
```

Creating an empty set is a bit tricky.

Empty curly braces {} will make an empty dictionary in Python. To make a set without any elements, we use the set() function without any argument.

In [13]:

```
# Distinguish set and dictionary while creating empty set

# initialize a with {}
a = {}

# check data type of a
print(type(a))

# initialize a with set()
a = set()

# check data type of a
print(type(a))
```

```
<class 'dict'>
<class 'set'>
```

Modifying a set in Python

Sets are mutable. However, since they are unordered, indexing has no meaning.

We cannot access or change an element of a set using indexing or slicing. Set data type does not support it.

We can add a 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.

In [14]:

```
# initialize my_set
my_set = {1, 3}
print(my_set)

# if you uncomment line 9,
# you will get an error
# TypeError: 'set' object does not support indexing

# my_set[0]

# add an element
# Output: {1, 2, 3}
my_set.add(2)
print(my_set)

# add multiple elements
# Output: {1, 2, 3, 4}
my_set.update([2, 3, 4])
print(my_set)

# add list and set
# Output: {1, 2, 3, 4, 5, 6, 8}
my_set.update([4, 5], {1, 6, 8})
print(my_set)
```

```
{1, 3}
{1, 2, 3}
{1, 2, 3, 4}
{1, 2, 3, 4, 5, 6, 8}
```

Removing elements from a set

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

The only difference between the two is that the `discard()` function leaves a set unchanged if the element is not present in the set. On the other hand, the `remove()` function will raise an error in such a condition (if element is not present in the set).

The following example will illustrate this.

In [15]:

```

# Difference between discard() and remove()

# initialize my_set
my_set = {1, 3, 4, 5, 6}
print(my_set)

# discard an element
# Output: {1, 3, 5, 6}
my_set.discard(4)
print(my_set)

# remove an element
# Output: {1, 3, 5}
my_set.remove(6)
print(my_set)

# discard an element
# not present in my_set
# Output: {1, 3, 5}
my_set.discard(2)
print(my_set)

# remove an element
# not present in my_set
# you will get an error.
# Output: KeyError

my_set.remove(2)

```

```

{1, 3, 4, 5, 6}
{1, 3, 5, 6}
{1, 3, 5}
{1, 3, 5}

```

```

-----
KeyError                                Traceback (most recent call last)
<ipython-input-15-8cb8edd7dd9a> in <module>
    26 # Output: KeyError
    27
--> 28 my_set.remove(2)

```

KeyError: 2

Similarly, we can remove and return an item using the pop() method.

Since set is an unordered data type, there is no way of determining which item will be popped. It is completely arbitrary.

We can also remove all the items from a set using the clear() method.

In [16]:

```
# initialize my_set
# Output: set of unique elements
my_set = set("HelloWorld")
print(my_set)

# pop an element
# Output: random element
print(my_set.pop())

# pop another element
my_set.pop()
print(my_set)

# clear my_set
# Output: set()
my_set.clear()
print(my_set)

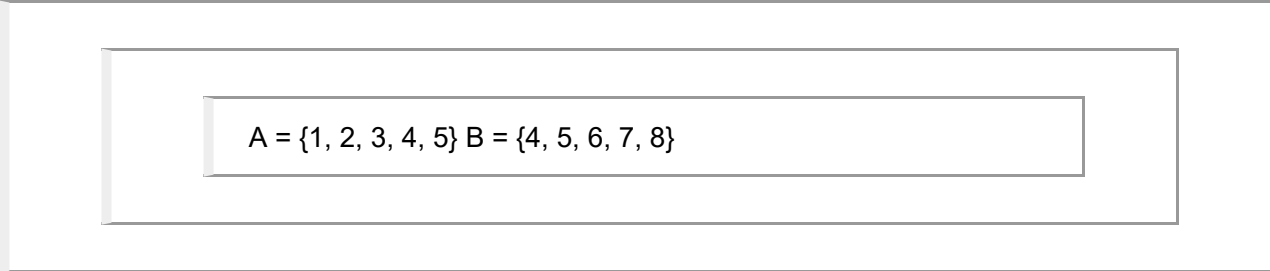
print(my_set)
```

```
{'H', 'W', 'd', 'l', 'e', 'o', 'r'}
H
{'d', 'l', 'e', 'o', 'r'}
set()
set()
```

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.

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



$A = \{1, 2, 3, 4, 5\}$ $B = \{4, 5, 6, 7, 8\}$

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 union() method.

In [17]:

```
# Set union method
# initialize A and B
A = {1, 2, 3, 4, 5}
B = {4, 5, 6, 7, 8}

# use | operator
# Output: {1, 2, 3, 4, 5, 6, 7, 8}
print(A | B)
```

{1, 2, 3, 4, 5, 6, 7, 8}

In [18]:

```
# use union function
>>> A.union(B)
{1, 2, 3, 4, 5, 6, 7, 8}

# use union function on B
>>> B.union(A)
{1, 2, 3, 4, 5, 6, 7, 8}
```

Out[18]:

{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 the sets.

Intersection is performed using & operator. Same can be accomplished using the intersection() method.

In [19]:

```
# Intersection of sets
# initialize A and B
A = {1, 2, 3, 4, 5}
B = {4, 5, 6, 7, 8}

# use & operator
# Output: {4, 5}
print(A & B)
```

{4, 5}

In [20]:

```
# use intersection function on A
>>> A.intersection(B)
{4, 5}

# use intersection function on B
>>> B.intersection(A)
{4, 5}
```

Out[20]:

{4, 5}

Set Difference

Difference of the set B from set A ($A - B$) is a set of elements that are only in A but not in B. Similarly, $B - A$ is a set of elements in B but not in A.

Difference is performed using $-$ operator. Same can be accomplished using the `difference()` method.

In [21]:

```
# Difference of two sets
# initialize A and B
A = {1, 2, 3, 4, 5}
B = {4, 5, 6, 7, 8}

# use - operator on A
# Output: {1, 2, 3}
print(A - B)
```

{1, 2, 3}

In [22]:

```
# use difference function on A
>>> A.difference(B)
{1, 2, 3}

# use - operator on B
>>> B - A
{8, 6, 7}

# use difference function on B
>>> B.difference(A)
{8, 6, 7}
```

Out[22]:

{6, 7, 8}

Set Symmetric Difference

Symmetric Difference of A and B is a set of elements in A and B but not in both (excluding the intersection).

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

In [23]:

```
# Symmetric difference of two sets
# initialize A and B
A = {1, 2, 3, 4, 5}
B = {4, 5, 6, 7, 8}

# use ^ operator
# Output: {1, 2, 3, 6, 7, 8}
print(A ^ B)
```

{1, 2, 3, 6, 7, 8}

In [24]:

```
# use symmetric_difference function on A
>>> A.symmetric_difference(B)
{1, 2, 3, 6, 7, 8}

# use symmetric_difference function on B
>>> B.symmetric_difference(A)
{1, 2, 3, 6, 7, 8}
```

Out[24]:

{1, 2, 3, 6, 7, 8}

Other Python Set Methods

There are many set methods, some of which we have already used above. Here is a list of all the methods that are available with the set objects:

Method	Description
<code>add()</code>	Adds an element to the set
<code>clear()</code>	Removes all elements from the set
<code>copy()</code>	Returns a copy of the set
<code>difference()</code>	Returns the difference of two or more sets as a new set
<code>difference_update()</code>	Removes all elements of another set from this set
<code>discard()</code>	Removes an element from the set if it is a member. (Do nothing if the element is not in set)
<code>intersection()</code>	Returns the intersection of two sets as a new set
<code>intersection_update()</code>	Updates the set with the intersection of itself and another
<code>isdisjoint()</code>	Returns True if two sets have a null intersection
<code>issubset()</code>	Returns True if another set contains this set
<code>issuperset()</code>	Returns True if this set contains another set
<code>pop()</code>	Removes and returns an arbitrary set element. Raises <code>KeyError</code> if the set is empty
<code>remove()</code>	Removes an element from the set. If the element is not a member, raises a <code>KeyError</code>
<code>symmetric_difference()</code>	Returns the symmetric difference of two sets as a new set
<code>symmetric_difference_update()</code>	Updates a set with the symmetric difference of itself and another
<code>union()</code>	Returns the union of sets in a new set
<code>update()</code>	Updates the set with the union of itself and others

Python Set add()

The set add() method adds a given element to a set. If the element is already present, it doesn't add any element.

The syntax of set add() method is:

set.add(elem) The add() method doesn't add an element to the set if it's already present in it.

Also, you don't get back a set if you use add() method when creating a set object.

noneValue = set().add(elem) The above statement doesn't return a reference to the set but 'None', because the statement returns the return type of add which is 'None',

Set add() Parameters The add() method takes a single parameter:

elem - the element that is added to the set

In [27]:

```
#Example 1: Add an element to a set
# set of vowels
vowels = {'a', 'e', 'i', 'u'}

# adding 'o'
vowels.add('o')
print('Vowels are:', vowels)

# adding 'a' again
vowels.add('a')
print('Vowels are:', vowels)
```

Vowels are: {'i', 'a', 'e', 'o', 'u'}

Vowels are: {'i', 'a', 'e', 'o', 'u'}

In [26]:

```
#Example 2: Add tuple to a set
# set of vowels
vowels = {'a', 'e', 'u'}

# a tuple ('i', 'o')
tup = ('i', 'o')

# adding tuple
vowels.add(tup)
print('Vowels are:', vowels)

# adding same tuple again
vowels.add(tup)
print('Vowels are:', vowels)
```

Vowels are: {'e', ('i', 'o'), 'u', 'a'}

Vowels are: {'e', ('i', 'o'), 'u', 'a'}

Python Set clear()

The clear() method removes all elements from the set.

The syntax of clear() method is:

set.clear() Set clear() Parameters The clear() method doesn't take any parameters.

Return value from Set clear()

The clear() method doesn't return any value and returns a 'None'.

In [28]:

```
#Example 1: Remove all elements from a Python set using clear()
# set of vowels
vowels = {'a', 'e', 'i', 'o', 'u'}
print('Vowels (before clear):', vowels)

# clearing vowels
vowels.clear()
print('Vowels (after clear):', vowels)
```

Vowels (before clear): {'i', 'a', 'e', 'o', 'u'}

Vowels (after clear): set()

Python Set copy()

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

A set can be copied using = operator in Python. For example:

numbers = {1, 2, 3, 4} new_numbers = numbers The problem with copying the set in this way is that if you modify the numbers set, the new_numbers set is also modified.

In [29]:

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

new_numbers.add(5)

print('numbers: ', numbers)
print('new_numbers: ', new_numbers)
```

numbers: {1, 2, 3, 4, 5}

new_numbers: {1, 2, 3, 4, 5}

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

The syntax of copy() is:

set.copy()

In [30]:

```
#Example 1: How the copy() method works for sets?
```

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

```
new_numbers = numbers.copy()
```

```
new_numbers.add(5)
```

```
print('numbers: ', numbers)
```

```
print('new_numbers: ', new_numbers)
```

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

```
new_numbers: {1, 2, 3, 4, 5}
```

Python Set difference()

If A and B are two sets. The set difference of A and B is a set of elements that exists only in set A but not in B. For example:

If A = {1, 2, 3, 4} B = {2, 3, 9}

Then, A - B = {1, 4} B - A = {9}

The syntax of difference() method in Python is:

A.difference(B) Here, A and B are two sets. The following syntax is equivalent to A-B.

The difference() method returns the difference of two sets which is also a set. It doesn't modify original sets.

In [31]:

```
#Example 1: How difference() works in Python?
```

```
A = {'a', 'b', 'c', 'd'}
```

```
B = {'c', 'f', 'g'}
```

```
# Equivalent to A-B
```

```
print(A.difference(B))
```

```
# Equivalent to B-A
```

```
print(B.difference(A))
```

```
{'b', 'd', 'a'}
```

```
{'g', 'f'}
```

In [33]:

```
#Example 2: Set Difference Using - Operator.
```

```
A = {'a', 'b', 'c', 'd'}
```

```
B = {'c', 'f', 'g'}
```

```
print(A-B)
```

```
print(B-A)
```

```
{'b', 'd', 'a'}
```

```
{'g', 'f'}
```

Python Set difference_update()

The difference_update() updates the set calling difference_update() method with the difference of sets.

If A and B are two sets. The set difference of A and B is a set of elements that exists only in set A but not in B.

The difference_update() updates the set calling difference_update() method with the difference of sets.

If A and B are two sets. The set difference of A and B is a set of elements that exists only in set A but not in B.

When you run the code,

result will be None A will be equal to A-B B will be unchanged

In [34]:

```
#Example: How difference_update() works?
```

```
A = {'a', 'c', 'g', 'd'}
```

```
B = {'c', 'f', 'g'}
```

```
result = A.difference_update(B)
```

```
print('A = ', A)
```

```
print('B = ', B)
```

```
print('result = ', result)
```

```
A = {'d', 'a'}
```

```
B = {'g', 'f', 'c'}
```

```
result = None
```

Python Set discard()

The discard() method removes a specified element from the set (if present).

The syntax of discard() in Python is:

s.discard(x) discard() Parameters The discard() method takes a single element x and removes it from the set (if present).

In [36]:

```
#Example 1: How discard() works?
```

```
numbers = {2, 3, 4, 5}
```

```
numbers.discard(3)
```

```
print('numbers = ', numbers)
```

```
numbers.discard(10)
```

```
print('numbers = ', numbers)
```

```
numbers = {2, 4, 5}
```

```
numbers = {2, 4, 5}
```

In [37]:

```
#Example 2: How discard() works?
```

```
numbers = {2, 3, 5, 4}
```

```
# Returns None
```

```
# Meaning, absence of a return value
```

```
print(numbers.discard(3))
```

```
print('numbers =', numbers)
```

None

```
numbers = {2, 4, 5}
```

Python Set intersection()

The intersection() method returns a new set with elements that are common to all sets.

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

```
B = {2, 3, 4, 9}
```

```
C = {2, 4, 9, 10}
```

Then,

```
A ∩ B = B ∩ A = {2, 3, 4}
```

```
A ∩ C = C ∩ A = {2, 4}
```

```
B ∩ C = C ∩ B = {2, 4, 9}
```

```
A ∩ B ∩ C = {2, 4}
```

The syntax of intersection() in Python is:

A.intersection(*other_sets) intersection() Parameters The intersection() allows arbitrary number of arguments (sets).

Note: * is not part of the syntax. It is used to indicate that the method allows arbitrary number of arguments.

In [38]:

```
#Example 1: How intersection() works?
```

```
A = {2, 3, 5, 4}
```

```
B = {2, 5, 100}
```

```
C = {2, 3, 8, 9, 10}
```

```
print(B.intersection(A))
```

```
print(B.intersection(C))
```

```
print(A.intersection(C))
```

```
print(C.intersection(A, B))
```

```
{2, 5}
```

```
{2}
```

```
{2, 3}
```

```
{2}
```

In [39]:

```
A = {100, 7, 8}
```

```
B = {200, 4, 5}
```

```
C = {300, 2, 3}
```

```
D = {100, 200, 300}
```

```
print(A.intersection(D))
```

```
print(B.intersection(D))
```

```
print(C.intersection(D))
```

```
print(A.intersection(B, C, D))
```

```
{100}
```

```
{200}
```

```
{300}
```

```
set()
```

You can also find the intersection of sets using & operator

In [40]:

```
#Example 3: Set Intersection Using & operator
```

```
A = {100, 7, 8}
```

```
B = {200, 4, 5}
```

```
C = {300, 2, 3, 7}
```

```
D = {100, 200, 300}
```

```
print(A & C)
```

```
print(A & D)
```

```
print(A & C & D)
```

```
print(A & B & C & D)
```

```
{7}
```

```
{100}
```

```
set()
```

```
set()
```

Python Set intersection_update()

The intersection_update() updates the set calling intersection_update() method with the intersection of sets.

The intersection of two or more sets is the set of elements which are common to all sets.

The syntax of intersection_update() is:

A.intersection_update(*other_sets) intersection_update() Parameters The intersection_update() method allows an arbitrary number of arguments (sets).

Note: * is not a part of the syntax. It is used to indicate that the method allows an arbitrary number of arguments.

```
result = A.intersection_update(B, C)
```

When you run the code,

result will be None

A will be equal to the intersection of A, B, and C

B remains unchanged

C remains unchanged

In [41]:

```
#Example 1: How intersection_update() Works?
```

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

```
B = {2, 3, 4, 5}
```

```
result = A.intersection_update(B)
```

```
print('result =', result)
```

```
print('A =', A)
```

```
print('B =', B)
```

```
result = None
```

```
A = {2, 3, 4}
```

```
B = {2, 3, 4, 5}
```

In [42]:

```
#Example 2: intersection_update() with Two Parameters
```

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

```
B = {2, 3, 4, 5, 6}
```

```
C = {4, 5, 6, 9, 10}
```

```
result = C.intersection_update(B, A)
```

```
print('result =', result)
```

```
print('C =', C)
```

```
print('B =', B)
```

```
print('A =', A)
```

```
result = None
```

```
C = {4}
```

```
B = {2, 3, 4, 5, 6}
```

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

Python Set 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} Here, sets A and B are disjoint sets.

isdisjoint() Parameters The isdisjoint() method takes a single argument (a set).

You can also pass an iterable (list, tuple, dictionary and string) to disjoint(). The isdisjoint() method will automatically convert iterables to set and checks whether the sets are disjoint or not.

Return Value from isdisjoint() The isdisjoint() method returns

True if two sets are disjoint sets (if set_a and set_b are disjoint sets in above syntax) False if two sets are not disjoint sets

In [43]:

```
#Example 1: How isdisjoint() works?
A = {1, 2, 3, 4}
B = {5, 6, 7}
C = {4, 5, 6}

print('Are A and B disjoint?', A.isdisjoint(B))
print('Are A and C disjoint?', A.isdisjoint(C))
```

Are A and B disjoint? True
Are A and C disjoint? False

In [45]:

```
#Example 2: isdisjoint() with Other Iterables as arguments
A = {'a', 'b', 'c', 'd'}
B = ['b', 'e', 'f']
C = '5de4'
D = {1 : 'a', 2 : 'b'}
E = {'a' : 1, 'b' : 2}

print('Are A and B disjoint?', A.isdisjoint(B))
print('Are A and C disjoint?', A.isdisjoint(C))
print('Are A and D disjoint?', A.isdisjoint(D))
print('Are A and E disjoint?', A.isdisjoint(E))
```

Are A and B disjoint? False
Are A and C disjoint? False
Are A and D disjoint? True
Are A and E disjoint? False

Python Set issubset()

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

Set A is said to be the subset of set B if all elements of A are in B.

Return Value from `issubset()`

The `issubset()` returns

True if A is a subset of B

False if A is not a subset of B

In [46]:

#Example: How `issubset()` works?

A = {1, 2, 3}

B = {1, 2, 3, 4, 5}

C = {1, 2, 4, 5}

Returns True

`print(A.issubset(B))`

Returns False

B is not subset of A

`print(B.issubset(A))`

Returns False

`print(A.issubset(C))`

Returns True

`print(C.issubset(B))`

True

False

False

True

python issuperset()

In [49]:

```
#Example: How issuperset() works?
```

```
A = {1, 2, 3}
```

```
B = {1, 2, 3, 4, 5}
```

```
C = {1, 2, 4, 5}
```

```
# Returns True
```

```
print(B.issuperset(A))
```

```
# Returns False
```

```
# A is not superset of B
```

```
print(A.issuperset(B))
```

```
# Returns False
```

```
print(A.issuperset(C))
```

```
# Returns True
```

```
print(B.issuperset(C))
```

True

False

False

True

Python Set pop()

The pop() method removes an arbitrary element from the set and returns the element removed.

The syntax of pop() for sets is:

```
set.pop()
```

```
pop() Parameters
```

The pop() method doesn't take any arguments.

In [51]:

```
#Example: How pop() works for Python Sets?
```

```
A = {'a', 'b', 'c', 'd'}
```

```
print('Return Value is', A.pop())
```

```
print('A = ', A)
```

Return Value is b

A = {'d', 'c', 'a'}

Python Set symmetric_difference()

The Python 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.

In [52]:

```
#Example 1: Working of symmetric_difference()
A = {'a', 'b', 'c', 'd'}
B = {'c', 'd', 'e' }
C = {}

print(A.symmetric_difference(B))
print(B.symmetric_difference(A))

print(A.symmetric_difference(C))
print(B.symmetric_difference(C))
```

```
{'b', 'e', 'a'}
{'a', 'b', 'e'}
{'b', 'c', 'd', 'a'}
{'e', 'c', 'd'}
```

Symmetric difference using ^ operator In Python, we can also find the symmetric difference using the ^ operator.

In [57]:

```
A = {'a', 'b', 'c', 'd'}
B = {'c', 'd', 'e' }
print(A ^ B)
print(B ^ A)

print(A ^ A)
print(B ^ B)
```

```
{'b', 'e', 'a'}
{'a', 'b', 'e'}
set()
set()
```

Python Set symmetric_difference_update()

The `symmetric_difference_update()` method finds the symmetric difference of two sets and updates the set calling it.

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.

In [58]:

```
#Example: Working of symmetric_difference_update()
A = {'a', 'c', 'd'}
B = {'c', 'd', 'e' }

result = A.symmetric_difference_update(B)

print('A =', A)
print('B =', B)
print('result =', result)
```

```
A = {'e', 'a'}
B = {'e', 'd', 'c'}
result = None
```

Here, the set A is updated with the symmetric difference of set A and B. However, the set B is unchanged.

Python Set union()

The Python set union() method returns a new set with distinct elements from all the sets.

The union of two or more sets is the set of all distinct elements present in all the sets. For example:

```
A = {1, 2}
B = {2, 3, 4}
C = {5}

Then,
AUB = BUA = {1, 2, 3, 4}
AUC = CUA = {1, 2, 5}
BUC = CUB = {2, 3, 4, 5}

AUBUC = {1, 2, 3, 4, 5}
```

Return Value from union() The union() method returns a new set with elements from the set and all other sets (passed as an argument). If the argument is not passed to union(), it returns a shallow copy of the set.

In [59]:

```
#Example 1: Working of union()
A = {'a', 'c', 'd'}
B = {'c', 'd', 2 }
C = {1, 2, 3}

print('A U B =', A.union(B))
print('B U C =', B.union(C))
print('A U B U C =', A.union(B, C))
print('A.union() =', A.union())
```

```
A U B = {2, 'd', 'c', 'a'}
B U C = {1, 2, 3, 'd', 'c'}
A U B U C = {1, 2, 3, 'd', 'c', 'a'}
A.union() = {'d', 'c', 'a'}
```

In [61]:

```
#Example 2: Set Union Using the | Operator
```

```
A = {'a', 'c', 'd'}
```

```
B = {'c', 'd', 2 }
```

```
C = {1, 2, 3}
```

```
print('A U B =', A | B)
```

```
print('B U C =', B | C)
```

```
print('A U B U C =', A | B | C)
```

```
A U B = {2, 'd', 'c', 'a'}
```

```
B U C = {1, 2, 3, 'd', 'c'}
```

```
A U B U C = {1, 2, 3, 'c', 'd', 'a'}
```

Python Set update()

The Python set update() method updates the set, adding items from other iterables.

The syntax of update() is:

A.update(iterable) Here, A is a set, and iterable can be any iterable such as list, set, dictionary, string, etc. The elements of the iterable are added to the set A.

In [62]:

```
#Example 1: Python set update()
```

```
A = {'a', 'b'}
```

```
B = {1, 2, 3}
```

```
result = A.update(B)
```

```
print('A =', A)
```

```
print('result =', result)
```

```
A = {1, 2, 3, 'b', 'a'}
```

```
result = None
```

In [64]:

```
#Example 2: Add elements of string and dictionary to Set
string_alphabet = 'abc'
numbers_set = {1, 2}

# add elements of the string to the set
numbers_set.update(string_alphabet)

print('numbers_set =', numbers_set)

info_dictionary = {'key': 1, 'lock' : 2}
numbers_set = {'a', 'b'}

# add keys of dictionary to the set
numbers_set.update(info_dictionary)
print('numbers_set =', numbers_set)
```

```
numbers_set = {1, 2, 'c', 'a', 'b'}
numbers_set = {'b', 'key', 'lock', 'a'}
```

Note: If dictionaries are passed to the update() method, the keys of the dictionaries are added to the set.

In []: