Python SETS

S1MCA

SETS

- 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*.
- Sets are written with curly brackets ({}).

```
>>>    thisset = {"apple", "banana", "cherry"}
       print(thisset)
```

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.

- **Duplicates Not Allowed**

Sets cannot have two items with the same value.

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

       print(thisset)
```

→Duplicate values will be ignored:

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

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

- True and 1 is considered the same value

```
>>>    thisset = {"apple", "banana", "cherry", True, 1, 2}

       print(thisset)
```

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

False and 0 is considered the same value

```
>>>    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.

Get the number of items in a set:

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

       print(len(thisset))
```

Set Items - Data Types

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

- 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.

```
thisset = set(("apple", "banana", "cherry")) # note the double round-brackets
print(thisset)
```

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.

```
>>>thisset = {"apple", "banana", "cherry"}
for x in thisset:
  print(x)
```

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

print("banana" in thisset)
```

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.

- >>>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.

```
>>>    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.).

```
>>>    thisset = {"apple", "banana", "cherry"}
       mylist = ["kiwi", "orange"]

       thisset.update(mylist)

       print(thisset)
```

REMOVE ITEM

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

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

       thisset.remove("banana")

       print(thisset)
```

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

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

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

       thisset.discard("banana")

       print(thisset)
```

→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
- Remove a random item by using the pop() method

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

       x = thisset.pop()

       print(x)

       print(thisset)
```

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

- The clear() method empties the set

```
>>>    thisset = {"apple", "banana", "cherry"}
       thisset.clear()
       print(thisset)
```

- The del keyword will delete the set completely

```
>>>    thisset = {"apple", "banana", "cherry"}
       del thisset
       print(thisset)
```

Loop Sets

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

>>> Loop through the set, and print the values:

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

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

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.

>>> 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.

>>>

```
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:

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

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

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

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

→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.

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

```
>>>    set1 = {"a", "b" , "c"}
       set2 = {1, 2, 3}
       set1.update(set2)
       print(set1)
```

→ 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.

```
>>>     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.

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

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

→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.

```
>>>    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.

```
>>>    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.

```
>>>     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.

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

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

→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.

```
>>>    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.

```
>>>    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.

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

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

→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.

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)
```

- **Set Methods**

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 or more sets
difference_update()	-=	Removes the items in this set that are also included in another, 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 other, specified set(s)
isdisjoint()		Returns whether two sets have a intersection or not
issubset()	<=	Returns True if all items of this set is present in another set
	<	Returns True if all items of this set is present in another, *larger* set
issuperset()	>=	Returns True if all items of another set is present in this set
	>	Returns True if all items of another, *smaller* set is present in 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 sets
symmetric_difference_update()	^=	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