Python - Join Sets

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.

Example

Join set1 and set2 into a new set:

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

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

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

Example

Use | to join two sets:

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

set3 = set1 | set2
print(set3)
```

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:

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

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

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

Example

Use | to join two sets:

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

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

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:

```
x = {"a", "b", "c"}
y = (1, 2, 3)
```

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

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:

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

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

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:

```
set1 = {"apple", "banana", "cherry"}
set2 = {"google", "microsoft", "apple"}

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

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

Example

Use & to join two sets:

```
set1 = {"apple", "banana", "cherry"}
set2 = {"google", "microsoft", "apple"}

set3 = set1 & set2
print(set3)
```

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:

```
set1 = {"apple", "banana", "cherry"}
set2 = {"google", "microsoft", "apple"}

set1.intersection_update(set2)

print(set1)
```

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:

```
set1 = {"apple", 1,  "banana", 0, "cherry"}
set2 = {False, "google", 1, "apple", 2, True}

set3 = set1.intersection(set2)

print(set3)
```

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:

```
set1 = {"apple", "banana", "cherry"}
set2 = {"google", "microsoft", "apple"}

set3 = set1.difference(set2)

print(set3)
```

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

Example

Use - to join two sets:

```
set1 = {"apple", "banana", "cherry"}
set2 = {"google", "microsoft", "apple"}

set3 = set1 - set2
print(set3)
```

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:

```
set1 = {"apple", "banana", "cherry"}
set2 = {"google", "microsoft", "apple"}
```

```
set1.difference_update(set2)

print(set1)
```

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:

```
set1 = {"apple", "banana", "cherry"}
set2 = {"google", "microsoft", "apple"}

set3 = set1.symmetric_difference(set2)

print(set3)
```

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

Example

Use ^ to join two sets:

```
set1 = {"apple", "banana", "cherry"}
set2 = {"google", "microsoft", "apple"}

set3 = set1 ^ set2
print(set3)
```

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:

```
set1 = {"apple", "banana", "cherry"}
set2 = {"google", "microsoft", "apple"}

set1.symmetric_difference_update(set2)

print(set1)
```

Python - Set Methods

Set Methods

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

Method	Shortcut	Description
add()		Adds an element to the set
clear()		Removes all the elements from the set
copy()		Returns a copy of the set
difference()	-	Returns a set containing the difference between two o sets
difference_update()	-=	Removes the items in this set that are also included in specified set
discard()		Remove the specified item
intersection()	&	Returns a set, that is the intersection of two other sets
intersection_update()	&=	Removes the items in this set that are not present in ot 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 set(s)
issuperset()	>=	Returns whether this set contains another set or not
	>	Returns whether all items in other, specified set(s) is p this set
pop()		Removes an element from the set
remove()		Removes the specified element
symmetric_difference()	^	Returns a set with the symmetric differences of two se
symmetric_difference_update()	^=	Inserts the symmetric differences from this set and an
union()	|	Return a set containing the union of sets
update()	|=	Update the set with the union of this set and others

Python - Dictionary

Dictionaries are used to store data values in key:value pairs.

A dictionary is a collection which is ordered*, changeable and do not allow duplicates.

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

Dictionaries are written with curly brackets, and have keys and values:

Example

Create and print a dictionary:

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
print(thisdict)
```

Dictionary Items

Dictionary items are ordered, changeable, and do not allow duplicates.

Dictionary items are presented in key:value pairs, and can be referred to by using the key name.

Example

Print the "brand" value of the dictionary:

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
print(thisdict["brand"])
```

Ordered or Unordered?

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

When we say that dictionaries are ordered, it means that the items have a defined order, and that order will not change.

Unordered means that the items do not have a defined order, you cannot refer to an item by using an index.

Changeable

Dictionaries are changeable, meaning that we can change, add or remove items after the dictionary has been created.

Duplicates Not Allowed

Dictionaries cannot have two items with the same key:

Example

Duplicate values will overwrite existing values:

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964,
  "year": 2020
}
print(thisdict)
```

Dictionary Length

To determine how many items a dictionary has, use the len() function:

Example

Print the number of items in the dictionary:

```
print(len(thisdict))
```

Dictionary Items - Data Types

The values in dictionary items can be of any data type:

Example

String, int, boolean, and list data types:

```
thisdict = {
  "brand": "Ford",
  "electric": False,
  "year": 1964,
  "colors": ["red", "white", "blue"]
}
```

type()

From Python's perspective, dictionaries are defined as objects with the data type 'dict':

```
<class 'dict'>
```

Example

Print the data type of a dictionary:

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
print(type(thisdict))
```

The dict() Constructor

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

Example

Using the dict() method to make a dictionary:

```
thisdict = dict(name = "John", age = 36, country = "Norway")
```

Python - Access Dictionary Items

Accessing Items

You can access the items of a dictionary by referring to its key name, inside square brackets:

Example

Get the value of the "model" key:

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
x = thisdict["model"]
```

There is also a method called get() that will give you the same result:

Example

Get the value of the "model" key:

```
x = thisdict.get("model")
```

Get Keys

The keys() method will return a list of all the keys in the dictionary.

Example

Get a list of the keys:

```
x = thisdict.keys()
```

The list of the keys is a *view* of the dictionary, meaning that any changes done to the dictionary will be reflected in the keys list.

Example

Add a new item to the original dictionary, and see that the keys list gets updated as well:

```
car = {
"brand": "Ford",
"model": "Mustang",
"year": 1964
}

x = car.keys()

print(x) #before the change

car["color"] = "white"

print(x) #after the change
```

Get Values

The values() method will return a list of all the values in the dictionary.

Example

Get a list of the values:

```
x = thisdict.values()
```

The list of the values is a *view* of the dictionary, meaning that any changes done to the dictionary will be reflected in the values list.

Example

Make a change in the original dictionary, and see that the values list gets updated as well:

```
car = {
"brand": "Ford",
"model": "Mustang",
"year": 1964
}

x = car.values()

print(x) #before the change

car["year"] = 2020

print(x) #after the change
```

Example

Add a new item to the original dictionary, and see that the values list gets updated as well:

```
car = {
"brand": "Ford",
"model": "Mustang",
"year": 1964
}

x = car.values()
```

```
print(x) #before the change

car["color"] = "red"

print(x) #after the change
```

Get Items

The items() method will return each item in a dictionary, as tuples in a list.

Example

Get a list of the key:value pairs

```
x = thisdict.items()
```

The returned list is a *view* of the items of the dictionary, meaning that any changes done to the dictionary will be reflected in the items list.

Example

Make a change in the original dictionary, and see that the items list gets updated as well:

```
car = {
"brand": "Ford",
"model": "Mustang",
"year": 1964
}

x = car.items()

print(x) #before the change

car["year"] = 2020

print(x) #after the change
```

Example

Add a new item to the original dictionary, and see that the items list gets updated as well:

```
car = {
"brand": "Ford",
"model": "Mustang",
"year": 1964
}

x = car.items()

print(x) #before the change

car["color"] = "red"

print(x) #after the change
```