Python Set

Author : Srishti Sawla

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

thisset = {"apple", "banana", "cherry"}

Note: Sets are unordered, so you cannot be sure in which order the items will appear.

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:

thisset = {"apple", "banana", "cherry", "apple"}

Get the Length of a Set

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

Set Items - Data Types

Set items can be of any data type:

Example

String, int and boolean data types:

```
set1 = {"apple", "banana", "cherry"}
set2 = {1, 5, 7, 9, 3}
set3 = {True, False, False}
```

A set can contain different data types:

Example

A set with strings, integers and boolean values:

```
set1 = {"abc", 34, True, 40, "male"}
```

type()

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

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

The set() Constructor

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

Access Items

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.

Loop through the set, and print the values:

```
thisset = {"apple", "banana", "cherry"}
```

Change Items

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

Add Items

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

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

```
thisset = {"apple", "banana", "cherry"}

thisset.add("orange")

print(thisset)
```

Add Sets

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

Example

Add elements from tropical into thisset:

```
thisset = {"apple", "banana", "cherry"}
tropical = {"pineapple", "mango", "papaya"}

thisset.update(tropical)

print(thisset)
```

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.).

Remove Item

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

Example

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

```
thisset = {"apple", "banana", "cherry"}

thisset.remove("banana")

print(thisset)
```

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

Example

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

```
thisset = {"apple", "banana", "cherry"}

thisset.discard("banana")

print(thisset)
```

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 the *last* item. Remember that sets are unordered, so you will not know what item that gets removed.

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

Example

Remove the last item by using the pop() method:

```
thisset = {"apple", "banana", "cherry"}

x = thisset.pop()

print(x)

print(thisset)
```

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:

```
thisset = {"apple", "banana", "cherry"}

thisset.clear()

print(thisset)
```

Example

The del keyword will delete the set completely:

```
thisset = {"apple", "banana", "cherry"}

del thisset

print(thisset)
```

Loop Items

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

Join Two Sets

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

You can use the union() method that returns a new set containing all items from both sets, or the update() method that inserts all the items from one set into another:

Example

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

```
set1 = {"a", "b" , "c"}
set2 = {1, 2, 3}

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

Try it Yourself »

Example

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

```
set1 = {"a", "b" , "c"}
set2 = {1, 2, 3}

set1.update(set2)
print(set1)
```

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

Keep ONLY the Duplicates

The **intersection_update()** method will keep only the items that are present in both sets.

Example

Keep the items that exist in both set x, and set y:

```
x = {"apple", "banana", "cherry"}
y = {"google", "microsoft", "apple"}
```

x.intersection_update(y)

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

Example

Return a set that contains the items that exist in both set x, and set y:

```
x = {"apple", "banana", "cherry"}
y = {"google", "microsoft", "apple"}

z = x.intersection(y)

print(z)
```

Keep All, But NOT the Duplicates

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

Example

Keep the items that are not present in both sets:

```
x = {"apple", "banana", "cherry"}
y = {"google", "microsoft", "apple"}
```

```
x.symmetric_difference_update(y)

print(x)
```

The symmetric_difference() method will return a new set, that contains only the elements that are NOT present in both sets.

Example

Return a set that contains all items from both sets, except items that are present in both:

```
x = {"apple", "banana", "cherry"}
y = {"google", "microsoft", "apple"}

z = x.symmetric_difference(y)

print(z)
```

Set built in functions

Python - Set Methods (w3schools.com)