Data Structures in Python: Complex Data Types- PART 1

John Karuitha, PhD

February 27, 2025

0.0.1 Dictionaries and Sets in Python

---

1 1. Dictionaries

A **dictionary** in Python is a collection of key-value pairs. Each key in a dictionary is unique, and each key maps to a value. Dictionaries are mutable, which means their content can be changed after creation.

1.1 Creating Dictionaries

You can create a dictionary using curly braces `{}` or the `dict()` constructor.

1.2 Example:

```
# Creating a dictionary
student = {
    "name": "Alice",
    "age": 21,
    "major": "Computer Science"
}

# Using the dict() constructor
student_2 = dict(name="Bob", age=22, major="Mathematics")
```

1.3 Accessing Values

Values are accessed using their corresponding keys.

```
print(student["name"])  # Output: Alice
print(student_2["major"])  # Output: Mathematics
```

```
Alice
Mathematics
```

---

2 Dictionary Methods

Method	Description	Example
`dict.get(key, default)`	Returns the value for the specified key. If the key is not found, returns the default value.	`student.get("age", "Not Found")` → `21`
`dict.keys()`	Returns a view object containing all keys in the dictionary.	`student.keys()` → `dict_keys(['name', 'age', 'major'])`
`dict.values()`	Returns a view object containing all values in the dictionary.	`student.values()` → `dict_values(['Alice', 21, 'Computer Science'])`

Method	Description	Example
`dict.items()`	Returns a view object containing key-value pairs as tuples.	`student.items()` → `[('name', 'Alice'), ('age', 21)]`
`dict.update()`	Updates the dictionary with the key-value pairs from another dictionary or iterable.	`student.update({"age": 22})`
`dict.pop(key)`	Removes the key-value pair with the specified key and returns the value. If key is not found, raises an error.	`student.pop("age")` → 21
`dict.clear()`	Removes all items from the dictionary.	`student.clear()` → `{}`
`dict.copy()`	Returns a shallow copy of the dictionary.	`student.copy()` → Copy of `student`

2.1 Examples:

```
# Using get() to safely access keys
print(student.get("name"))  # Output: Alice
print(student.get("GPA", "Not Available"))  # Output: Not Available

# Iterating over keys and values
for key, value in student.items():
    print(f"{key}: {value}")

# Updating a dictionary
student.update({"GPA": 3.8})
print(student)  # Output: {'name': 'Alice', 'age': 21, 'major': 'Computer Science', 'GPA': 3
```

```
Alice
Not Available
name: Alice
age: 21
major: Computer Science
{'name': 'Alice', 'age': 21, 'major': 'Computer Science', 'GPA': 3.8}
```

3 2. Sets

A **set** in Python is an unordered collection of unique elements. Sets are useful when you need to store items without duplicates and perform mathematical set operations like union, intersection, and difference. Sets are mutable, but their elements must be immutable (e.g., strings, numbers, tuples).

3.1 Creating Sets

Sets can be created using curly braces `{}` or the `set()` constructor.

Example:

```python
# Creating a set
fruits = {"apple", "banana", "cherry"}

# Using the set() constructor
numbers = set([1, 2, 3, 4, 5])
```

3.2 Accessing Elements

Since sets are unordered, they don't support indexing or slicing. You can iterate over them or check for membership.

```python
print("apple" in fruits)  # Output: True
print("orange" in fruits)  # Output: False
```

```
True
False
```

---

4 Set Methods

Method	Description	Example
`set.add(element)`	Adds an element to the set. If the element already exists, it won't be added again.	`fruits.add("orange")` → `{"apple", "banana", "cherry", "orange"}`
`set.remove(element)`	Removes the specified element. Raises an error if the element doesn't exist.	`fruits.remove("banana")`
`set.discard(element)`	Removes the specified element. Does not raise an error if the element doesn't exist.	`fruits.discard("kiwi")`
`set.pop()`	Removes and returns an arbitrary element from the set.	`fruits.pop()` → Removes a random item
`set.clear()`	Removes all elements from the set.	`fruits.clear()` → `set()`
`set.union()`	Returns a set containing all elements from both sets.	`fruits.union({"grape", "apple"})`
`set.intersection()`	Returns a set containing elements common to both sets.	`fruits.intersection({"apple", "grape"})`
`set.difference()`	Returns a set containing elements in the first set but not in the second.	`fruits.difference({"apple"})`
`set.symmetric_difference()`	Returns a set containing elements not common to both sets.	`fruits.symmetric_difference({"apple` `"grape"})`

4.1 Examples:

```
# Adding and removing elements
fruits.add("kiwi")
print(fruits)  # Output: {'apple', 'banana', 'cherry', 'kiwi'}

fruits.remove("banana")
print(fruits)  # Output: {'apple', 'cherry', 'kiwi'}

# Union and intersection
set_a = {1, 2, 3}
set_b = {3, 4, 5}
print(set_a.union(set_b))  # Output: {1, 2, 3, 4, 5}
print(set_a.intersection(set_b))  # Output: {3}

# Set difference
print(set_a.difference(set_b))  # Output: {1, 2}
```

```
{'kiwi', 'cherry', 'banana', 'apple'}
{'kiwi', 'cherry', 'apple'}
{1, 2, 3, 4, 5}
{3}
{1, 2}
```

---

5 Key Differences Between Dictionaries and Sets

Feature	Dictionary	Set
Structure	Key-value pairs	Unique elements
Access	Values accessed via keys	No direct access by index
Duplicate Items	Keys must be unique; values can duplicate	All elements must be unique
Example	`{"name": "Alice", "age": 21}`	`{"apple", "banana", "cherry"}`

---

6 Summary

- **Dictionaries** store data as key-value pairs and are great for associating unique keys with values.
- **Sets** store unique, unordered items and are useful for mathematical operations like union, intersection, and difference.
- Both structures provide powerful tools for handling collections of data efficiently in Python.

References