Python Sets

programiz.com/python-programming/set

Join our newsletter for the latest updates.

In this tutorial, you'll learn everything about Python sets; how they are created, adding or removing elements from them, and all operations performed on sets in Python.

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.

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)
```

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 a
print(type(a))

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

# check data type of a
print(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, 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

# 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 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)
```

Output

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

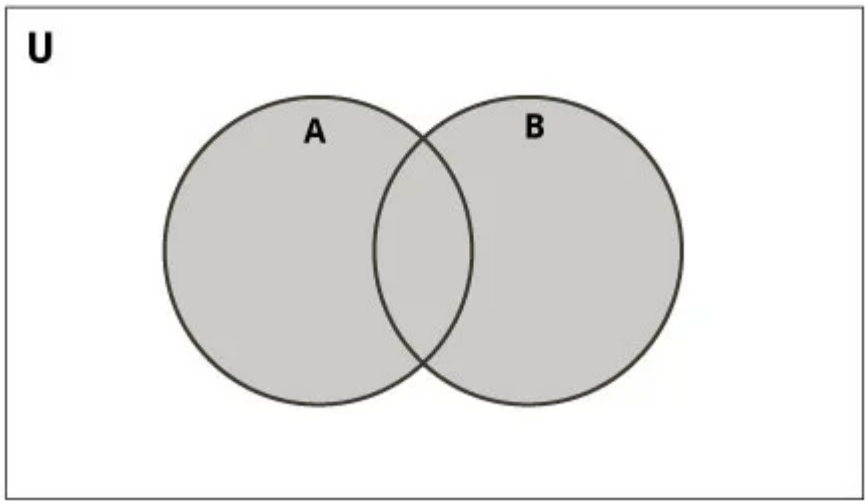
Python Set Operations

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

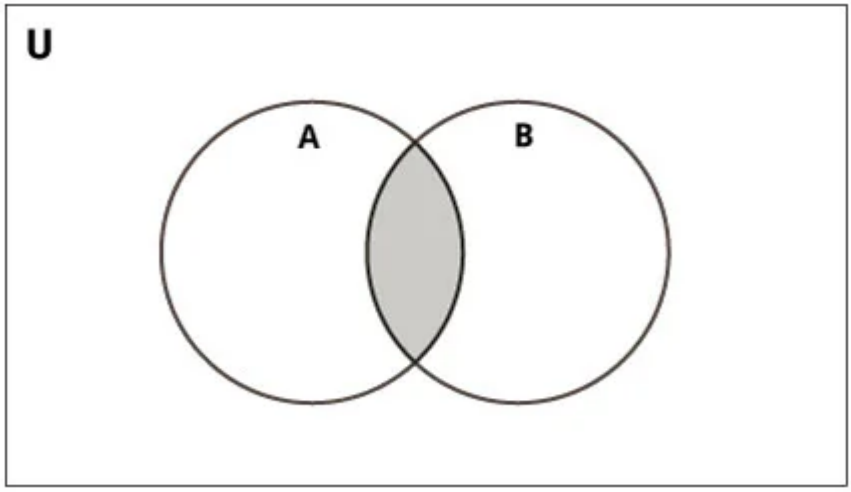
```
{1, 2, 3, 4, 5, 6, 7, 8}
```

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

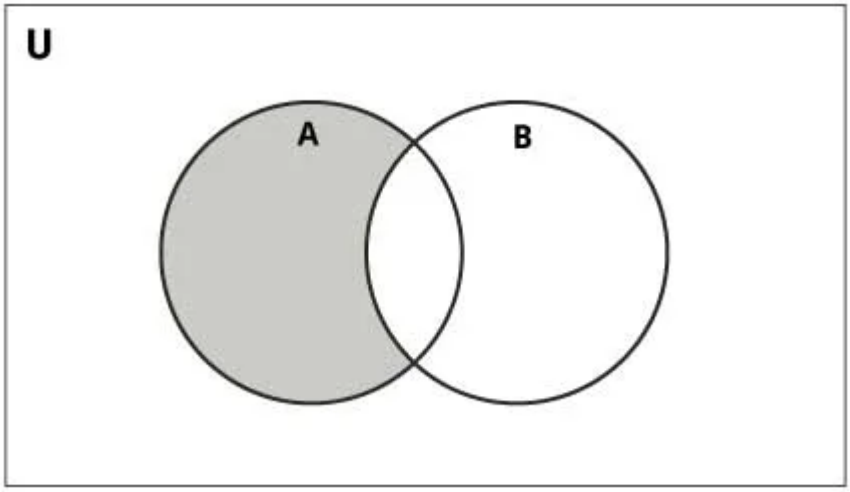
```
{4, 5}
```

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}
```

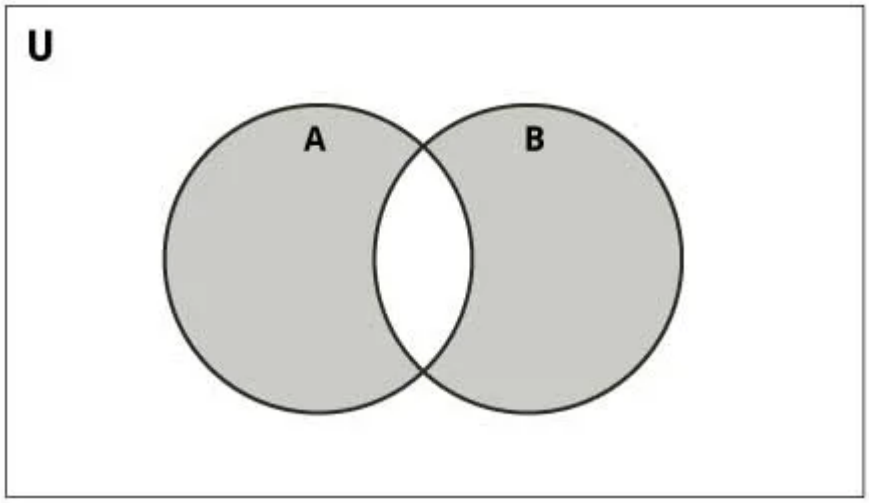
Try the following examples on Python shell.

```
# 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}
```

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

Other Set Operations

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)
```

Output

```
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)
...
a
p
e
l
```

Built-in Functions with Set

Built-in functions like `all()`, `any()`, `enumerate()`, `len()`, `max()`, `min()`, `sorted()`, `sum()` etc. are commonly used with sets to perform different tasks.

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.

Python Frozenset

Frozenset is a new class that has the characteristics of a set, but its elements cannot be changed once assigned. While tuples are immutable lists, frozensets are immutable sets.

Sets being mutable are unhashable, so they can't be used as dictionary keys. On the other hand, frozensets are hashable and can be used as keys to a dictionary.

Frozensets can be created using the frozenset() function.

This data type supports methods like `copy()`, `difference()`, `intersection()`, `isdisjoint()`, `issubset()`, `issuperset()`, `symmetric_difference()` and `union()`. Being immutable, it does not have methods that add or remove elements.

```
# Frozensets
# initialize A and B
A = frozenset([1, 2, 3, 4])
B = frozenset([3, 4, 5, 6])
```

Try these examples on Python shell.

```
>>> A.isdisjoint(B)
False
>>> A.difference(B)
frozenset({1, 2})
>>> A | B
frozenset({1, 2, 3, 4, 5, 6})
>>> A.add(3)
...
AttributeError: 'frozenset' object has no attribute 'add'
```