A set is an unordered collection of items. Every set element is unique (no duplicates) and must be immutable (cannot be changed).

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.

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](https://www.programiz.com/python-programming/list), sets or [dictionaries](https://www.programiz.com/python-programming/dictionary) 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)

****Output****

{1, 2, 3}

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

Try the following examples as well.

# 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]}

****Output****

{1, 2, 3, 4}

{1, 2, 3}

Traceback (most recent call last):

File "<string>", line 15, in <module>

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.

# Distinguish set and dictionary while creating empty set

# initialize a with {}

a = {}

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

# initialize a with set()

a = set()

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

****Output****

<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](https://www.programiz.com/python-programming/tuple), lists, [strings](https://www.programiz.com/python-programming/string) 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

# 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)

****Output****

{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.

# 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)

****Output****

{1, 3, 4, 5, 6}

{1, 3, 5, 6}

{1, 3, 5}

{1, 3, 5}

Traceback (most recent call last):

File "<string>", line 28, in <module>

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.

# initialize my\_set# Output: set of unique elements

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

# pop an element# Output: random elementprint(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)

****Output****

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

H

{'r', 'W', 'o', 'd', 'e'}

set()

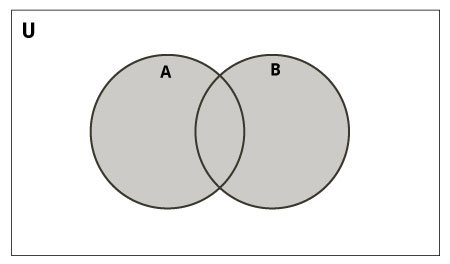
## Python Set Operations

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}

### **Set Union**

Set Union in Python

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)

****Output****

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

Try the following examples on Python shell.

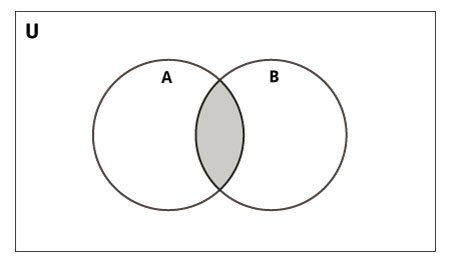
# use union function>>> A.union(B)

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

# use union function on B>>> B.union(A)

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

### **Set Intersection**

Set Intersection in Python

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)

****Output****

{4, 5}

Try the following examples on Python shell.

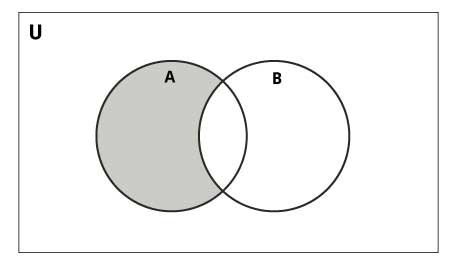
# use intersection function on A>>> A.intersection(B)

{4, 5}

# use intersection function on B>>> B.intersection(A)

{4, 5}

### **Set Difference**

Set Difference in Python

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)

****Output****

{1, 2, 3}

Try the following examples on Python shell.

# use difference function on A>>> A.difference(B)

{1, 2, 3}

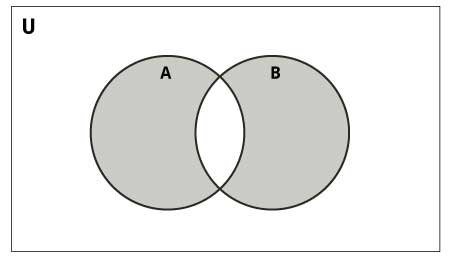
# use - operator on B>>> B - A

{8, 6, 7}

# use difference function on B>>> B.difference(A)

{8, 6, 7}

### **Set Symmetric Difference**

Set Symmetric Difference in Python

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)

****Output****

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

Try the following examples on Python shell.

# 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}