# Python 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
- Sets are written with curly brackets { }
- Example
    - Create a Set

In [4]:

```
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
- Example
    - Duplicate values will be ignored

In [5]:

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

print(thisset)
```

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

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

In [6]:

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

print(thisset)
```

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

# Get the Length of a Set

- To determine how many items a set has, use the len() function
- Example
  - Get the number of items in a set

In [7]:

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

print(len(thisset))
```

```
3
```

# Set Items - Data Types

- Set items can be of any data type
- Example
  - String, int and boolean data types

In [9]:

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

print(set1)
print(set2)
print(set3)
```

```
{'cherry', 'banana', 'apple'}
{1, 3, 5, 7, 9}
{False, True}
```

- A set can contain different data types
- Example
  - A set with strings, integers and boolean values

In [10]:

```
set1 = {"abc", 34, True, 40, "male"}
print(set1)
```

```
{True, 34, 'male', 40, 'abc'}
```

# The set() Constructor

- It is also possible to use the set() constructor to make a set
- Example
  - Using the set() constructor to make a set

In [11]:

```
thisset = set(("apple", "banana", "cherry")) # note the double round-brac
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.
- Example
  - Loop through the set, and print the values

In [14]:

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

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

```
cherry
banana
apple
```

- Example
  - Check if "banana" is present in the set

In [13]:

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

print("banana" in thisset)
```

```
True
```

# Change Items

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

# Add Set Items

- To add one item to a set use the add() method
- Syntax set.add(elmnt)
    - elmnt - Required. The element to add to the set
- Example
    - Add an item to a set, using the add() method

In [15]:

```
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
- Syntax set.update(set)
    - set - Required. The iterable insert into the current set
- Example
    - Add elements from tropical into thisset

In [16]:

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

thisset.update(tropical)

print(thisset)
```

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

# 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.).
- Example
    - Add elements of a list to at set

In [17]:

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

thisset.update(mylist)

print(thisset)
```

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

# Remove Set Items

- To remove an item in a set, use the remove(), or the discard() method
- Syntax for remove set.remove(item)
  - item - Required. The item to search for, and remove
- Syntax for discard set.discard(value)
  - value - Required. The item to search for, and remove
- Example
  - Remove "banana" by using the remove() method

In [18]:

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

thisset.remove("banana")

print(thisset)
```

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

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

- Example
  - Remove "banana" by using the discard() method

In [29]:

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

thisset.discard("banana")

print(thisset)
```

```
{'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 a random item, so you cannot be sure what item that gets removed.
- The return value of the pop() method is the removed item
- Syntax set.pop()
- No parameter values
- Example
  - Remove a random item by using the pop() method

In [3]:

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

x = thisset.pop()

print(x)
print(thisset)
```

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

- The clear() method empties the set
- Syntax set.clear()
- No parameter values
- Example

In [4]:

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

thisset.clear()

print(thisset)
```

```
set()
```

- The del keyword will delete the set completely
- Example

In [5]:

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

del thisset

print(thisset)
```

```
--------------------------------------------------------------------------
---
NameError                                 Traceback (most recent call la
st)
Cell In[5], line 5
      1 thisset = {"apple", "banana", "cherry"}
      3 del thisset
----> 5 print(thisset)

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

# 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
- Syntax set.union(set1, set2...)
  - 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

- Example
    - The union() method returns a new set with all items from both sets

In [6]:

```
set1 = {"a", "b" , "c"}
set2 = {1, 2, 3}

set3 = set1.union(set2)
print(set3)
```

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

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

In [ ]:

```
set1 = {"a", "b" , "c"}
set2 = {1, 2, 3}

set1.update(set2)
print(set1)
```

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

# Set Methods

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

- 1.copy()
- The copy() method copies the set
- Syntax - The set.copy()
- No parameters
- Example
    - Copy the fruits set

In [7]:

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

x = fruits.copy()

print(x)
```

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

- 2.difference()
- 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 - The set.difference(set)
    - set - Required. The set to check for differences in

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

In [9]:

```
x = {"apple", "banana", "cherry"}
y = {"google", "microsoft", "apple"}

z = x.difference(y)

print(z)
```

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

- Example
    - Reverse the first example. Return a set that contains the items that only exist in set y, and not in set x

In [11]:

```
x = {"apple", "banana", "cherry"}
y = {"google", "microsoft", "apple"}

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

```
{'google', 'microsoft'}
```

- 3.difference_update()
- 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 - The set.difference_update(set)
    - set - Required. The set to check for differences in
- Example
    - Remove the items that exist in both sets

In [12]:

```
x = {"apple", "banana", "cherry"}
y = {"google", "microsoft", "apple"}

x.difference_update(y)

print(x)
```

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

- 4.intersection()
- 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 - The set.intersection(set1, set2 ... etc)
  - 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

- Example
  - Return a set that contains the items that exist in both set x, and set y

In [13]:

```
x = {"apple", "banana", "cherry"}
y = {"google", "microsoft", "apple"}

z = x.intersection(y)

print(z)
```

```
{'apple'}
```

- Example
  - Compare 3 sets, and return a set with items that is present in all 3 sets

In [14]:

```
x = {"a", "b", "c"}
y = {"c", "d", "e"}
z = {"f", "g", "c"}

result = x.intersection(y, z)

print(result)
```

```
{'c'}
```

- 5.intersection_update()
- 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 - The set.intersection_update(set1, set2 ... etc)
  - 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
- Example
  - Remove the items that is not present in both x and y

In [15]:

```
x = {"apple", "banana", "cherry"}
y = {"google", "microsoft", "apple"}

x.intersection_update(y)
```

```
print(x)
```

```
{'apple'}
```

- Example
  - Compare 3 sets, and return a set with items that is present in all 3 sets

In [19]:

```
x = {"a", "b", "c"}
y = {"c", "d", "e"}
z = {"f", "g", "c"}

x.intersection_update(y, z)

print(x)
```

```
{'c'}
```

- 6.isdisjoint()
- The isdisjoint() method returns True if none of the items are present in both sets, otherwise it returns False
- Syntax - The set.isdisjoint(set)
  - set - Required. The set to search for equal items in
- Example
  - Return True if no items in set x is present in set y

In [20]:

```
x = {"apple", "banana", "cherry"}
y = {"google", "microsoft", "facebook"}

z = x.isdisjoint(y)

print(z)
```

```
True
```

- Example
  - Return False if one or more items are present in both sets

In [21]:

```
x = {"apple", "banana", "cherry"}
y = {"google", "microsoft", "apple"}

z = x.isdisjoint(y)

print(z)
```

```
False
```

- 7.issubset()
- The issubset() method returns True if all items in the set exists in the specified set, otherwise it returns False
- Syntax - The set.issubset(set)
  - set- Required. The set to search for equal items in

- Example
    - Return True if all items in set x are present in set y

In [22]:

```
x = {"a", "b", "c"}
y = {"f", "e", "d", "c", "b", "a"}

z = x.issubset(y)

print(z)
```

```
True
```

- Example
    - What if not all items are present in the specified set?
    - Return False if not all items in set x are present in set y

In [25]:

```
x = {"a", "b", "c"}
y = {"f", "e", "d", "c", "b"}

z = x.issubset(y)

print(z)
```

```
False
```

- 8.issuperset()
- The issuperset() method returns True if all items in the specified set exists in the original set, otherwise it returns False
- Syntax - The set.issuperset(set)
    - set - Required. The set to search for equal items in
- Example
    - Return True if all items set y are present in set x

In [24]:

```
x = {"f", "e", "d", "c", "b", "a"}
y = {"a", "b", "c"}

z = x.issuperset(y)

print(z)
```

```
True
```

- Example
    - What if not all items are present in the specified set?
    - Return False if not all items in set y are present in set x

In [26]:

```
x = {"f", "e", "d", "c", "b"}
y = {"a", "b", "c"}

z = x.issuperset(y)

print(z)
```

```
False
```

- 9.symmetric_difference()
- 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 - The set.symmetric_difference(set)
  - set - Required. The set to check for matches in
- Example
  - Return a set that contains all items from both sets, except items that are present in both sets

In [27]:

```
x = {"apple", "banana", "cherry"}
y = {"google", "microsoft", "apple"}

z = x.symmetric_difference(y)

print(z)
```

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

- 10.symmetric_difference_update()
- The symmetric_difference_update() method updates the original set by removing items that are present in both sets, and inserting the other items
- Syntax - The set.symmetric_difference_update(set)
  - set - Required. The set to check for matches in
- Example
  - Remove the items that are present in both sets, AND insert the items that is not present in both sets

In [28]:

```
x = {"apple", "banana", "cherry"}
y = {"google", "microsoft", "apple"}

x.symmetric_difference_update(y)

print(x)
```

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