PYTHON SET --> { }

A Python `set` is an unordered collection of unique elements. It is defined by placing elements inside curly braces `{}`, separated by commas. Sets are a versatile data structure in Python that provide various operations for performing common set operations, such as union, intersection, difference, and membership tests.

Here are some key characteristics and features of Python sets:

1. Uniqueness:

- A set cannot contain duplicate elements. If you try to add an element that already exists in the set, it won't create a duplicate; the set remains unchanged.

2. Unordered:

- Sets are unordered, meaning that the order in which elements are stored is not guaranteed. Therefore, you cannot access elements in a set using indices or slices.

3. Mutable:

- Sets are mutable, allowing you to add and remove elements after the set has been created.

4. Set Operations:

- **Union (|):** Returns a new set containing all distinct elements from both sets.

```
set1 = {1, 2, 3}
set2 = {3, 4, 5}
union_set = set1 | set2  # or set1.union(set2)
```

- **Intersection (`&`):** Returns a new set containing common elements from both sets.

  ```
  intersection_set = set1 & set2  # or set1.intersection(set2)
  ```

- **Difference (-):** Returns a new set with elements from the first set that are not in the second set.

  ```
  difference_set = set1 - set2  # or set1.difference(set2)
  ```

- **Symmetric Difference (`^`):** Returns a new set with elements in either of the sets, but not both.

  ```
  symmetric_difference_set = set1 ^ set2  # or set1.symmetric_difference(set2)
  ```

5. Set Methods:

- **add():** Adds an element to the set.

  ```
  set1.add(4)
  ```

- **remove():** Removes a specified element from the set. Raises an error if the element is not present.

  ```
  set1.remove(2)
  ```

- **discard():** Removes a specified element from the set if it is present. Does not raise an error if the element is not present.

  ```
  set1.discard(2)
  ```

- **clear():** Removes all elements from the set, leaving it empty.

```
set1.clear()
```

- **copy():** Creates a shallow copy of the set.

```
set_copy = set1.copy()
```

Example:

```
# Creating sets
set1 = {1, 2, 3, 4}
set2 = {3, 4, 5, 6}

# Set operations
union_set = set1 | set2
intersection_set = set1 & set2
difference_set = set1 - set2
symmetric_difference_set = set1 ^ set2

# Set methods
set1.add(5)
set1.remove(2)
set1.discard(3)
set1.clear()

# Displaying results
print("Union:", union_set)
print("Intersection:", intersection_set)
print("Difference:", difference_set)
print("Symmetric Difference:", symmetric_difference_set)
```

Sets are useful when dealing with situations where you need to ensure uniqueness and perform set-related operations efficiently. Keep in mind that the order of elements in a set is not guaranteed, and if you need an ordered collection, you may want to use a different data structure, such as a list or a tuple.

```
dir(set)
```

```
['__and__',
 '__class__',
 '__contains__',
 '__delattr__',
 '__dir__',
 '__doc__',
 '__eq__',
 '__format__',
 '__ge__',
 '__getattribute__',
 '__gt__',
 '__hash__',
 '__iand__',
 '__init__',
 '__init_subclass__',
 '__ior__',
 '__isub__',
 '__iter__',
 '__ixor__',
 '__le__',
 '__len__',
 '__lt__',
 '__ne__',
 '__new__',
 '__or__',
 '__rand__',
 '__reduce__',
 '__reduce_ex__',
 '__repr__',
 '__ror__',
 '__rsub__',
      '__rxor__',
      '__setattr__',
      '__sizeof__',
      '__str__',
      '__sub__',
      '__subclasshook__',
      '__xor__',
      'add',
      'clear',
      'copy',
      'difference',
      'difference_update',
      'discard',
      'intersection',
      'intersection_update',
      'isdisjoint',
      'issubset',
      'issuperset',
      'pop',
      'remove',
      'symmetric_difference',
      'symmetric_difference_update',
      'union',
      'update']
```

Built-in Functions with Set

Function Description

***all()** *Returns True if all elements of the set are true (or if the set is empty).

***any()** *Returns True if any element of the set is true. If the set is empty, returns False.

***enumerate()** * Returns an enumerate object. It contains the index and value for all the items of the set as a pair.

len() Returns the length (the number of items) in the set.

max() Returns the largest item in the set.

min() Returns the smallest item in the set.

sorted() Returns a new sorted list from elements in the set(does not sort the set itself).

sum() Returns the sum of all elements in the set.

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

add() Adds an element to the set

clear() Removes all elements from the set

copy() Returns a copy of the set

difference() Returns the difference of two or more sets as a new set

difference_update() Removes all elements of another set from this set

discard() Removes an element from the set if it is a member. (Do nothing if the element is not in set)

intersection() Returns the intersection of two sets as a new set

intersection_update() Updates the set with the intersection of itself and another

isdisjoint() Returns True if two sets have a null intersection

issubset() Returns True if another set contains this set

issuperset() Returns True if this set contains another set

pop() Removes and returns an arbitrary set element. Raises KeyError if the set is empty

remove() Removes an element from the set. If the element is not a member, raises a KeyError

symmetric_difference() Returns the symmetric difference of two sets as a new set

symmetric_difference_update() Updates a set with the symmetric difference of itself and another

union() Returns the union of sets in a new set

update() Updates the set with the union of itself and others

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.

```
# 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)}
```

```
# 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-2-40880666f5f2> in <module>
     13 # this will cause an error.
     14 
---> 15 my_set = {1, 2, [3, 4]}

TypeError: unhashable type: 'list'
```

SEARCH STACK OVERFLOW

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.

```
# 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.

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

# my_set[0]
# if you uncomment the above line
# you will get an error
# TypeError: 'set' object does not support indexing


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

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


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

```
{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).

```
# 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-5-8cb8edd7dd9a> in <module>
     26 # Output: KeyError
     27
---> 28 my_set.remove(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.

```
# 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)
```

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

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

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.

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.

```
# 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}
```

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.

```
# 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}
```

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.

```
# 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)

print(B - A)
```

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

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().

```
# 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}
```

Set Membership Test

We can test if an item exists in a set or not, using the in keyword.

```
# in keyword in a set
# initialize my_set
my_set = set("apple")

# check if 'a' is present
# Output: True
print('a' in my_set)

# check if 'p' is present
# Output: False
print('p' not in my_set)
```

```
True
False
```

Iterating Through a Set

We can iterate through each item in a set using a for loop.

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

```
    p
    e
    a
    l
```

Set add()

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

Syntax of Set add() The syntax of add() method is:

set.add(elem)

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 add() method takes a single parameter:

elem - the element that is added to the set Return Value from Set add() add() method doesn't return any value and returns None.

```
prime_numbers = {2, 3, 5, 7}

# add 11 to prime_numbers
prime_numbers.add(11)


print(prime_numbers)

# Output: {2, 3, 5, 7, 11}

# 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)
```

```
    {2, 3, 5, 7, 11}
    Vowels are: {'a', 'u', 'e', 'o', 'i'}
    Vowels are: {'a', 'u', 'e', 'o', 'i'}
```

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

Set remove()

The remove() method removes the specified element from the set. Syntax of Set remove() The syntax of the remove() method is:

set.remove(element) remove() Parameters The remove() method takes a single element as an argument and removes it from the set.

Return Value from remove() The remove() removes the specified element from the set and updates the set. It doesn't return any value.

If the element passed to remove() doesn't exist, KeyError exception is thrown.

```
languages = {'Python', 'Java', 'English'}

# remove English from the set
languages.remove('English')


print(languages)

# Output: {'Python', 'Java'}

# remove English from the set, Which is already removed
languages.remove('English')

print(languages)

# Output: Error
```

```
{'Python', 'Java'}
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-23-476b845a85de> in <module>
     10 
     11 # remove English from the set, Which is already removed
---> 12 languages.remove('English')
     13 
     14 print(languages)

KeyError: 'English'
```

SEARCH STACK OVERFLOW

Set copy()

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

copy() Syntax The syntax of copy() method is:

set.copy() Here, the items of the set are copied.

copy() Parameters The copy() method doesn't take any parameters.

copy() Return Value The copy() method returns

the copy of the set

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

# copies the items of numbers to new_numbers
new_numbers = numbers.copy()

print(new_numbers)

# Output:  {1, 2, 3, 4}
```

Copy Set using = operator

```
names = {"John", "Charlie", "Marie"}

# copy set using = operator
new_names = names

print('Original Names: ', names)
print('Copied Names: ', new_names)
```

```
Original Names:  {'Charlie', 'Marie', 'John'}
Copied Names:  {'Charlie', 'Marie', 'John'}
```

Add items to the set after copy()

We can also modify the copied set using different methods.

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

print('numbers: ', numbers)

# add 5 to the copied set
new_numbers.add(5)

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

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

Set clear()

clear() Syntax The syntax of the clear() method is:

set.clear() Here, set.clear() clears the set by removing all the elements of the set.

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

clear() Return Value The clear() method doesn't return any value.

```
# set of vowels
vowels = {'a', 'e', 'i', 'o', 'u'}
print('Vowels (before clear):', vowels)

# clear vowels
vowels.clear()

print('Vowels (after clear):', vowels)


# set of strings
names = {'John', 'Mary', 'Charlie'}
print('Strings (before clear):', names)

# clear strings
names.clear()
print('Strings (after clear):', names)
```

```
Vowels (before clear): {'a', 'u', 'e', 'o', 'i'}
Vowels (after clear): set()
Strings (before clear): {'Mary', 'Charlie', 'John'}
Strings (after clear): set()
```

Set discard()

discard() Syntax The syntax of discard() method is:

a.discard(x) Here, a is the set and x is the item to discard.

discard() Parameter The discard method takes a single argument:

x - an item to remove from the set discard() Return Value The discard() method doesn't return any value.

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

print(numbers)
# removes 3 and returns the remaining set
result = numbers.discard(3)

print(result)
print(numbers)

# Output: numbers =  {2, 4, 5}
```

```
{2, 3, 4, 5}
None
{2, 4, 5}
```

Set pop() pop() Syntax The syntax of the pop() method is:

set.pop() Here, pop() removes an item from the set and updates it.

pop() Parameters The pop() method doesn't take any parameters.

pop() Return Value The pop() method returns:

a removed item from the set TypeError exception if the set is empty

```
A = {'a', 'b', 'c', 'd'}
removed_item = A.pop()
print(removed_item)

# Output: c


A = {'10', 'Ten', '100', 'Hundred'}

# removes random item of set A
print('Pop() removes:', A.pop())

# updates A to new set without the popped item
print('Updated set A:', A)



# empty set
A1 ={}

# throws an error
print(A1.pop())
```

```
b
Pop() removes: 10
Updated set A: {'Ten', '100', 'Hundred'}
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-31-decdc1be6e1c> in <module>
     20 
     21 # throws an error
---> 22 print(A1.pop())

TypeError: pop expected at least 1 arguments, got 0
```

SEARCH STACK OVERFLOW

Set difference()

The difference() method computes the difference of two sets and returns items that are unique to the first set.

difference() Syntax The syntax of the difference() method is:

A.difference(B) Here, A and B are two sets.

difference() Parameter The difference() method takes a single argument:

B - a set whose items are not included in the resulting set difference() Return Value The difference() method returns:

a set with elements unique to the first set

```
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', 'a', 'd'}
{'f', 'g'}
```

Set Difference Using - Operator We can also find the set difference using - operator in Python. For example,

```
A = {'a', 'b', 'c', 'd'}
B = {'c', 'f', 'g'}

# prints the items of A that are not present in B
print(A - B)

# prints the items of B that are not present in A
print(B - A)
```

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

Set difference_update()

The difference_update() method computes the difference between two sets (A - B) and updates set A with the resulting set.

```
# sets of numbers
A = {1, 3, 5, 7, 9}
B = {2, 3, 5, 7, 11}

# computes A - B and updates A with the resulting set
A.difference_update(B)

print('A = ', A)

# Output: A =  {1, 9}


A = {'a', 'c', 'g', 'd'}
B = {'c', 'f', 'g'}

print('A before (A - B) =', A)

A.difference_update(B)

print('A after (A - B) = ', A)
```

```
A =  {1, 9}
A before (A - B) = {'c', 'a', 'd', 'g'}
A after (A - B) =  {'a', 'd'}
```

Set intersection()

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

Syntax of Set intersection() The syntax of intersection() in Python is:

A.intersection(*other_sets) intersection() Parameters 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.

Return Value from Intersection() intersection() method returns the intersection of set A with all the sets (passed as argument).

If the argument is not passed to intersection(), it returns a shallow copy of the set (A).

```
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))  # Common to both AB  and C

print(A.intersection())
```

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

Set intersection_update() The intersection_update() finds the intersection of different sets and updates it to the set that calls the method.

intersection_update() Syntax The syntax of the intersection_update() method is:

A.intersection_update(*sets) Here, *sets indicates that set A can be intersected with one or more sets.

intersection_update() Parameter The intersection_update() method allows an random number of arguments:

*sets - denotes that the methods can take one or more arguments For example,

A.intersection_updata(B, C) Here, the method has two arguments, B and C.

intersection_update() Return Value The intersection_update() method doesn't return any value.

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

# updates set A with the items common to both sets A and B
A.intersection_update(B)

print('A =', A)

print('B =', B)
# Output: A = {2, 3, 4}




A = {1, 2, 3, 4}
B = {2, 3, 4, 5, 6}
C = {4, 5, 6, 9, 10}

# performs intersection between A, B and C and updates the result to set A
A.intersection_update(B, C)

print('A =', A)
print('B =', B)
print('C =', C)

# we have used intersection_update() to compute the intersection between sets A, B and C. The result of intersection is updated to s

# That's why we have A = {4} as the output because 4 is the only item that is present in all three sets. Whereas, sets B and C are u
```

```
    A = {2, 3, 4}
    B = {2, 3, 4, 5}
    A = {4}
    B = {2, 3, 4, 5, 6}
    C = {4, 5, 6, 9, 10}
```

symmetric_difference() The symmetric_difference() method returns all the items present in given sets, except the items in their intersections.

symmetric_difference() Syntax The syntax of the symmetric_difference() method is:

A.symmetric_difference(B) Here, A and B are two sets.

symmetric_difference() Parameter The symmetric_difference() method takes a single parameter:

B - a set that pairs with set A to find their symmetric difference symmetric_difference() Return Value The symmetric_difference() method returns:

a set with all the items of A and B excluding the excluding the identical items

```
A = {'Python', 'Java', 'Go'}
B = {'Python', 'JavaScript', 'C' }

# returns the symmetric difference of A and B to result variable
result = A.symmetric_difference(B)

print(result)


A = {'a', 'b', 'c'}
B = {'a', 'b', 'c'}

# returns empty set
result = A.symmetric_difference(B)

print(result)
```

```
    {'Go', 'Java', 'JavaScript', 'C'}
    set()
```

Symmetric Difference Using ^ Operator

```
A = {'a', 'b', 'c', 'd'}
B = {'c', 'd', 'e' }
C = {'i'}

# works as (A).symmetric_difference(B)
print(A ^ B)

# symmetric difference of 3 sets
print(A ^ B ^ C)
```

```
{'b', 'a', 'e'}
{'b', 'e', 'a', 'i'}
```

The symmetric_difference_update()

method finds the symmetric difference of two sets (non-similar items of both sets) and updates it to the set that calls the method.

symmetric_difference_update() Syntax The syntax of the symmetric_difference_update() method is:

A.symmetric_difference_update(B) Here, A and B are two sets.

symmetric_difference_update() Parameter The symmetric_difference_update() method takes a single parameter:

B - a set that pairs with set A to find non-similar items of both sets symmetric_difference_update() Return Value The symmetric_difference_update() method doesn't return any value.

```
# create two sets A and B
A = {'Python', 'Java', 'Go'}
B = {'Python', 'JavaScript', 'C' }

print('Original Set A:', A)

# updates A with the symmetric difference of A and B
A.symmetric_difference_update(B)

print('Updated Set A:', A)



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

# updates A with the symmetric difference of A and B
A.symmetric_difference_update(B)

print(A)

# Output: {a, e}
```

```
     Original Set A: {'Python', 'Java', 'Go'}
     Updated Set A: {'JavaScript', 'Java', 'Go', 'C'}
     {'e', 'a'}
```

Set union()

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

Syntax of Set union() The syntax of union() is:

A.union(*other_sets) Note: * is not part of the syntax. It is used to indicate that the method can take 0 or more arguments.

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.

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

Set update()

update() Syntax The syntax of the update() method is:

A.update(B) Here, A is a set and B can be any iterable like list, set, dictionary, string, etc.

update() Parameter The update() method can take any number of arguments. For example, A.update(B, C, D) Here,

B, C, D - iterables whose items are added to set A update() Return Value The update() method doesn't return any value.

```
A = {1, 3, 5}
B = {2, 4, 6}
C = {0}

print('Original A:', A)

# adds items of B and C to A and updates A
A.update(B, C)

print('A after update()', A)
```

```
Original A: {1, 3, 5}
A after update() {0, 1, 2, 3, 4, 5, 6}
```

```
# string
alphabet = 'odd'
```