**Complex Data Types in Python: Lists, Tuples, Dictionaries, and Sets**

Python has several built-in complex data types that help in organizing and storing multiple values efficiently. The most commonly used ones are:

1. **Lists**
2. **Tuples**
3. **Dictionaries**
4. **Sets**

Let’s explore each of these in detail.

**1. Lists**

A **list** is an ordered collection of elements that can be of different data types. Lists are **mutable**, meaning their elements can be modified after creation.

**Characteristics of Lists:**

* Ordered (Elements have a specific order)
* Mutable (Can be changed after creation)
* Allow duplicate values
* Can store elements of different data types
* Indexed (Elements can be accessed using an index)

**Syntax:**

my\_list = [10, "Hello", 3.14, True, [1, 2, 3]]

**Operations on Lists:**

# Accessing elements

print(my\_list[1]) # Output: "Hello"

# Modifying elements

my\_list[1] = "Python"

print(my\_list) # Output: [10, "Python", 3.14, True, [1, 2, 3]]

# Adding elements

my\_list.append(42) # Adds 42 to the end

my\_list.insert(2, "New") # Inserts "New" at index 2

# Removing elements

my\_list.remove(10) # Removes first occurrence of 10

my\_list.pop(2) # Removes element at index 2

# Iterating over a list

for item in my\_list:

print(item)

**2. Tuples**

A **tuple** is similar to a list, but it is **immutable**, meaning it cannot be modified after creation.

**Characteristics of Tuples:**

* Ordered
* Immutable (Cannot be changed after creation)
* Allow duplicate values
* Can store different data types
* Indexed

**Syntax:**

my\_tuple = (10, "Hello", 3.14, True)

**Operations on Tuples:**

# Accessing elements

print(my\_tuple[1]) # Output: "Hello"

# Concatenating tuples

new\_tuple = my\_tuple + (20, "Python")

print(new\_tuple) # Output: (10, "Hello", 3.14, True, 20, "Python")

# Unpacking tuples

a, b, c, d = my\_tuple

print(a, b, c, d) # Output: 10 Hello 3.14 True

# Iterating over a tuple

for item in my\_tuple:

print(item)

**Note**: Since tuples are immutable, you **cannot** modify elements after creation.

**3. Dictionaries**

A **dictionary** is a collection of key-value pairs, where each key maps to a specific value. It is **unordered** and **mutable**.

**Characteristics of Dictionaries:**

* Unordered (No fixed order)
* Mutable (Can be modified after creation)
* Keys must be unique
* Values can be of any data type

**Syntax:**

my\_dict = {

"name": "Alice",

"age": 25,

"city": "New York"

}

**Operations on Dictionaries:**

# Accessing values

print(my\_dict["name"]) # Output: Alice

# Modifying values

my\_dict["age"] = 26

# Adding new key-value pairs

my\_dict["profession"] = "Engineer"

# Removing key-value pairs

del my\_dict["city"]

# Checking if a key exists

if "age" in my\_dict:

print("Age exists!")

# Iterating over a dictionary

for key, value in my\_dict.items():

print(f"{key}: {value}")

**4. Sets**

A **set** is an unordered collection of unique elements. Sets are useful for storing **distinct values** and performing mathematical operations like union and intersection.

**Characteristics of Sets:**

* Unordered (No specific order)
* Mutable (Can add or remove elements)
* No duplicate values
* Does not support indexing (Since it's unordered)

**Syntax:**

my\_set = {1, 2, 3, 4, 5}

**Operations on Sets:**

# Adding elements

my\_set.add(6)

# Removing elements

my\_set.remove(3) # Removes 3 (Raises an error if not found)

my\_set.discard(7) # Removes 7 (Does NOT raise an error if not found)

# Checking if an element exists

print(2 in my\_set) # Output: True

# Set operations

A = {1, 2, 3, 4}

B = {3, 4, 5, 6}

# Union (All elements from both sets)

print(A | B) # Output: {1, 2, 3, 4, 5, 6}

# Intersection (Common elements)

print(A & B) # Output: {3, 4}

# Difference (Elements in A but not in B)

print(A - B) # Output: {1, 2}

**Comparison of Lists, Tuples, Dictionaries, and Sets**

| **Feature** | **List** | **Tuple** | **Dictionary** | **Set** |
| --- | --- | --- | --- | --- |
| Ordered | ✅ | ✅ | ❌ | ❌ |
| Mutable | ✅ | ❌ | ✅ | ✅ |
| Allows Duplicates | ✅ | ✅ | ❌ (Keys) | ❌ |
| Indexed | ✅ | ✅ | ❌ (Keys instead) | ❌ |
| Best Used For | Ordered collections with modifications | Fixed collections | Key-value pairs | Unique items & set operations |

**Conclusion**

* **Use a list** when you need an ordered, mutable sequence.
* **Use a tuple** when you need an ordered, immutable sequence.
* **Use a dictionary** when you need key-value pairs for fast lookups.
* **Use a set** when you need to store unique elements and perform set operations.