Set Methods in Python | Python Tutorial - Day #32

Show Course Contents

Overview Q&A Downloads Announcements

Day 32 - Joining Sets & Set Methods

Sets in python more or less work in the same way as sets in mathematics. We can perform operations like union and intersection on the sets just like in mathematics.

1. union() and update():

Example:

```
cities = {"Tokyo", "Madrid", "Berlin", "Delhi"}
cities2 = {"Tokyo", "Seoul", "Kabul", "Madrid"}
cities3 = cities.union(cities2)
print(cities3)
```

Output:

```
{'Tokyo', 'Madrid', 'Kabul', 'Seoul', 'Berlin', 'Delhi'}
```

Copy

Example:

```
cities = {"Tokyo", "Madrid", "Berlin", "Delhi"}
cities2 = {"Tokyo", "Seoul", "Kabul", "Madrid"}
cities.update(cities2)
print(cities)
```

Output:

```
{'Berlin', 'Madrid', 'Tokyo', 'Delhi', 'Kabul', 'Seoul'}
```

II. intersection and intersection_update():

The intersection() and intersection_update() methods prints only items that are similar to both the sets. The intersection() method returns a new set whereas intersection_update() method updates into the existing set from another set.

Example:

```
cities = {"Tokyo", "Madrid", "Berlin", "Delhi"}
cities2 = {"Tokyo", "Seoul", "Kabul", "Madrid"}
cities3 = cities.intersection(cities2)
print(cities3)
```

Output:

```
{'Madrid', 'Tokyo'}
```

Example :

```
cities = {"Tokyo", "Madrid", "Berlin", "Delhi"}
cities2 = {"Tokyo", "Seoul", "Kabul", "Madrid"}
cities.intersection_update(cities2)
print(cities)
```

Output:

```
{'Tokyo', 'Madrid'}
```

III. symmetric_difference and symmetric_difference_update():

The symmetric_difference() and symmetric_difference_update() methods prints only items that are not similar to both the sets. The symmetric_difference() method returns a new set whereas symmetric_difference_update() method updates into the existing set from another set.

Example:

```
cities = {"Tokyo", "Madrid", "Berlin", "Delhi"}
cities2 = {"Tokyo", "Seoul", "Kabul", "Madrid"}
cities3 = cities.symmetric_difference(cities2)
print(cities3)
```

Output:

```
{'Seoul', 'Kabul', 'Berlin', 'Delhi'}
```

Example:

```
cities = {"Tokyo", "Madrid", "Berlin", "Delhi"}
cities2 = {"Tokyo", "Seoul", "Kabul", "Madrid"}
cities.symmetric_difference_update(cities2)
print(cities)
```

Output:

```
{'Kabul', 'Delhi', 'Berlin', 'Seoul'}
```

IV. difference() and difference_update():

The difference() and difference_update() methods prints only items that are only present in the original set and not in both the sets. The difference() method returns a new set whereas difference_update() method updates into the existing set from another set.

Example:

```
cities = {"Tokyo", "Madrid", "Berlin", "Delhi"}
cities2 = {"Seoul", "Kabul", "Delhi"}
cities3 = cities.difference(cities2)
print(cities3)
```

Output:

```
{'Tokyo', 'Madrid', 'Berlin'}
```

Example:

```
cities = {"Tokyo", "Madrid", "Berlin", "Delhi"}
cities2 = {"Seoul", "Kabul", "Delhi"}
print(cities.difference(cities2))
```

Output:

```
{'Tokyo', 'Berlin', 'Madrid'}
```

Set Methods

There are several in-built methods used for the manipulation of set.They are explained below

isdisjoint():

The isdisjoint() method checks if items of given set are present in another set. This method returns False if items are present, else it returns True.

Example:

```
cities = {"Tokyo", "Madrid", "Berlin", "Delhi"}
cities2 = {"Tokyo", "Seoul", "Kabul", "Madrid"}
print(cities.isdisjoint(cities2))
```

Output:

```
False
```

issuperset():

The issuperset() method checks if all the items of a particular set are present in the original set. It returns True if all the items are present, else it returns False.

Example:

```
cities = {"Tokyo", "Madrid", "Berlin", "Delhi"}
cities2 = {"Seoul", "Kabul"}
print(cities.issuperset(cities2))
cities3 = {"Seoul", "Madrid","Kabul"}
print(cities.issuperset(cities3))
```

Output:

```
False
False
```

issubset():

The issubset() method checks if all the items of the original set are present in the particular set. It returns True if all the items are present, else it returns False.

Example:

```
cities = {"Tokyo", "Madrid", "Berlin", "Delhi"}
cities2 = {"Delhi", "Madrid"}
print(cities2.issubset(cities))
```

Output:

```
True
```

add()

If you want to add a single item to the set use the add() method.

Example:

```
cities = {"Tokyo", "Madrid", "Berlin", "Delhi"}
cities.add("Helsinki")
print(cities)
```

Output:

```
{'Tokyo', 'Helsinki', 'Madrid', 'Berlin', 'Delhi'}
```

update()

If you want to add more than one item, simply create another set or any other iterable object(list, tuple, dictionary), and use the update() method to add it into the existing set.

Example:

```
cities = {"Tokyo", "Madrid", "Berlin", "Delhi"}
cities2 = {"Helsinki", "Warsaw", "Seoul"}
cities.update(cities2)
print(cities)
```

Output:

```
{'Seoul', 'Berlin', 'Delhi', 'Tokyo', 'Warsaw', 'Helsinki', 'Madr
```

remove()/discard()

We can use remove() and discard() methods to remove items form list.

Example :

```
cities = {"Tokyo", "Madrid", "Berlin", "Delhi"}
cities2 = {"Helsinki", "Warsaw", "Seoul"}
cities.update(cities2)
print(cities)
```

Output:

```
{'Delhi', 'Berlin', 'Madrid'}
```

The main difference between remove and discard is that, if we try to delete an item which is not present in set, then remove() raises an error, whereas discard() does not raise any error.

Example:

```
cities = {"Tokyo", "Madrid", "Berlin", "Delhi"}
cities.remove("Seoul")
print(cities)
```

Output:

```
KeyError: 'Seoul'
```

pop()

This method removes the last item of the set but the catch is that we don't know which item gets popped as sets are unordered. However, you can access the popped item if you assign the pop() method to a variable.

Example:

```
cities = {"Tokyo", "Madrid", "Berlin", "Delhi"}
item = cities.pop()
print(cities)
print(item)
```

Output:

```
{'Tokyo', 'Delhi', 'Berlin'} Madrid
```

del

del is not a method, rather it is a keyword which deletes the set entirely.

Example:

```
cities = {"Tokyo", "Madrid", "Berlin", "Delhi"}
del cities
print(cities)
```

Output:

NameError: name 'cities' is not defined We get an error because our entire set has been deleted and there is no variable called cities which contains a set.
What if we don't want to delete the entire set, we just want to delete all items within that set?

clear():

This method clears all items in the set and prints an empty set.

Example:

```
cities = {"Tokyo", "Madrid", "Berlin", "Delhi"}
cities.clear()
print(cities)
```

Output:

```
set()
```

Check if item exists

You can also check if an item exists in the set or not.

Example:

```
info = {"Carla", 19, False, 5.9}
if "Carla" in info:
    print("Carla is present.")
else:
    print("Carla is absent.")
```

Output:

```
Carla is present.
```

Next Lesson>>

< < Previous

Next > >

CodeWithHarry