

TECHNOLOGY



Python

Lists, Dictionary, Sets, and Tuples



Learning Objectives

By the end of this lesson, you will be able to:

- 🕒 Understand Lists and its methods
- 🕒 Understand Dictionaries and Sets
- 🕒 Classify Mutable and Immutable objects
- 🕒 Understand tuple() constructor



Understanding Lists

Understanding Lists

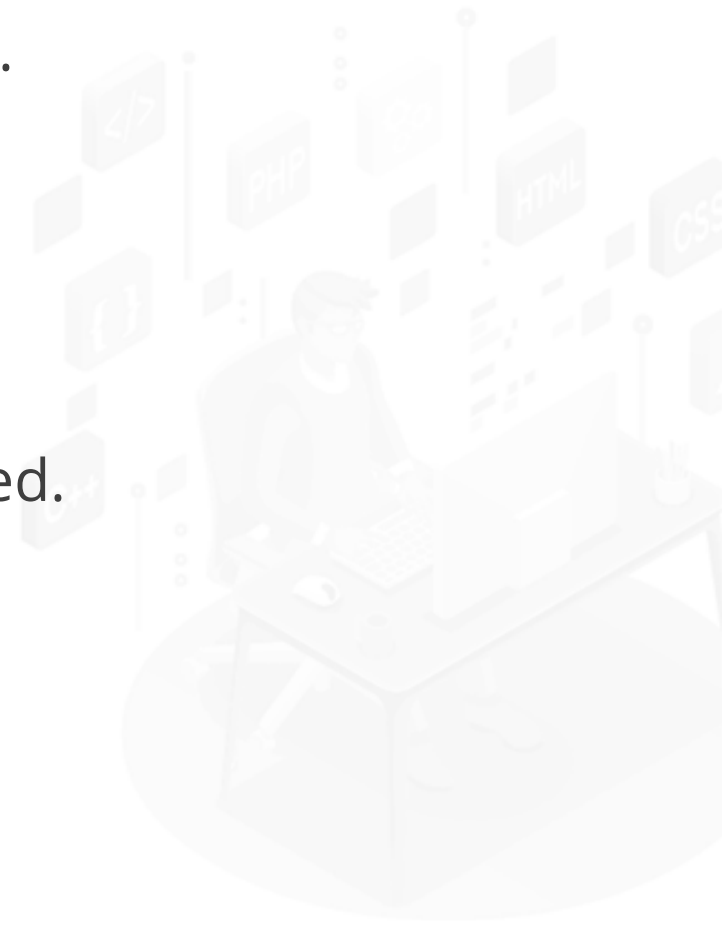
- A list is a collection of items that are ordered and mutable.
- Lists are one of the most used data structures in Python, as they can hold any type of data, including integers, floats, strings, and other lists.

Here are some key features of lists:

1. **Ordered:** The items in a list are stored in a specific order, which means you can access them using an index. The first item in a list has an index of 0, the second item has an index of 1, and so on.

Understanding Lists

2. **Mutable:** You can add, remove, or modify items in a list after it has been created.
3. **Heterogeneous:** A list can hold items of different types.
4. **Dynamic:** The size of a list can change dynamically as items are added or removed.



Understanding Lists

Here are some examples of working with lists in Python:

Creating a list

```
fruits = ['apple', 'banana', 'orange']
```

Accessing items in a list

```
print(fruits[0]) # Output: 'apple'
```

```
print(fruits[1]) # Output: 'banana'
```

```
print(fruits[2]) # Output: 'orange'
```



Understanding Lists

Here are some examples of working with lists in Python:

Adding an item to a list

```
fruits.append('grape')  
print(fruits)  # Output: ['apple', 'banana', 'orange', 'grape']
```

Removing an item from a list

```
fruits.remove('banana')  
print(fruits)  # Output: ['apple', 'orange', 'grape']
```

Modifying an item in a list

```
fruits[0] = 'kiwi'  
print(fruits)  # Output: ['kiwi', 'orange', 'grape']
```



Methods of Lists

Methods of Lists

Python provides several methods to work with lists:

1. **append():** This method is used to add an element to the end of the list

```
my_list = [1, 2, 3]
my_list.append(4)
print(my_list)  # Output: [1, 2, 3, 4]
```

2. **insert():** This method is used to insert an element at a specific position in the list.

```
my_list = [1, 2, 3]
my_list.insert(1, 5)
print(my_list)  # Output: [1, 5, 2, 3]
```



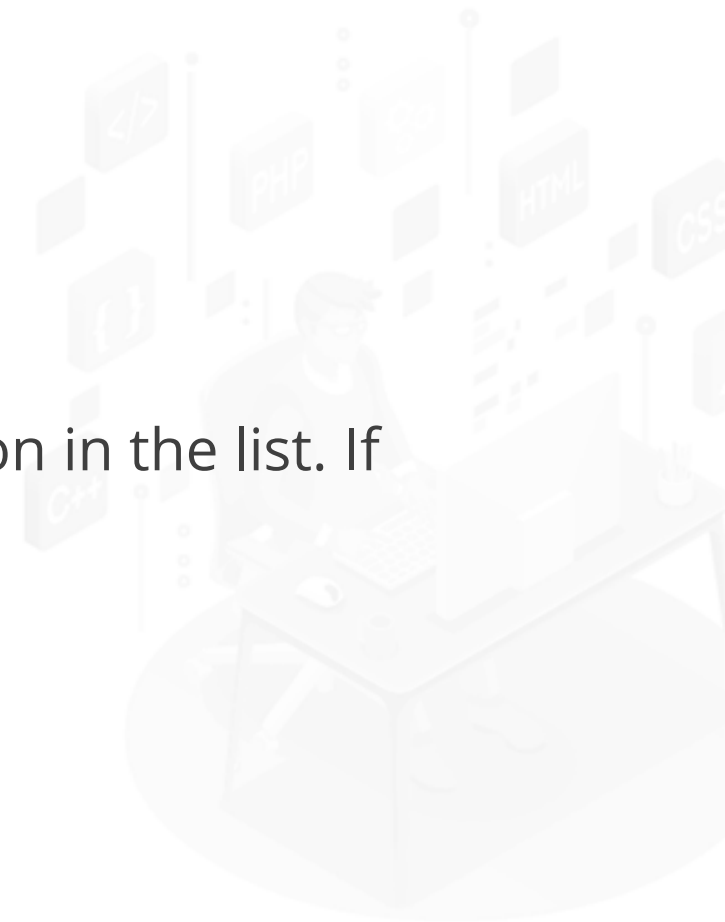
Methods of Lists

3. **remove():** This method is used to remove the first occurrence of a specified element from the list

```
my_list = [1, 2, 3, 2]
my_list.remove(2)
print(my_list)  # Output: [1, 3, 2]
```

4. **pop():** This method is used to remove and return the element at a specific position in the list. If no position is specified, it removes and returns the last element.

```
my_list = [1, 2, 3]
last_element = my_list.pop()
print(last_element)  # Output: 3
print(my_list)  # Output: [1, 2]
```



Methods of Lists

5. **sort()**: This method is used to sort the elements of the list in ascending order.

```
my_list = [3, 1, 2]
my_list.sort()
print(my_list)  # Output: [1, 2, 3]
```

6. **reverse()**: This method is used to reverse the order of the elements in the list.

```
my_list = [1, 2, 3]
my_list.reverse()
print(my_list)  # Output: [3, 2, 1]
```



Understanding Dictionaries

Understanding Dictionaries

- A dictionary is a collection of key-value pairs, where each key is associated with a value. It is also known as an associative array, hash table, or simply a map.
- To create a dictionary in Python, you can use curly braces {} or the **dict()** constructor function.

Here is an example:

```
# Creating a dictionary using {}  
my_dict = {"apple": 2, "banana": 3, "orange": 1}  
  
# Creating a dictionary using dict()  
my_dict = dict(apple=2, banana=3, orange=1)
```



Understanding Dictionaries

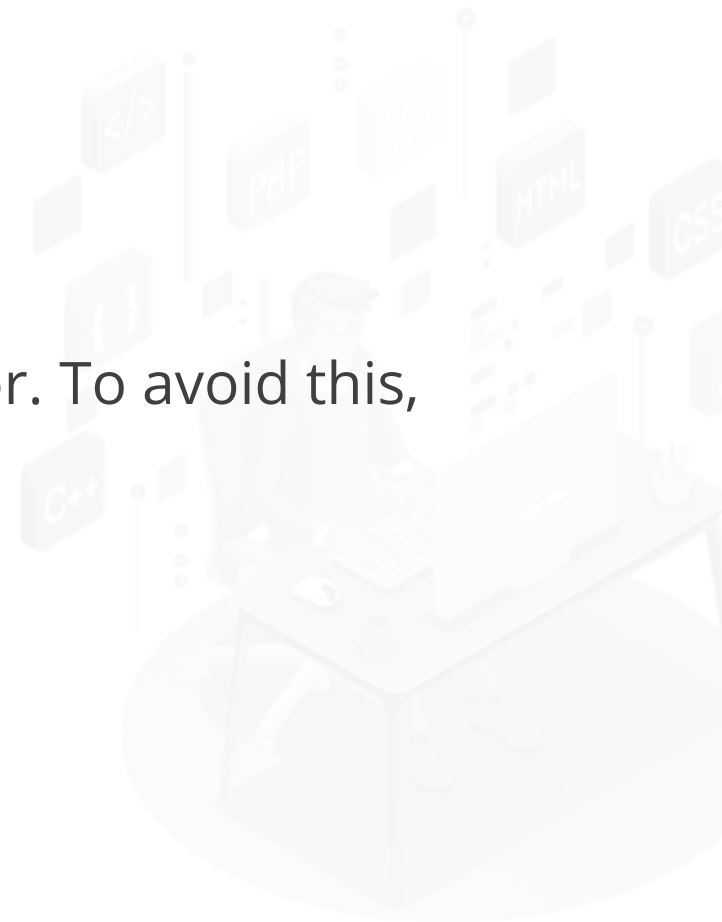
- To access the values of a dictionary, you can use the keys as the index.

Here is an example

```
# Accessing the value of a dictionary  
print(my_dict["apple"])  # Output: 2
```

- If you try to access a key that doesn't exist in the dictionary, you'll get a `KeyError`. To avoid this, you can use the **`get()`** method, which returns `None` if the key doesn't exist:

```
# Using the get() method to avoid KeyError  
print(my_dict.get("grape"))  # Output: None
```



Understanding Dictionaries

You can also iterate over the keys or values of a dictionary using the **keys()** and **values()** methods:

Iterating over the keys of a dictionary

```
for key in my_dict.keys():  
    print(key)
```

Iterating over the values of a dictionary

```
for value in my_dict.values():
```



Methods of Dictionaries

Methods of Dictionaries

Here are some of the most used methods for dictionaries in Python:

1. `clear()`: This method is used to remove all the elements from a dictionary.

```
my_dict = {'a': 1, 'b': 2, 'c': 3}
```

```
my_dict.clear()
```

```
print(my_dict)
```

Output: {}

2. `copy()`: This method is used to create a shallow copy of a dictionary.

```
my_dict = {'a': 1, 'b': 2, 'c': 3}
```

```
new_dict = my_dict.copy()
```

```
print(new_dict) # Output: {'a': 1, 'b': 2, 'c': 3}
```



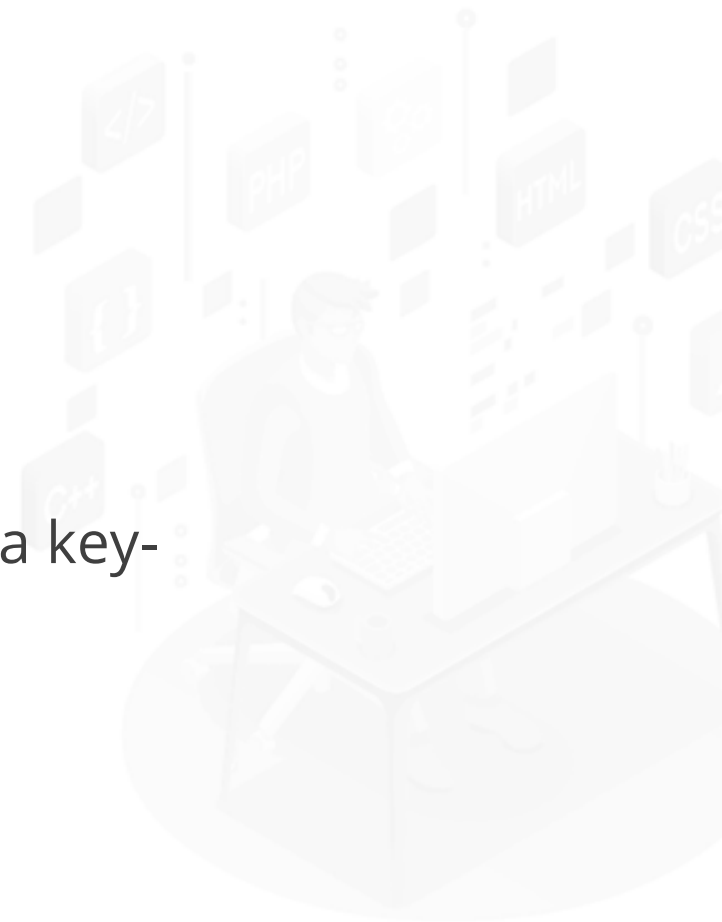
Methods of Dictionaries

3. **get()**: This method is used to get the value of a key in a dictionary. If the key does not exist, it returns a default value.

```
my_dict = {'a': 1, 'b': 2, 'c': 3}
print(my_dict.get('a')) # Output: 1
print(my_dict.get('d', 'Not Found'))
# Output: Not Found
```

4. **items()**: This method is used to return a list of tuples, where each tuple contains a key-value pair.

```
my_dict = {'a': 1, 'b': 2, 'c': 3}
print(my_dict.items())
# Output: dict_items([('a', 1), ('b', 2), ('c', 3)])
```



Methods of Dictionaries

5. **keys()**: This method is used to return a list of keys in a dictionary.

```
my_dict = {'a': 1, 'b': 2, 'c': 3}
print(my_dict.keys()) # Output: dict_keys(['a', 'b', 'c'])
```

6. **values()**: This method is used to return a list of values in a dictionary.

```
my_dict = {'a': 1, 'b': 2, 'c': 3}
print(my_dict.values()) # Output: dict_values([1, 2, 3])
```



Understanding Sets and Its Methods

Understanding Sets and Its Methods

- In Python, a set is an unordered collection of unique elements.
- Sets are mutable, which means you can add or remove elements from a set.

Here are some of the most used methods for sets in Python:

add(): This method is used to add an element to a set.

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

```
my_set.add(4)
```

```
print(my_set) # Output: {1, 2, 3, 4}
```



Understanding Sets and Its Methods

2. **update()**: This method is used to add multiple elements to a set.

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

```
my_set.update([3, 4, 5])
```

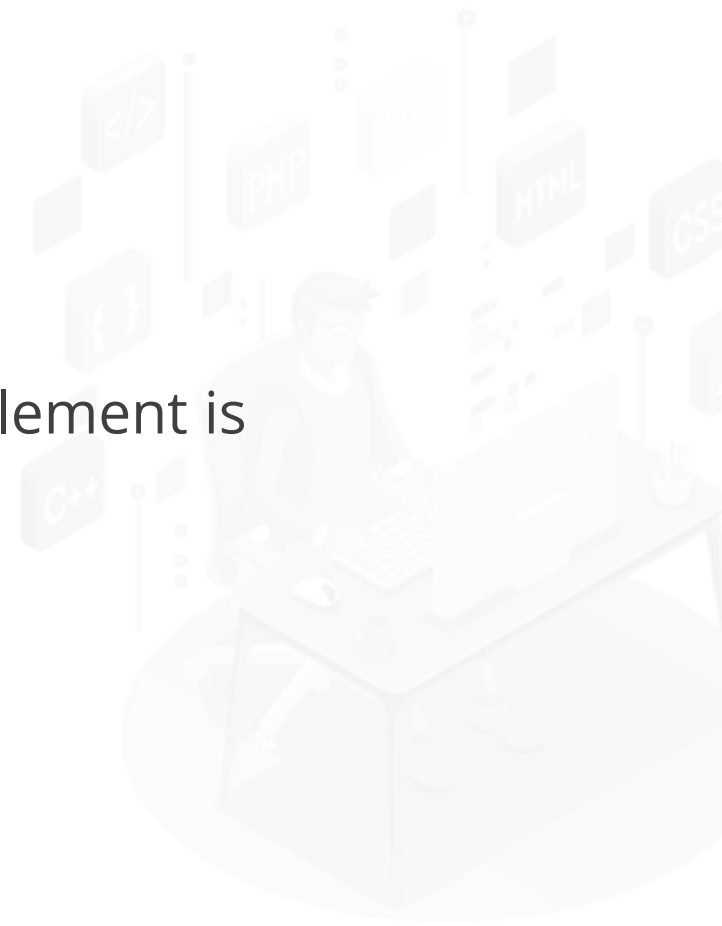
```
print(my_set) # Output: {1, 2, 3, 4, 5}
```

3. **remove()**: This method is used to remove a specified element from a set. If the element is not present in the set, it raises a KeyError.

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

```
my_set.remove(2)
```

```
print(my_set) # Output: {1, 3}
```



Understanding Sets and Its Methods

4. **discard():** This method is used to remove a specified element from a set. If the element is not present in the set, it does nothing.

```
My_set = {1, 2, 3}
My_set.discard(2)
print(my_set) # Output: {1, 3}
```

5. **pop():** This method is used to remove and return an arbitrary element from a set. If the set is empty, it raises a KeyError.

```
my_set = {1, 2, 3}
element = my_set.pop()
print(element) # Output: 1
print(my_set) # Output: {2, 3}
```



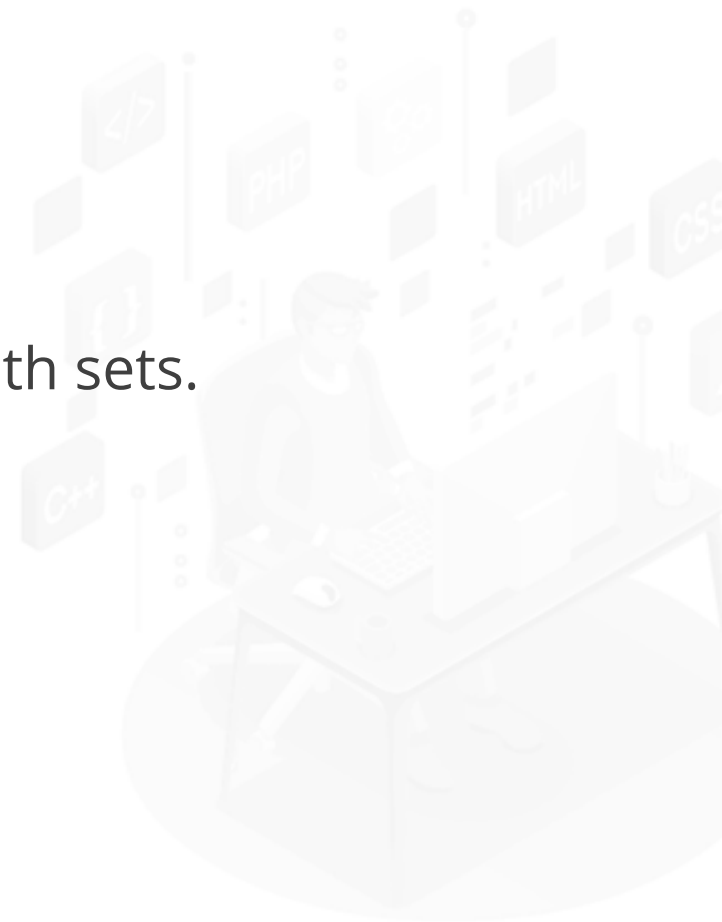
Understanding Sets and Its Methods

6. **clear():** This method is used to remove all elements from a set.

```
my_set = {1, 2, 3}
my_set.clear()
print(my_set) # Output: set()
```

7. **union():** This method is used to return a new set containing all elements from both sets.

```
my_set1 = {1, 2, 3}
my_set2 = {3, 4, 5}
new_set = my_set1.union(my_set2)
print(new_set) # Output: {1, 2, 3, 4, 5}
```



Understanding Tuples

Understanding Tuples

Tuple is an ordered, immutable collection of elements. Tuples are similar to lists, but once a tuple is created, its elements cannot be changed.

Here are some examples of tuples and how they can be used in Python:

1. Creating a tuple:

```
my_tuple = (1, 2, 3)
```

2. Accessing elements in a tuple:

```
my_tuple = (1, 2, 3)
print(my_tuple[0]) # Output: 1
```



Understanding Tuples

3. Unpacking a tuple:

```
my_tuple = (1, 2, 3)
a, b, c = my_tuple
print(a, b, c) # Output: 1 2 3
```

4. Creating a tuple with a single element:

```
my_tuple = (1,)
```

5. Concatenating tuples:

```
my_tuple1 = (1, 2)
my_tuple2 = (3, 4)
new_tuple = my_tuple1 + my_tuple2
print(new_tuple) # Output: (1, 2, 3, 4)
```



Understanding Tuples

6. Multiplying tuples:

```
my_tuple = (1, 2)
new_tuple = my_tuple * 3
print(new_tuple) # Output: (1, 2, 1, 2, 1, 2)
```

7. Checking if an element is in a tuple:

```
my_tuple = (1, 2, 3)
print(2 in my_tuple) # Output: True
```

8. Finding the length of a tuple:

```
my_tuple = (1, 2, 3)
print(len(my_tuple)) # Output: 3
```



Mutable or Immutable Objects

Mutable Objects

- Mutable objects are those that can be changed after they are created. If you modify a mutable object, the object itself is changed.
- Examples of mutable objects in Python include lists, sets, and dictionaries.

```
my_list = [1, 2, 3]
```

```
new_list = my_list
```

```
my_list.append(4)
```

```
print(my_list) # Output: [1, 2, 3, 4] (the original list is modified)
```

```
print(new_list) # Output: [1, 2, 3, 4] (new_list is still pointing to the modified object)
```



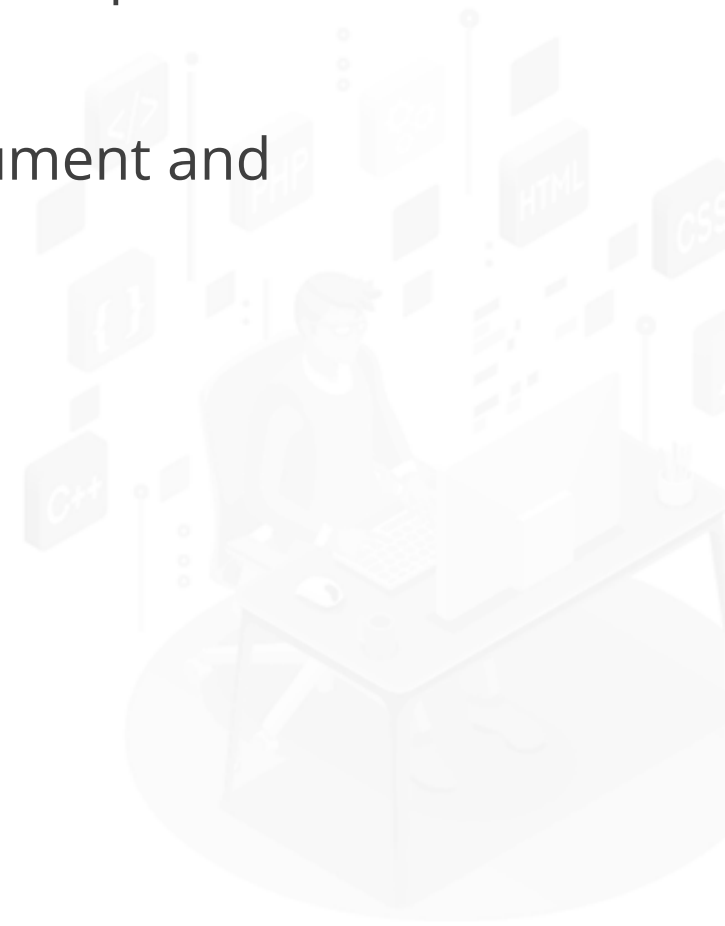
tuple() Constructor

tuple() Constructor

- In Python, the tuple() constructor is a built-in function that can be used to create a tuple object.
- The constructor takes an iterable object (such as a list, tuple, or string) as its argument and returns a new tuple with the same elements.

Here's an example of using the tuple() constructor to create a tuple from a list:

```
my_list = [1, 2, 3]
my_tuple = tuple(my_list)
print(my_tuple) # Output: (1, 2, 3)
```



Key Takeaways

- Dictionary is a collection of key-value pairs, where each key is associated with a value.
- List is an ordered collection of items, and it is a mutable data type, which means you can change its content.
- Set is an unordered collection of unique elements.
- Tuple is an ordered, immutable collection of elements.

