**Set in Python**

A set is an **unordered collection of items**. Every element is **unique (no duplicates**) and must be **immutable** (which cannot be changed). However, the **set itself is mutable**. We can add or remove items from it.

Set gives an advantage over List is that it has **a highly optimized method for checking whether a specific element is contained in the set**. This is based on a data structure known as a [hash table](https://www.geeksforgeeks.org/hashing-set-1-introduction/).

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

**Create a set**

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

# 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)

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***](https://www.programiz.com/python-programming/list)***, set or***[***dictionary***](https://www.programiz.com/python-programming/dictionary)***, as its element.***

#my\_set = {1, 2, [3, 4]} **# Cause an Error**

# we can make set from a list

# Output: {1, 2, 3}

my\_set = set([1,2,3,2]) **# List to Set Conversion**

print(my\_set)

**Creating an empty set is a bit tricky**

# initialize a with {}

a = {}

# check data type of a

# Output: <class 'dict'>

print(type(a))

# initialize a with set()

a = set()

# check data type of a

# Output: <class 'set'>

print(type(a))

**change a set in Python**

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

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

We can add 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)

# if you uncomment line 9,

# you will get an error

# TypeError: 'set' object does not support indexing

#my\_set[0] **# Indexing not possible**

# add an element

# Output: {1, 2, 3} **# add Method**

my\_set.add(2)

print(my\_set)

# add multiple elements

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

my\_set.update([2,3,4]) **# update Method**

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)

**Remove elements from a set**

A particular item can be removed from set using methods, **discard**() and **remove**().

Using discard() if the item does not exist in the set, it remains unchanged.

But remove() will raise an error in such condition.

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

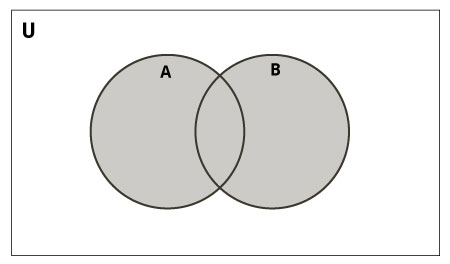
We can also remove all items from a set using clear().

We can also delete a set using del() function

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

Compute the union of two or more sets.

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().**

# 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)

# 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 in PythonSet 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**().

# initialize A and B

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

B = {4, 5, 6, 7, 8}

# use & operator

# Output: {4, 5}

print(A & B)

# use intersection function on A

>>> A.intersection(B)

{4, 5}

# use intersection function on B

>>> B.intersection(A)

{4, 5}

### Set Difference in PythonSet 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().

# 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)

# use - operator on B

>>> B – A {8, 6, 7}

# use difference function on A

>>> A.difference(B)

{1, 2, 3}

# use difference function on B

>>> B.difference(A)

{8, 6, 7}

### Set Symmetric Difference in PythonSet Symmetric Difference

Symmetric Difference of A and B is a set of elements in both A and B except those that are common in both.

Symmetric difference is performed **using ^ operator**. Same can be accomplished using the method **symmetric\_difference**().

# 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)

# 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}

### x1.isdisjoint(x2)

Determines whether or not two sets have any elements in common.

x1.isdisjoint(x2) returns True if x1 and x2 have no elements in common:

>>> If x1.isdisjoint(x2) is True, then x1 & x2 is the empty set:

>>> x1 = {1, 3, 5}

>>> x2 = {2, 4, 6}

>>> x1.isdisjoint(x2)

True

>>> x1 & x2

set()

Note: There is no operator that corresponds to the .isdisjoint() method.

### x1.issubset(x2)

x1 <= x2

Determine whether one set is a subset of the other.

In set theory, a set x1 is considered a subset of another set x2 if every element of x1 is in x2.

x1.issubset(x2) and x1 <= x2 return True if x1 is a subset of x2:

>>> x1 = {'foo', 'bar', 'baz'}

>>> x1.issubset({'foo', 'bar', 'baz', 'qux', 'quux'})

True

>>> x2 = {'baz', 'qux', 'quux'}

>>> x1 <= x2

False

### x1 < x2

Determines whether one set is a proper subset of the other.

A proper subset is the same as a subset, except that the sets can’t be identical. A set x1 is considered a proper subset of another set x2 if every element of x1 is in x2, and x1 and x2 are not equal.

x1 < x2 returns True if x1 is a proper subset of x2:

>>> x1 = {'foo', 'bar'}

>>> x2 = {'foo', 'bar', 'baz'}

>>> x1 < x2

True

>>> x1 = {'foo', 'bar', 'baz'}

>>> x2 = {'foo', 'bar', 'baz'}

>>> x1 < x2

False

Note: The < operator is the only way to test whether a set is a proper subset. There is no corresponding method.

### x1.issuperset(x2)

x1 >= x2

Determine whether one set is a superset of the other.

A superset is the reverse of a subset. A set x1 is considered a superset of another set x2 if x1 contains every element of x2.

x1.issuperset(x2) and x1 >= x2 return True if x1 is a superset of x2:

>>> x1 = {'foo', 'bar', 'baz'}

>>> x1.issuperset({'foo', 'bar'})

True

>>> x2 = {'baz', 'qux', 'quux'}

>>> x1 >= x2

False

You have already seen that a set is considered a subset of itself. A set is also considered a superset of itself:

>>> x = {1, 2, 3, 4, 5}

>>> x.issuperset(x)

True

>>> x >= x

True

### x1 > x2

Determines whether one set is a proper superset of the other.

A proper superset is the same as a superset, except that the sets can’t be identical. A set x1 is considered a proper superset of another set x2 if x1 contains every element of x2, and x1 and x2 are not equal.

x1 > x2 returns True if x1 is a proper superset of x2:

>>> x1 = {'foo', 'bar', 'baz'}

>>> x2 = {'foo', 'bar'}

>>> x1 > x2

True

>>> x1 = {'foo', 'bar', 'baz'}

>>> x2 = {'foo', 'bar', 'baz'}

>>> x1 > x2

False

A set is not a proper superset of itself:

>>> x = {1, 2, 3, 4, 5}

>>> x > x

False

Note: The > operator is the only way to test whether a set is a proper superset. There is no corresponding method.

### Modifying a Set

Although the elements contained in a set must be of immutable type, sets themselves can be modified. Like the operations above, there are a mix of operators and methods that can be used to change the contents of a set.

**Augmented Assignment Operators and Methods**

Each of the union, intersection, difference, and symmetric difference operators listed above has an augmented assignment form that can be used to modify a set. For each, there is a corresponding method as well.

**Modify a set by union.**

x1.update(x2) and x1 |= x2 add to x1 any elements in x2 that x1 does not already have:

>>> x1 = {'foo', 'bar', 'baz'}

>>> x2 = {'foo', 'baz', 'qux'}

>>> x1 |= x2

>>> x1

{'qux', 'foo', 'bar', 'baz'}

>>> x1.update(['corge', 'garply'])

>>> x1

{'qux', 'corge', 'garply', 'foo', 'bar', 'baz'}

**Modify a set by intersection.**

x1.intersection\_update(x2) and x1 &= x2 update x1, retaining only elements found in both x1 and x2:

>>> x1 = {'foo', 'bar', 'baz'}

>>> x2 = {'foo', 'baz', 'qux'}

>>> x1 &= x2

>>> x1

{'foo', 'baz'}

>>> x1.intersection\_update(['baz', 'qux'])

>>> x1

{'baz'}

**Modify a set by difference.**

x1.difference\_update(x2) and x1 -= x2 update x1, removing elements found in x2:

>>> x1 = {'foo', 'bar', 'baz'}

>>> x2 = {'foo', 'baz', 'qux'}

>>> x1 -= x2

>>> x1

{'bar'}

>>> x1.difference\_update(['foo', 'bar', 'qux'])

>>> x1

set()

**Modify a set by symmetric difference.**

x1.symmetric\_difference\_update(x2) and x1 ^= x2 update x1, retaining elements found in either x1 or x2, but not both:

>>> x1 = {'foo', 'bar', 'baz'}

>>> x2 = {'foo', 'baz', 'qux'}

>>>

>>> x1 ^= x2

>>> x1

{'bar', 'qux'}

>>>

>>> x1.symmetric\_difference\_update(['qux', 'corge'])

>>> x1

{'bar', 'corge'}

Programs:

|  |
| --- |
| # Creating a set using string  test\_set = set("geEks")    # Iterating using for loop  for val in test\_set:      print(val) |

**Output:**

k

s

e

g

E

**What is the output of the code shown below?**

|  |
| --- |
| set1 = {1, 2, 3}  set2 = set1.add(4)  print(set2) |

**Options:**

1. {1, 2, 3, 4}
2. {1, 2, 3}
3. Invalid Syntax
4. None

# Python Program to Count the Number of Vowels Present in a String using Sets