

Python Sets

```
In [1]: myset = {"apple", "banana", "cherry"}
```

Set

Sets are used to store multiple items in a single variable.

Set is one of 4 built-in data types in Python used to store collections of data, the other 3 are List, Tuple, and Dictionary, all with different qualities and usage.

A set is a collection which is unordered, unchangeable*, and unindexed.

* Note: Set items are unchangeable, but you can remove items and add new items.

Sets are written with curly brackets.

Example
Get your own Python Server
Create a Set:

```
In [2]: thisset = {"apple", "banana", "cherry"}  
print(thisset)
```

```
{'banana', 'cherry', 'apple'}
```

Set Items

Set items are unordered, unchangeable, and do not allow duplicate values.

Unordered

Unordered means that the items in a set do not have a defined order.

Set items can appear in a different order every time you use them, and cannot be referred to by index or key.

Unchangeable

Set items are unchangeable, meaning that we cannot change the items after the set has been created.

Once a set is created, you cannot change its items, but you can remove items and add new items.

Duplicates Not Allowed

Sets cannot have two items with the same value.

Example

Duplicate values will be ignored:

```
In [3]: thisset = {"apple", "banana", "cherry", "apple"}
```

```
print(thisset)
```

```
{'banana', 'cherry', 'apple'}
```

Note: The values True and 1 are considered the same value in sets, and are treated as duplicates:

Example

True and 1 is considered the same value:

```
In [4]: thisset = {"apple", "banana", "cherry", True, 1, 2}
```

```
print(thisset)
```

```
{True, 2, 'cherry', 'apple', 'banana'}
```

```
In [5]: thisset = {"apple", "banana", "cherry", True, 1, 2}
```

```
print(thisset)
```

```
{True, 2, 'cherry', 'apple', 'banana'}
```

Note: The values False and 0 are considered the same value in sets, and are treated as duplicates:

Example

False and 0 is considered the same value:

```
In [6]: thisset = {"apple", "banana", "cherry", False, True, 0}

print(thisset)

{False, True, 'cherry', 'apple', 'banana'}
```

Get the Length of a Set

To determine how many items a set has, use the len() function.

Example

Get the number of items in a set:

```
In [7]: thisset = {"apple", "banana", "cherry"}

print(len(thisset))

3
```

Set Items - Data Types

Set items can be of any data type:

Example

String, int and boolean data types:

```
In [8]: set1 = {"apple", "banana", "cherry"}
set2 = {1, 5, 7, 9, 3}
set3 = {True, False, False}
print(set1)
print(set2)
print(set3)

{'banana', 'cherry', 'apple'}
{1, 3, 5, 7, 9}
{False, True}
```

type()

From Python's perspective, sets are defined as objects with the data type 'set':

```
<class 'set'>
```

Example

What is the data type of a set?

```
In [9]: myset = {"apple", "banana", "cherry"}
print(type(myset))

<class 'set'>
```

The set() Constructor

It is also possible to use the set() constructor to make a set.

Example

Using the set() constructor to make a set:

```
In [10]: thisset = set(("apple", "banana", "cherry")) # note the double round-brackets
print(thisset)

{'banana', 'cherry', 'apple'}
```

Python Collections (Arrays)

There are four collection data types in the Python programming language:

List is a collection which is ordered and changeable. Allows duplicate members.

Tuple is a collection which is ordered and unchangeable. Allows duplicate members.

Set is a collection which is unordered, unchangeable*, and unindexed. No duplicate members.

Dictionary is a collection which is ordered** and changeable. No duplicate members.

**Set items are unchangeable, but you can remove items and add new items.*

****As of Python version 3.7, dictionaries are ordered. In Python 3.6 and earlier, dictionaries are unordered.**

When choosing a collection type, it is useful to understand the properties of that type. Choosing the right type for a particular data set could mean retention of meaning, and, it could mean an increase in efficiency or security.

Access Items

You cannot access items in a set by referring to an index or a key.

But you can loop through the set items using a for loop, or ask if a specified value is present in a set, by using the in keyword.

Example

Loop through the set, and print the values:

```
In [11]: thisset = {"apple", "banana", "cherry"}

for x in thisset:
    print(x)
```

```
banana
cherry
apple
```

Example

Check if "banana" is present in the set:

```
In [12]: thisset = {"apple", "banana", "cherry"}

print("banana" in thisset)
```

```
True
```

Example

Check if "banana" is NOT present in the set:

```
In [13]: thisset = {"apple", "banana", "cherry"}

print("banana" not in thisset)
```

```
False
```

Change Items

Once a set is created, you cannot change its items, but you can add new items.

Python - Add Set Items

Add Item

Once a set is created, you cannot change its items, but you can add new items.

To add one item to a set use the add() method.

Example

Get your own Python Server

Add an item to a set, using the add() method:

```
In [14]: thisset = {"apple", "banana", "cherry"}

thisset.add("orange")

print(thisset)
```

```
{'orange', 'banana', 'cherry', 'apple'}
```

Add Sets

To add items from another set into the current set, use the `update()` method.

Example

Add elements from tropical into thisset:

```
In [15]: thisset = {"apple", "banana", "cherry"}
         tropical = {"pineapple", "mango", "papaya"}

         thisset.update(tropical)

         print(thisset)

{'papaya', 'cherry', 'apple', 'mango', 'pineapple', 'banana'}
```

Add Any Iterable

The object in the `update()` method does not have to be a set, it can be any iterable object (tuples, lists, dictionaries etc.).

Example

Add elements of a list to a set:

```
In [16]: thisset = {"apple", "banana", "cherry"}
         mylist = ["kiwi", "orange"]

         thisset.update(mylist)

         print(thisset)

{'cherry', 'apple', 'kiwi', 'orange', 'banana'}
```

Remove Item

To remove an item in a set, use the `remove()`, or the `discard()` method.

Example

Remove "banana" by using the `remove()` method:

```
In [17]: thisset = {"apple", "banana", "cherry"}

         thisset.remove("banana")

         print(thisset)

{'cherry', 'apple'}
```

Note: If the item to remove does not exist, `remove()` will raise an error.

Example

Remove "banana" by using the `discard()` method:

```
In [18]: thisset = {"apple", "banana", "cherry"}

         thisset.discard("banana")

         print(thisset)

{'cherry', 'apple'}
```

Note: If the item to remove does not exist, `discard()` will NOT raise an error.

You can also use the `pop()` method to remove an item, but this method will remove a random item, so you cannot be sure what item that gets removed.

The return value of the `pop()` method is the removed item.

Example

Remove a random item by using the pop() method:

```
In [20]: thisset = {"apple", "banana", "cherry"}

x = thisset.pop()

print(x)

print(thisset)
```

```
banana
{'cherry', 'apple'}
```

Note: Sets are unordered, so when using the pop() method, you do not know which item that gets removed.

--

Example
The clear() method empties the set:

```
In [21]: thisset = {"apple", "banana", "cherry"}

thisset.clear()

print(thisset)

set()
```

Example
The del keyword will delete the set completely:

```
In [22]: thisset = {"apple", "banana", "cherry"}

del thisset

print(thisset)
```

```
-----
NameError                                Traceback (most recent call last)
Cell In[22], line 5
      1 thisset = {"apple", "banana", "cherry"}
      3 del thisset
----> 5 print(thisset)

NameError: name 'thisset' is not defined
```

Loop Items

--

You can loop through the set items by using a for loop:

ExampleGet your own Python Server
Loop through the set, and print the values:

```
In [23]: thisset = {"apple", "banana", "cherry"}

for x in thisset:
    print(x)
```

```
banana
cherry
apple
```

Join Sets

--

There are several ways to join two or more sets in Python.

The union() and update() methods joins all items from both sets.

The intersection() method keeps ONLY the duplicates.

The difference() method keeps the items from the first set that are not in the other set(s).

The symmetric_difference() method keeps all items EXCEPT the duplicates.

Union

The union() method returns a new set with all items from both sets.

ExampleGet your own Python Server
Join set1 and set2 into a new set:

```
In [24]: set1 = {"a", "b", "c"}
set2 = {1, 2, 3}

set3 = set1.union(set2)
print(set3)

{1, 2, 3, 'a', 'b', 'c'}
```

You can use the | operator instead of the union() method, and you will get the same result.

Example

```
In [25]: set1 = {"a", "b", "c"}
set2 = {1, 2, 3}

set3 = set1 | set2
print(set3)

{1, 2, 3, 'a', 'b', 'c'}
```

Join Multiple Sets

--

All the joining methods and operators can be used to join multiple sets.

When using a method, just add more sets in the parentheses, separated by commas:

Example

Join multiple sets with the union() method:

```
In [26]: set1 = {"a", "b", "c"}
set2 = {1, 2, 3}
set3 = {"John", "Elena"}
set4 = {"apple", "bananas", "cherry"}

myset = set1.union(set2, set3, set4)
print(myset)

{'John', 1, 2, 3, 'Elena', 'cherry', 'apple', 'b', 'bananas', 'a', 'c'}
```

When using the | operator, separate the sets with more | operators:

Example

Use | to join two sets:

```
In [27]: set1 = {"a", "b", "c"}
set2 = {1, 2, 3}
set3 = {"John", "Elena"}
set4 = {"apple", "bananas", "cherry"}

myset = set1 | set2 | set3 | set4
print(myset)

{'John', 1, 2, 3, 'Elena', 'cherry', 'apple', 'b', 'bananas', 'a', 'c'}
```

Join a Set and a Tuple

--

The union() method allows you to join a set with other data types, like lists or tuples.

The result will be a set.

Example

Join a set with a tuple:

```
In [28]: x = {"a", "b", "c"}
          y = (1, 2, 3)

          z = x.union(y)
          print(z)

          {1, 2, 3, 'b', 'a', 'c'}
```

Note: The `|` operator only allows you to join sets with sets, and not with other data types like you can with the `union()` method.

Update

The `update()` method inserts all items from one set into another.

The `update()` changes the original set, and does not return a new set.

Example

The `update()` method inserts the items in `set2` into `set1`:

```
In [29]: set1 = {"a", "b", "c"}
          set2 = {1, 2, 3}

          set1.update(set2)
          print(set1)

          {1, 2, 3, 'a', 'b', 'c'}
```

Note: Both union() and update() will exclude any duplicate items.

Intersection

Keep ONLY the duplicates

The `intersection()` method will return a new set, that only contains the items that are present in both sets.

Example

Join set1 and set2, but keep only the duplicates:

```
In [30]: set1 = {"apple", "banana", "cherry"}
set2 = {"google", "microsoft", "apple"}

set3 = set1.intersection(set2)
print(set3)

{'apple'}
```

You can use the `&` operator instead of the `intersection()` method, and you will get the same result.

Example
Use & to join two sets:

```
In [31]: set1 = {"apple", "banana", "cherry"}
          set2 = {"google", "microsoft", "apple"}

          set3 = set1 & set2
          print(set3)

          {'apple'}
```

Note: The `&` operator only allows you to join sets with sets, and not with other data types like you can with the `intersection()` method.

The `intersection_update()` method will also keep ONLY the duplicates, but it will change the original set instead of returning a new set.

Example

Keep the items that exist in both `set1`, and `set2`:

```
In [32]: set1 = {"apple", "banana", "cherry"}
set2 = {"google", "microsoft", "apple"}

set1.intersection_update(set2)

print(set1)

{'apple'}
```

The values `True` and `1` are considered the same value. The same goes for `False` and `0`.

Example

Join sets that contains the values `True`, `False`, `1`, and `0`, and see what is considered as duplicates:

```
In [33]: set1 = {"apple", 1, "banana", 0, "cherry"}
set2 = {False, "google", 1, "apple", 2, True}

set3 = set1.intersection(set2)

print(set3)

{False, 1, 'apple'}
```

Difference

--

The `difference()` method will return a new set that will contain only the items from the first set that are not present in the other set.

Example

Keep all items from `set1` that are not in `set2`:

```
In [34]: set1 = {"apple", "banana", "cherry"}
set2 = {"google", "microsoft", "apple"}

set3 = set1.difference(set2)

print(set3)

{'banana', 'cherry'}
```

You can use the `-` operator instead of the `difference()` method, and you will get the same result.

Example

Use `-` to join two sets:

```
In [35]: set1 = {"apple", "banana", "cherry"}
set2 = {"google", "microsoft", "apple"}

set3 = set1 - set2
print(set3)

{'banana', 'cherry'}
```

Note: The `-` operator only allows you to join sets with sets, and not with other data types like you can with the `difference()` method.

The `difference_update()` method will also keep the items from the first set that are not in the other set, but it will change the original set instead of returning a new set.

Example

Use the `difference_update()` method to keep the items that are not present in both sets:

```
In [36]: set1 = {"apple", "banana", "cherry"}
set2 = {"google", "microsoft", "apple"}

set1.difference_update(set2)

print(set1)

{'banana', 'cherry'}
```

Symmetric Differences

The `symmetric_difference()` method will keep only the elements that are NOT present in both sets.

Example

Keep the items that are not present in both sets:

```
In [37]: set1 = {"apple", "banana", "cherry"}
set2 = {"google", "microsoft", "apple"}

set3 = set1.symmetric_difference(set2)

print(set3)

{'cherry', 'google', 'microsoft', 'banana'}
```

You can use the `^` operator instead of the `symmetric_difference()` method, and you will get the same result.

Example

Use `^` to join two sets:

```
In [38]: set1 = {"apple", "banana", "cherry"}
set2 = {"google", "microsoft", "apple"}

set3 = set1 ^ set2
print(set3)

{'cherry', 'google', 'microsoft', 'banana'}
```

Note: The `^` operator only allows you to join sets with sets, and not with other data types like you can with the `symmetric_difference()` method.

The `symmetric_difference_update()` method will also keep all but the duplicates, but it will change the original set instead of returning a new set.

Example

Use the `symmetric_difference_update()` method to keep the items that are not present in both sets:

```
In [39]: set1 = {"apple", "banana", "cherry"}
set2 = {"google", "microsoft", "apple"}

set1.symmetric_difference_update(set2)

print(set1)

{'microsoft', 'banana', 'cherry', 'google'}
```

Set Methods

--

Python has a set of built-in methods that you can use on sets.

Method	Shortcut	Description
<code>add()</code>		Adds an element to the set
<code>clear()</code>		Removes all the elements from the set
<code>copy()</code>		Returns a copy of the set
<code>difference()</code>	-	Returns a set containing the difference between two or more sets
<code>difference_update()</code>	--	Removes the items in this set that are also included in another, specified set
<code>discard()</code>		Remove the specified item

intersection() sets	&	Returns a set, that is the intersection of two other sets
intersection_update() other, specified set(s)	&=	Removes the items in this set that are not present in other, specified set(s)
isdisjoint()		Returns whether two sets have a intersection or not
issubset()	<=	Returns whether another set contains this set or not
	<	Returns whether all items in this set is present in
other, specified set(s)		
issuperset()	>=	Returns whether this set contains another set or not
	>	Returns whether all items in other, specified set(s) is
present in this set		
pop()		Removes an element from the set randomly
remove()		Removes the specified element
symmetric_difference()	^	Returns a set with the symmetric differences of two sets
symmetric_difference_update() another	^=	Inserts the symmetric differences from this set and another
union()		Return a set containing the union of sets
update()	=	Update the set with the union of this set and others