Sets

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 both unordered and unindexed.

Sets are written with curly brackets.

Set items are:

- unordered
- unchangeable
- do not allow duplicate values

In [1]:
```python
# create a set
thisset = {"apple", "banana", "cherry"}
print(thisset)
```

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

In [2]:
```python
# duplicate values will be ignored
thisset = {"apple", "banana", "cherry", "apple"}
print(thisset)
```

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

Access Set 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 [3]:
```
thisset = {"apple", "banana", "cherry"}
for x in thisset:
    print(x)
```

```
cherry
banana
apple
```

In [4]:
```
# Check if "banana" is present in the set:

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

```
True
```

Add items to set

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

In [5]:
```
# To add one item to a set use the add() method.
thisset = {"apple", "banana", "cherry"}
thisset.add("orange")
print(thisset)
```

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

In [6]:
```
# 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)
```

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

In [7]:
```python
# The object in the update() method does not have 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)
```

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

Remove Item

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

In [8]:
```python
thisset = {"apple", "banana", "cherry"}
thisset.remove("banana")
print(thisset)
```

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

In [9]:
```python
# Note: If the item to remove does not exist, remove() will raise an error.
thisset.remove("orange")
print(thisset)
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-9-2aa27a28b3db> in <module>
      1 # Note: If the item to remove does not exist, remove() will raise an error.
----> 2 thisset.remove("orange")
      3 print(thisset)

KeyError: 'orange'
```

In [10]:
```
# Remove "banana" by using the discard() method:
thisset = {"apple", "banana", "cherry"}
thisset.discard("banana")
print(thisset)
```

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

In [11]:
```
# Note: If the item to remove does not exist, discard() will NOT raise an error.
thisset.discard("orange")
print(thisset)

```

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

In [12]:
```
# 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.

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

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

In [13]:
```
# The clear() method empties the set:

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

```
set()
```

In [14]:
```python
# The del keyword will delete the set completely:

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

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-14-a80624893d06> in <module>
      3 thisset = {"apple", "banana", "cherry"}
      4 del thisset
----> 5 print(thisset)

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

Loop Items

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

In [15]:
```python
thisset = {"apple", "banana", "cherry"}

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

```
cherry
banana
apple
```

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.

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

In [16]:
```
# 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, 'b', 'a', 'c'}
```

In [17]:
```
# The update() method inserts the items in set2 into set1:

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

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

Keep ONLY the Duplicates

The intersection_update() method will keep only the items that are present in both sets.

In [18]:
```
# 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'}
```

In [19]:
```
# 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 [20]:
```
# 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', 'microsoft', 'google', 'cherry'}
```

In [21]:
```
# 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', 'microsoft', 'google', 'cherry'}
```

Set Methods

In [22]:
```python
# The copy() method copies the set.

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

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

In [23]:
```python
# 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 set
x = {"apple", "banana", "cherry"}
y = {"google", "microsoft", "apple"}
z = x.difference(y)
print(z)
```

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

In [24]:
```python
# The issubset() method returns True if all items in the set exists in the specified set, otherw
# returns False.
x = {"a", "b", "c"}
y = {"f", "e", "d", "c", "b", "a"}
z = x.issubset(y)
print(z)
```

```
True
```

In [25]:
```python
# The issuperset() method returns True if all items in the specified set exists in the original
# otherwise it returns False.
x = {"f", "e", "d", "c", "b", "a"}
y = {"a", "b", "c"}
z = x.issuperset(y)
print(z)
```

```
True
```

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

```
False
```

In [ ]:
```

```