DATA STRUCTURES -II

Sets

A **set** is an **unordered** and **mutable** collection of **unique**, **immutable** items. This means:

- **Unordered**: The items have no defined order or index.
- **Mutable**: You can add or remove items from the set.
- **Unique**: Sets automatically discard any duplicate values.
- **Immutable Items**: The items within a set must be of an immutable type (like numbers, strings, or tuples). You cannot have a list inside a set.

Creating and Working with Sets

Sets are created with curly braces `{}`. To create an empty set, you must use the `set()` constructor, as `{}` creates an empty dictionary.

```python
# Creating sets
my_set = {1, 2, "hello", 3.14}
numbers = {1, 2, 3, 2, 1} # Duplicates are automatically removed
print(numbers) # Output: {1, 2, 3}

# Creating an empty set
empty_set = set()

# Checking for an item's existence
print("hello" in my_set) # Output: True
```

Key Set Methods

- **Adding Items**
  - `.add(item)`: Adds a single element to the set.
  - `.update(iterable)`: Adds all elements from an iterable (like a list or another set).

  ```python
  numbers.add(4)
  numbers.update([5, 6])
  print(numbers) # Output: {1, 2, 3, 4, 5, 6}
  ```
- **Removing Items**
  - `.remove(item)`: Removes a specified item. Raises a `KeyError` if the item is not found.
  - `.discard(item)`: Removes a specified item, but does nothing if the item is not found.
  - `.pop()`: Removes and returns an arbitrary item from the set.

  ```python
  numbers.discard(1)
  numbers.remove(2)
  print(numbers) # Output: {3, 4, 5, 6}
  ```

- Set Operations (Joining Sets)

  These are powerful methods for comparing and combining sets.

- **Union (|)**: Combines two sets, returning a new set with all unique items from both.
- **Intersection (&)**: Returns a new set with only the items that are present in both sets.
- **Difference (-)**: Returns a new set with items from the first set that are not in the second set.
- **Symmetric Difference (^)**: Returns a new set with items that are in either set, but not in both.

```python
set_a = {1, 2, 3, 4}
set_b = {3, 4, 5, 6}

print(set_a | set_b) # Union: {1, 2, 3, 4, 5, 6}
print(set_a & set_b) # Intersection: {3, 4}
print(set_a - set_b) # Difference: {1, 2}
print(set_a ^ set_b) # Symmetric Difference: {1, 2, 5, 6}
```

Dictionaries

A **dictionary** is an **unordered** and **mutable** collection that stores data in **key-value pairs**. This means:

- **Keys must be unique and immutable** (e.g., strings, numbers, or tuples). The last value for a duplicate key will be the one that is kept.
- **Values** can be of any data type and can be duplicated.
- Items are accessed by their **key**, not by an index.

Creating and Working with Dictionaries

```python
# Creating a dictionary
person = {
    "name": "John",
    "age": 30,
    "city": "New York"
}

# Accessing values by key
print(person["name"]) # Output: John

# A safer way to access values is with the .get() method
# It returns None (or a default value) if the key doesn't exist, avoiding errors.
print(person.get("country", "Unknown")) # Output: Unknown
```

Key Dictionary Methods

- **Adding and Updating Items**
  - You can add or update an item by assigning a value to a key.
  - `.update(other_dict)`: Merges the key-value pairs from another dictionary into the current one.

```python
person["age"] = 31 # Updates the value for the 'age' key
person["gender"] = "Male" # Adds a new key-value pair
print(person) # Output: {'name': 'John', 'age': 31, 'city': 'New York', 'gender': 'Male'}
```

- **Removing Items**

    - .pop(key): Removes the item with the specified key and returns its value.
    - .popitem(): Removes and returns the last inserted key-value pair as a tuple.
    - del my_dict[key]: Deletes the item with the specified key.

```
Python
# Removes the 'city' key and its value
city = person.pop("city")
print(city) # Output: New York
```

- **Accessing Keys, Values, and Items**
    - .keys(): Returns a view of all the keys in the dictionary.
    - .values(): Returns a view of all the values.
    - .items(): Returns a view of all key-value pairs as tuples.

```
Python
print(person.keys())   # Output: dict_keys(['name', 'age', 'gender'])
print(person.values()) # Output: dict_values(['John', 31, 'Male'])
print(person.items())  # Output: dict_items([('name', 'John'), ('age', 31), ('gender', 'Male')])
```

Data Structures: A Quick Comparison

Feature	Lists	Tuples	Sets	Dictionaries
Mutability	**Mutable** (Changeable)	**Immutable** (Unchangeable)	**Mutable** (Changeable)	**Mutable** (Changeable)
Ordering	**Ordered**	**Ordered**	**Unordered**	**Unordered** (Ordered in Python 3.7+)
Duplicates	**Allowed**	**Allowed**	**Not Allowed**	Keys are **Unique**
Syntax	[1, 2, 3]	(1, 2, 3)	{1, 2, 3}	{'a': 1, 'b': 2}
Access	By **Index** my_list[0]	By **Index** my_tuple[0]	No Indexing	By **Key** my_dict['a']
Use Case	Ordered collections that need to be modified.	Ordered data that should not be changed.	Storing unique items and performing mathematical set operations.	Storing data as key-value pairs for quick lookups.