Set

Set is an unordered collection of items. Every element is unique (no duplicates). However, the set itself is mutable (we can add or remove items). Sets can be used to perform mathematical set operations like union, intersection, symmetric difference etc.

A set is created by placing all the items (elements) inside curly braces {}, separated by comma or by using the built-in function set(). It can have any number of items and they may be of different types (integer, float, tuple, string etc.). But a set cannot have a mutable element, like list, set or dictionary, as its element.

>>> # set of integers

>>> my\_set = {1, 2, 3}

>>> # set of mixed datatypes

>>> my\_set = {1.0, "Hello", (1, 2, 3)}

>>> # set donot have duplicates

>>> {1,2,3,4,3,2}

{1, 2, 3, 4}

>>> # set cannot have mutable items

>>> my\_set = {1, 2, [3, 4]}

...

TypeError: unhashable type: 'list'

>>> # but we can make set from a list

>>> set([1,2,3,2])

{1, 2, 3}

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.

>>> a = {}

>>> type(a)

<class 'dict'>

>>> a = set()

>>> type(a)

<class 'set'>

Changing a Set in Python

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 elements using the method add(). Multiple elements can be added using update() method. The update() method can take tuples, lists, strings or other sets as its argument. In all cases, duplicates are avoided.

>>> my\_set = {1,3}

>>> my\_set[0]

...

TypeError: 'set' object does not support indexing

>>> my\_set.add(2)

>>> my\_set

{1, 2, 3}

>>> my\_set.update([2,3,4])

>>> my\_set

{1, 2, 3, 4}

>>> my\_set.update([4,5], {1,6,8})

>>> my\_set

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

Removing Elements from a Set

A particular item can be removed from set using methods like 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. The following example will illustrate this.

>>> my\_set = {1, 3, 4, 5, 6}

>>> my\_set.discard(4)

>>> my\_set

{1, 3, 5, 6}

>>> my\_set.remove(6)

>>> my\_set

{1, 3, 5}

>>> my\_set.discard(2)

>>> my\_set

{1, 3, 5}

>>> my\_set.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. We can also remove all items from a set using clear().

>>> my\_set = set("HelloWorld")

>>> my\_set.pop()

'r'

>>> my\_set.pop()

'W'

>>> my\_set

{'d', 'e', 'H', 'o', 'l'}

>>> my\_set.clear()

>>> my\_set

set()

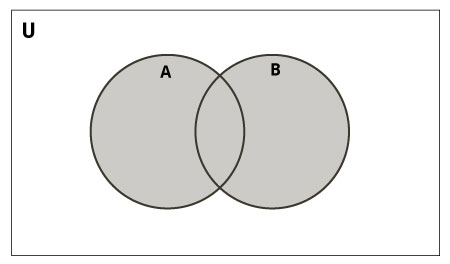
Python Set Operation

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}

Set Union



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().

>>> A | B

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

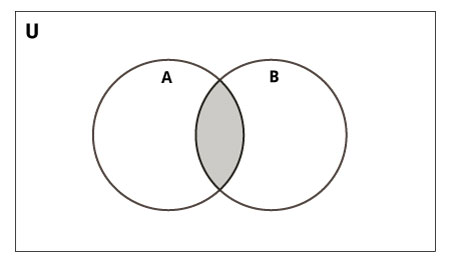
>>> A.union(B)

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

>>> B.union(A)

{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().

>>> A & B

{4, 5}

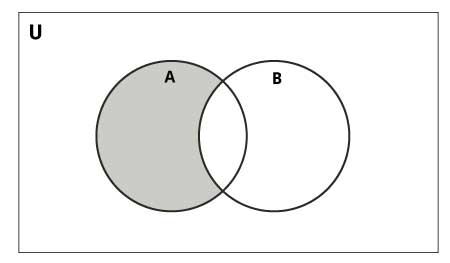
>>> A.intersection(B)

{4, 5}

>>> B.intersection(A)

{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().

>>> A - B

{1, 2, 3}

>>> A.difference(B)

{1, 2, 3}

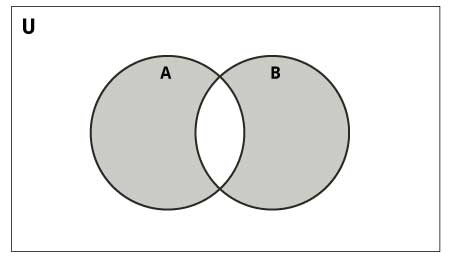
>>> B - A

{8, 6, 7}

>>> B.difference(A)

{8, 6, 7}

Set Symmetric Difference



Symmetric Difference of A and B is a set of element in both A and B except those common in both. Symmetric difference is performed using ^ operator. Same can be accomplished using the method symmetric\_difference().

>>> A ^ B

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

>>> A.symmetric\_difference(B)

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

>>> B.symmetric\_difference(A)

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

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 set objects.

|  |  |
| --- | --- |
| Python Set Methods | |
| Method | Description |
| add() | Add an element to a set |
| clear() | Remove all elemets form a set |
| copy() | Return a shallow copy of a set |
| difference() | Return the difference of two or more sets as a new set |
| difference\_update() | Remove all elements of another set from this set |
| discard() | Remove an element from set if it is a member. (Do nothing if the element is not in set) |
| intersection() | Return the intersection of two sets as a new set |
| intersection\_update() | Update the set with the intersection of itself and another |
| isdisjoint() | Return True if two sets have a null intersection |
| issubset() | Return True if another set contains this set |
| issuperset() | Return True if this set contains another set |
| pop() | Remove and return an arbitary set element. Raise KeyError if the set is empty |
| remove() | Remove an element from a set. It the element is not a member, raise a KeyError |
| symmetric\_difference() | Return the symmetric difference of two sets as a new set |
| symmetric\_difference\_update() | Update a set with the symmetric difference of itself and another |
| union() | Return the union of sets in a new set |
| update() | Update a set with the union of itself and others |

Other Set Operations

Set Membership Test

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

>>> my\_set = set("apple")

>>> 'a' in my\_set

True

>>> 'p' not in my\_set

False

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

set comprehension

>>> s = { x for x in range(10) }  
>>> s   
set([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])