**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](https://www.w3schools.com/python/python_lists.asp), [Tuple](https://www.w3schools.com/python/python_tuples.asp), and [Dictionary](https://www.w3schools.com/python/python_dictionaries.asp), all with different qualities and usage.

A set is a collection which is both unordered and unindexed.

Sets are written with curly brackets.

Ex:

s={1,2,3,4}  
print(type(s))  
print("set:",s)  
t=set(range(5))  
print("range values of set:",t)

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 :**

Sets 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 add new items.

**Duplicates Not Allowed**

Ex:

l=eval(input("enter list of values"))  
s=set(l)  
print(s)

output:

"C:\Users\sekha\PycharmProjects\set data structure\venv\Scripts\python.exe" "C:/Users/sekha/PycharmProjects/set data structure/duplicate list.py"

enter list of values["apple","apple","mab","cat"]

{'apple', 'mab', 'cat'}

**Len():**

* Get the Length of a Set
* To determine how many items a set has, use the len() method.

Ex:-

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

print(len(thisset))

output:3

**Set Items - Data Types**

* **Set items can be of any data type:**

Ex:

set1 = {"apple", "banana", "cherry"}

set2 = {1, 5, 7, 9, 3}

set3 = {True, False, False}

print(set1)

print(set2)

print(set3)

output:

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

{1, 3, 5, 7, 9}

{False, True}

**set()** **Constructor :**

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

Ex:-

thisset = set(("apple", "banana", "cherry"))

print(thisset)

Output:

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

**Note: the set list is unordered, so the result will display the items in a random order.**

Python Collections (Arrays)

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

[List](https://www.w3schools.com/python/python_lists.asp) is a collection which is ordered and changeable. Allows duplicate members.

[Tuple](https://www.w3schools.com/python/python_tuples.asp) is a collection which is ordered and unchangeable. Allows duplicate members.

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

[Dictionary](https://www.w3schools.com/python/python_dictionaries.asp) is a collection which is ordered\* and changeable. No duplicate members.

**Python - Access Set Items**

**Ex: for loop using**

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

for x in thisset:

print(x)

**output:**

banana  
apple  
cherry

**Check if "banana" is present in the set:**

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

print("banana" in thisset)

**output:**

True.

**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.

Ex:

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

thisset.add("orange")

print(thisset)

**output:**

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

**Add Sets**

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

Ex:

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

tropical = {"pineapple", "mango", "papaya"}

thisset.update(tropical)

print(thisset)

**output:**

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

**Add Any Iterable**

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

**Ex:- Add elements of a list to at set:**

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

mylist = ["kiwi", "orange"]

thisset.update(mylist)

print(thisset)  
**output:**

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

**Python - Remove Set Items**

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

Ex:

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

thisset.remove("banana")

print(thisset)

**output:**

{'cherry', 'apple'}

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

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

Ex:

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

thisset.discard("banana")

print(thisset)

**output:**

{'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 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

Ex:

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

x = thisset.pop()

print(x) #removed item

print(thisset) #the set after removal

**output:**

cherry  
{'apple', 'banana'}

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

**clear():-**

* **method empties the set:**

Ex:

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

thisset.clear()

print(thisset)

**output:**

**set()**

**del ():**

* keyword will delete the set completely:

Ex:

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

del thisset

print(thisset) #this will raise an error because the set no longer exists

**output:**

Traceback (most recent call last):  
  File "demo\_set\_del.py", line 5, in <module>  
    print(thisset) #this will raise an error because the set no longer exists  
NameError: name 'thisset' is not defined

**Python - Loop Sets**

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

Ex:-

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

for x in thisset:

print(x)

**output:**

apple  
cherry  
banana

**Python - Join Sets**

* 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:

**union()** :-

* method returns a new set with all items from both sets:

Ex:

set1 = {"a", "b" , "c"}

set2 = {1, 2, 3}

set3 = set1.union(set2)

print(set3)

output:

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

**update():**

* method inserts the items in set2 into set1

Ex:

set1 = {"a", "b" , "c"}

set2 = {1, 2, 3}

set1.update(set2)

print(set1)

**output:**

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

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

keep ONLY the Duplicates

**intersection\_update() :**

* will keep only the items that are present in both sets.

Ex:

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

y = {"google", "microsoft", "apple"}

x.intersection\_update(y)

print(x)

**output:**

{‘apple’}

**intersection()** :-

* will return a *new* set, that only contains the items that are present in both sets.

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

y = {"google", "microsoft", "apple"}

z = x.intersection(y)

print(z)

**output:**

{'apple'}

Keep All, But NOT the Duplicates

**symmetric\_difference\_update()**:

* will keep only the elements that are NOT present in both sets.

**Keep the items that are not present in both sets**

Ex:

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

y = {"google", "microsoft", "apple"}

x.symmetric\_difference\_update(y)

print(x)

**output:**

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

**symmetric\_difference():**

* method will return a new set, that contains only the elements that are NOT present in both sets.

Ex: 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)

**output:**

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

**Python Set add() Method**

**Add():**

**Definition and Usage:**

The add() method adds an element to the set.

If the element already exists, the add() method does not add the element.

**Syntax**

***set*.add(*elmnt*)**

Parameter Values

|  |  |
| --- | --- |
| Parameter | Description |
| *Elmnt* | Required. The element to add to the set |

Ex:

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

thisset.add("orange")

print(thisset)

**output:**

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

**Ex: Try to add an element that already exists**

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

thisset.add("apple")

print(thisset)

**output:**

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

**Set clear() Method:**

* Remove all elements from the fruits set

**Definition and Usage:**

**The clear() method removes all elements in a set.**

**Syntax**

***set*.clear()**

Ex:

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

thisset.clear()

print(thisset)

**output:**

set()

**Set copy() Method:**

Definition and Usage:

**The copy() method copies the set.**

**Syntax**

***set*.copy()**

Ex:

fruits = {"apple", "banana", "cherry"}  
x = fruits.copy()  
print(x)

**output:**

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

**Set difference() Method:**

**Definition and Usage :**

* The difference() method returns a set that contains the difference between two sets.
* Meaning: The returned set contains items that exist only in the first set, and not in both sets.

**Syntax**

***set*.difference(*set*)**

Parameter Values

|  |  |
| --- | --- |
| Parameter | Description |
| *Set* | Required. The set to check for differences in |

Ex:

**Return a set that contains the items that only exist in set x, and not in set y**

x = {"apple", "banana", "cherry"}  
y = {"google", "microsoft", "apple"}  
z = x.difference(y)  
print(z)

**Output:**

{'banana', 'cherry'}

Ex:1

**Reverse the first example. Return a set that contains the items that only exist in set y, and not in set x**x = {"apple", "banana", "cherry"}  
y = {"google", "microsoft", "apple"}  
z = y.difference(x)  
print(z)

**Output:**

{'google', 'microsoft'}

**Set difference\_update() Method :**

**Definition and Usage:**

* The difference\_update() method removes the items that exist in both sets.
* The difference\_update() method is different from the difference() method, because the difference() method *returns a new set*, without the unwanted items, and the difference\_update() method *removes* the unwanted items from the original set.

**Syntax**

***set*.difference\_update(*set*)**

Parameter Values

|  |  |
| --- | --- |
| Parameter | Description |
| *Set* | Required. The set to check for differences in |

Ex:

x = {"apple", "banana", "cherry"}  
y = {"google", "microsoft", "apple"}  
x.difference\_update(y)  
print(x)

**Output:**

{'banana', 'cherry'}

**Set discard() Method:**

**Definition and Usage:**

* The discard() method removes the specified item from the set.
* This method is different from the remove() method, because the remove() method *will raise an error* if the specified item does not exist, and the discard() method *will not*.

**Syntax**

***set*.discard(*value*)**

Parameter Values

|  |  |
| --- | --- |
| Parameter | Description |
| *Value* | Required. The item to search for, and remove |

Ex;

thisset = {"apple", "banana", "cherry"}  
thisset.discard("banana")  
print(thisset)

**Output:**

{'apple', 'cherry'}

**Set intersection() Method:**

**Definition and Usage**

* The intersection() method returns a set that contains the similarity between two or more sets.
* Meaning: The returned set contains only items that exist in both sets, or in all sets if the comparison is done with more than two sets.

**Syntax**

***set*.intersection(*set1, set2 ... etc*)**

Parameter Values

|  |  |  |
| --- | --- | --- |
| Parameter |  | Description |
| *set1* |  | Required. The set to search for equal items in |
| *set2* |  | Optional The other set to search for equal items  in. You can compare as many sets you like. Separate the sets with a comma |

Ex:

x = {"apple", "banana", "cherry"}  
y = {"google", "microsoft", "apple"}  
z = x.intersection(y)  
print(z)

Output:

{'apple'}

Ex: Compare 3 sets, and return a set with items that is present in all 3 sets

x = {"a", "b", "c"}  
y = {"c", "d", "e"}  
z = {"f", "g", "c"}  
result = x.intersection(y, z)  
print(result)

**Output:**

{'c'}

**Python Set intersection\_update() Method:**

**Definition and Usage:**

* The intersection\_update() method removes the items that is not present in both sets (or in all sets if the comparison is done between more than two sets).
* The intersection\_update() method is different from the intersection() method, because the intersection() method *returns a new set*, without the unwanted items, and the intersection\_update() method *removes* the unwanted items from the original set.

**Syntax**

***set*.intersection\_update(*set1, set2 ... etc*)**

Parameter Values

|  |  |
| --- | --- |
| Parameter | Description |
| *set1* | Required. The set to search for equal items in |
| *set2* | Optional. The other set to search for equal items  in. You can compare as many sets you like. Separate the sets with a comma |

Ex:

x = {"apple", "banana", "cherry"}  
y = {"google", "microsoft", "apple"}  
x.intersection\_update(y)  
print(x)  
Ex:Compare 3 sets, and return a set with items that is present in all 3 sets  
x = {"a", "b", "c"}  
y = {"c", "d", "e"}  
z = {"f", "g", "c"}  
x.intersection\_update(y, z)  
print(x)

**output:**

{'apple'}

{'c'}

**Set pop() Method:**

**Definition and Usage:**

* The pop() method removes a random item from the set.
* This method returns the removed item.

**Syntax**

**set.pop()**

Ex:**Remove a random item from the set**  
fruits = {"apple", "banana", "cherry"}  
fruits.pop()  
print(fruits)  
  
**Ex:Return the removed element**  
fruits = {"apple", "banana", "cherry"}  
x = fruits.pop()  
print(x)

**Output:**

{'banana', 'cherry'}

Apple

**Set remove() Method:**

**Definition and Usage:**

* The remove() method removes the specified element from the set.
* This method is different from the discard() method, because the remove() method will raise an error if the specified item does not exist, and the discard() method will not.

**Syntax**

**set.remove(item)**

Parameter Values

|  |  |
| --- | --- |
| Parameter | Description |
| Item | Required. The item to search for, and remove |

Ex:

fruits = {"apple", "banana", "cherry"}  
fruits.remove("banana")  
print(fruits)

**output:**

{'apple', 'cherry'}

**Set union() Method:**

**Definition and Usage:**

* The union() method returns a set that contains all items from the original set, and all items from the specified set(s).
* You can specify as many sets you want, separated by commas.
* It does not have to be a set, it can be any iterable object.
* If an item is present in more than one set, the result will contain only one appearance of this item.

**Syntax**

**set.union(set1, set2...)**

Parameter Values

|  |  |
| --- | --- |
| Parameter | Description |
| set1 | Required. The iterable to unify with |
| set2 | Optional. The other iterable to unify with. You can compare as many iterables as you like. Separate each iterable with a comma |

Ex:Return a set that contains all items from both sets, duplicates are excluded  
x = {"apple", "banana", "cherry"}  
y = {"google", "microsoft", "apple"}  
z = x.union(y)  
print(z)

**Output:**

{'google', 'microsoft', 'apple', 'banana', 'cherry'}  
Ex: unify more than 2 sets.  
x = {"a", "b", "c"}  
y = {"f", "d", "a"}  
z = {"c", "d", "e"}  
result = x.union(y, z)   
print(result)

**Output**:

{'a', 'c', 'd', 'e', 'b', 'f'}

**Set update() Method:**

**Definition and Usage:**

* The update() method updates the current set, by adding items from another set.
* If an item is present in both sets, only one appearance of this item will be present in the updated set.

**Syntax**

***set*.update(*set*)**

Parameter Values

|  |  |
| --- | --- |
| Parameter | Description |
| *Set* | Required. The set insert into the current set |

Ex:Insert the items from set y into set x  
x = {"apple", "banana", "cherry"}  
y = {"google", "microsoft", "apple"}  
x.update(y)  
print(x)

**Output:**

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

**Set symmetric\_difference() Method:**

**Definition and Usage:**

* The symmetric\_difference() method returns a set that contains all items from both set, but not the items that are present in both sets.
* Meaning: The returned set contains a mix of items that are not present in both sets.

**Syntax**

***set*.symmetric\_difference(*set*)**

Parameter Values

|  |  |
| --- | --- |
| Parameter | Description |
| *Set* | Required. The set to check for matches in |

Ex:

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

y = {"google", "microsoft", "apple"}

z = x.symmetric\_difference(y)

print(z)

**output:**

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

**Set symmetric\_difference\_update() Method :**

**Definition and Usage:**

The symmetric\_difference\_update() method updates the original set by removing items that are present in both sets, and inserting the other items.

Syntax

set.symmetric\_difference\_update(set)

Parameter Values

|  |  |
| --- | --- |
| Parameter | Description |
| Set | Required. The set to check for matches in |

Ex:

x = {"apple", "banana", "cherry"}  
y = {"google", "microsoft", "apple"}  
x.symmetric\_difference\_update(y)  
print(x)

output:

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

# **Set issuperset() Method:**

## Definition and Usage:

The issuperset() method returns True if all items in the specified set exists in the original set, otherwise it retuns False.

## Syntax

***set*.issuperset(*set***)

## Parameter Values

|  |  |
| --- | --- |
| Parameter | Description |
| *Set* | Required. The set to search for equal items in |

Ex:Return True if all items set y are present in set x  
x = {"f", "e", "d", "c", "b", "a"}  
y = {"a", "b", "c"}  
z = x.issuperset(y)  
print(z)

Output:

True

Ex: Return False if not all items in set y are present in set x

x = {"f", "e", "d", "c", "b"}

y = {"a", "b", "c"}

z = x.issuperset(y)

print(z)

Output:

False

# **Set issubset() Method:**

## Definition and Usage:

The issubset() method returns True if all items in the set exists in the specified set, otherwise it retuns False.

## Syntax

***set*.issubset(*set*)**

## Parameter Values

|  |  |
| --- | --- |
| Parameter | Description |
| *Set* | Required. The set to search for equal items in |

Ex:Return True if all items set x are present in set y  
x = {"a", "b", "c"}  
y = {"f", "e", "d", "c", "b", "a"}  
  
z = x.issubset(y)  
print(z)  
 **output:**

**True**

Ex 1:Return False if not all items in set x are present in set y  
x = {"a", "b", "c"}  
y = {"f", "e", "d", "c", "b"}  
z = x.issubset(y)  
print(z)

**Output:**

**False**