Python Sets

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

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, Tuple, and Dictionary, all with different qualities and usage.

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.

```
In [1]: # ExampleGet your own Python Server
        # Create a Set:

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

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

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.

In [2]:
```
# Example
# Duplicate values will be ignored:

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

print(thisset)
# Note: The values True and 1 are considered the same value in sets,
# and are treated as duplicates:
```

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

In [3]:
```
# Example
# True and 1 is considered the same value:

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

print(thisset)
```

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

Get the Length of a Set

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

In [ ]:
```
# Example
# Get the number of items in a set:

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

print(len(thisset))
```

Set Items - Data Types

Set items can be of any data type:

In [ ]:
```
Example
String, int and boolean data types:
```

In [4]:
```python
# 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:

In [5]:
```python
# 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'>

In [6]:
```python
# Example
# What is the data type of a set?

myset = {"apple", "banana", "cherry"}
print(type(myset))
```

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

The set() Constructor

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

In [7]:
```python
# Example
# Using the set() constructor to make a set:

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

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

Python Collections (Arrays)

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

- List is a collection which is ordered and changeable. Allows duplicate members.

- Tuple is a collection which is ordered and unchangeable. Allows duplicate members.

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

- Dictionary is a collection which is ordered** and changeable. No

*Set items are unchangeable, but you can remove items and add new items.

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

When choosing a collection type, it is useful to understand the properties of that type. Choosing the right type for a particular data set could mean retention of meaning, and, it could mean an increase in efficiency or security.

Python - Access Set Items

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.

In [8]:
```
# ExampleGet your own Python Server
# Loop through the set, and print the values:

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

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

```
cherry
banana
apple
```

Example

Check if "banana" is present in the set:

In [ ]:
```
thisset = {"apple", "banana", "cherry"}

print("banana" in thisset)
```

Change Items

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

Python - Add Set 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.

In [12]:
```
# <!-- ExampleGet your own Python Server
# Add an item to a set, using the add() method:
#  -->
thisset = {"apple", "banana", "cherry"}

thisset.add("orange")

print(thisset)
```

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

Add Sets

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

In [13]:
```
# Example
# Add elements from tropical into thisset:

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

thisset.update(tropical)

print(thisset)
```

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

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.).

In [14]:
```
# Example
# Add elements of a list to at set:

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

thisset.update(mylist)

print(thisset)
```

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

Python - Remove Set Items

Remove Item

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

In [15]:
```
# ExampleGet your own Python Server
# 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.
```

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

In [16]:
```
# 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 erro
```

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

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.

In [17]:
```
# Example
# Remove a random 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.
```

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

In [18]:
```
# Example
# The clear() method empties the set:

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

thisset.clear()

print(thisset)
```

```
set()
```

In [19]:
```
# Example
# The del keyword will delete the set completely:

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

del thisset

print(thisset)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_552\882802990.py in <module>
      6 del thisset
      7 
----> 8 print(thisset)

NameError: name 'thisset' is not defined
```

Python - Loop Sets

Loop Items

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

In [20]:
```
# <!-- ExampleGet your own Python Server
# Loop through the set, and print the values:
#  -->
thisset = {"apple", "banana", "cherry"}

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

```
cherry
banana
apple
```

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:

In [21]:
```
# ExampleGet your own Python Server
# 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)
```

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

In [ ]:
```
# 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.

In [23]:
```python
# 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)

print(x)
```

```
{'apple'}
```

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

In [24]:
```python
# 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)
```

```
{'apple'}
```

Keep All, But NOT the Duplicates

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

In [25]:
```python
# 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)
```

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

The symmetric_difference() method will return a new set, that contains only the elements

that are NOT present in both sets

In [26]:
```
# 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)
```

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

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

In [27]:
```
# Example
# True and 1 is considered the same value:

x = {"apple", "banana", "cherry", True}
y = {"google", 1, "apple", 2}


z = x.symmetric_difference(y)
```

In [28]:
```
z
```

Out[28]:
```
{2, 'banana', 'cherry', 'google'}
```

Python - Set Methods

Set Methods

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

Method 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 whether another set contains this set or not

- issuperset() Returns whether this set contains another set or not

- 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

In [ ]: