A Python set is a well-defined collection of distinct objects, called elements or members.

Sets are unordered. Set elements are unique–if you try to add an element to a set a second time, it has no effect.

A set itself may be modified, but the elements contained in a set must be immutable (actually hashable, but for the types you are familiar with so far, immutable and hashable are effectively the same thing).

Sets are unordered collections of unique objects, there are two types of set:

1. Sets - They are mutable and new elements can be added once sets are defined

```
basket = {'apple', 'orange', 'apple', 'pear', 'orange', 'banana'}
print(basket) # duplicates will be removed
```

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

```
a = set('abracadabra')
print(a)
```

```
{'c', 'r', 'b', 'd', 'a'}
```

2. Frozen Sets - They are immutable and new elements cannot added after its defined.

```
b = frozenset('asdfagsa')
print(b)
```

```
frozenset({'g', 'f', 'd', 's', 'a'})
```

```
cities = frozenset(["Frankfurt", "Basel","Freiburg"])
print(cities)
```

```
frozenset({'Frankfurt', 'Freiburg', 'Basel'})
```

Operations on sets

```
# with other sets
# Intersection
{1, 2, 3, 4, 5}.intersection({3, 4, 5, 6})
```

```
{3, 4, 5}
```

```
{1, 2, 3, 4, 5} & {3, 4, 5, 6}
```

```
{3, 4, 5}
```

```
# Union
{1, 2, 3, 4, 5}.union({3, 4, 5, 6})
```

```
{1, 2, 3, 4, 5, 6}
```

```
 # {1, 2, 3, 4, 5, 6}
{1, 2, 3, 4, 5} | {3, 4, 5, 6}
```

```
{1, 2, 3, 4, 5, 6}
```

```
# Difference
{1, 2, 3, 4}.difference({2, 3, 5})
```

```
{1, 4}
```

```
 # {1, 4}
{1, 2, 3, 4} - {2, 3, 5}
```

```
{1, 4}
```

```
# Symmetric difference with
{1, 2, 3, 4}.symmetric_difference({2, 3, 5})
```

```
{1, 4, 5}
```

```
{1, 2, 3, 4} ^ {2, 3, 5}
```

```
{1, 4, 5}
```

```
# Superset check
{1, 2}.issuperset({1, 2, 3})
```

```
False
```

```
{1, 2} >= {1, 2, 3}
```

```
False
```

```
# Subset check
{1, 2}.issubset({1, 2, 3})
```

```
True
```

```
{1, 2} <= {1, 2, 3}
```

```
True
```

```
{1}.ispropersubset({1,2,3})
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-54-83ee6cb7daea> in <cell line: 1>()
----> 1 {1}.ispropersubset({1,2,3})

AttributeError: 'set' object has no attribute 'ispropersubset'
```

EXPLAIN ERROR

```
set()<{1,2,3}
```

```
True
```

```
{1}<{1,2,3}
```

```
True
```

```
# Disjoint check
{1, 2}.isdisjoint({3, 4})
```

```
True
```

```
{1, 2}.isdisjoint({1, 4}) # False
```

```
False
```

with single elements

```
# Existence check
2 in {1,2,3}
```

```
True
```

```
4 in {1,2,3}
```

```
False
```

```
4 not in {1,2,3}
```

```
True
```

```
# Add and Remove
s = {1,2,3}
s.add(4)
s
```

```
{1, 2, 3, 4}
```

```
s.discard(3)
s
```

```
{1, 2, 4}
```

```
s.discard(5)
s
```

```
{1, 2, 4}
```

```
s = {1,2,3}
s.remove(2)
s
```

```
{1, 3}
```

```
s.remove(2)
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-30-805863e1e466> in <cell line: 1>()
----> 1 s.remove(2)

KeyError: 2
```

EXPLAIN ERROR

Set operations return new sets, but have the corresponding in-place versions:

method	in-place operation	in-place method
union	s |= t	update
intersection	s &= t	intersection_update
difference	s -= t	difference_update
symmetric_difference	s ^= t	symmetric_difference_update

```
#For example:
s = {1, 2}
s.update({3, 4})
s
```

```
{1, 2, 3, 4}
```

Get the unique elements of a list

Let's say you've got a list of restaurants -- maybe you read it from a file. You care about the unique restaurants in the list. The best way to get the unique elements from a list is to turn it into a set:

```
restaurants = ["McDonald's", "Burger King", "McDonald's", "Chicken Chicken"]
unique_restaurants = set(restaurants)
print(unique_restaurants)
```

```
{'Chicken Chicken', 'Burger King', "McDonald's"}
```

```
restaurants = ["McDonald's", "Burger King", "Burger king","McDonald's", "Chicken Chicken"]
restaurants.remove("McDonald's")
restaurants
```

```
['Burger King', 'Burger king', "McDonald's", 'Chicken Chicken']
```

Note that the set is not in the same order as the original list; that is because sets are unordered, just like dicts.

```
# Removes all duplicates and returns another list
list(set(restaurants))
```

```
['Chicken Chicken', 'Burger King', "McDonald's"]
```

Set of Sets

```
{{1,2}, {3,4}}
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-35-c5fc32a1a2cf> in <cell line: 1>()
----> 1 {{1,2}, {3,4}}

TypeError: unhashable type: 'set'
```

EXPLAIN ERROR

Instead, use frozenset:

```
{frozenset({1, 2}), frozenset({3, 4})}
```

```
{frozenset({3, 4}), frozenset({1, 2})}
```

Set Operations using Methods and Builtins

```
#We define two sets a and b
a = {1, 2, 2, 3, 4}
b = {3, 3, 4, 4, 5} #NOTE: {1} creates a set of one element, but {} creates an empty dict. The correct way to create an
#empty set is set().
```

```
#Intersection
#a.intersection(b) returns a new set with elements present in both a and b
a.intersection(b)
```

```
{3, 4}
```

```
#Union
#a.union(b) returns a new set with elements present in either a and b
```

```
a.union(b)
```

```
{1, 2, 3, 4, 5}
```

```
#Difference
#a.difference(b) returns a new set with elements present in a but not in b
a.difference(b)
```

```
{1, 2}
```

```
b.difference(a)
```

```
{5}
```

```
#Symmetric Difference
#a.symmetric_difference(b) returns a new set with elements present in either a or b but not in both
a.symmetric_difference(b)
```

```
{1, 2, 5}
```

```
b.symmetric_difference(a)

#NOTE: a.symmetric_difference(b) == b.symmetric_difference(a)
```

```
{1, 2, 5}
```

```
#Subset and superset
#c.issubset(a) tests whether each element of c is in a.
#a.issuperset(c) tests whether each element of c is in a.
c = {1, 2}
c.issubset(a)
```

```
True
```

```
a.issuperset(c)
```

```
True
```

TESTING THE LEARNING SO FAR:

1. **Basic Set Operations:**
   - What is a set in Python?
   - Explain the difference between a set and a list in Python.
   - How do you create an empty set in Python?
   - Describe the basic set operations: union, intersection, and difference.
2. **Set Methods:**
   - Name three common methods available for sets in Python.
   - How do you add an element to a set?
   - Explain the purpose of the `remove()` and `discard()` methods in sets.
3. **Set Operations using Operators:**
   - What operators are used for set union and intersection in Python?
   - Write code to perform the union of two sets.
   - How do you check if one set is a subset of another set?
4. **Immutability and Sets:**
   - Can you have a set of sets in Python?
   - Explain why sets are mutable but elements of sets must be immutable.
   - Provide an example of an immutable data type that can be used as a set element.
5. **Frozen Sets:**

- What is a frozen set in Python?
- How does a frozen set differ from a regular set?
- In what scenarios might you choose to use a frozen set?

6. **Removing Duplicates using Sets:**
   - How can you use a set to remove duplicates from a list?
   - Write a short code snippet to demonstrate removing duplicates from a list using a set.
7. **Real-world Applications:**
   - Provide an example of a real-world scenario where using a set in Python would be beneficial.
   - How could sets be used in data analysis or processing?

Exercise 1: Lists

1. Create a list of integers from 1 to 5.
2. Append the number 6 to the list.
3. Extend the list with another list: [7, 8, 9].
4. Access and print the third element of the list.
5. Slice the list to get elements from index 2 to 4.
6. Reverse the list.
7. Remove the element 7 from the list.
8. Check if the element 5 is present in the list.
9. Find the index of the element 8 in the list.
10. Create a new list that contains only the even numbers from the original list.

Exercise 2: Tuples

1. Create a tuple with three different data types: int, float, and string.
2. Access and print the second element of the tuple.
3. Concatenate the tuple with another tuple containing three elements.
4. Unpack the tuple into three variables: a, b, and c.
5. Check if the string "apple" is present in the tuple.
6. Create a new tuple by repeating the original tuple three times.
7. Find the index of the float element in the tuple.
8. Convert the tuple to a list.
9. Create a tuple of your favorite colors and concatenate it with the original tuple.
10. Use a tuple to swap the values of two variables.

Exercise 3: Sets

1. Create a set with the first five prime numbers.
2. Add the number 7 to the set.
3. Remove the number 3 from the set.
4. Check if the set is a subset of {2, 3, 5, 7, 11}.
5. Find the union of the set with {7, 11, 13}.
6. Create a frozen set from the original set.
7. Check if the set has any common elements with {1, 4, 9}.
8. Remove all elements from the set.
9. Create a set of your favorite fruits and find the intersection with { 'apple', 'banana', 'orange' }.
10. Use set comprehension to create a set of squares for numbers 1 to 10.