Prepared by *Coderistic*

Lists vs Tuples vs Sets vs Dictionaries

Here are some **real-world use cases** for each data structure in Python:

1️⃣ List (`list`) – Storing Ordered Data

✅ **Use Case:** Storing and processing a list of students in a class.

```
students = ["Alice", "Bob", "Charlie", "David"]
students.append("Eve")  # Adding a new student
print(students[0])  # Accessing the first student
```

🔹 **Why?** Lists allow adding, removing, and modifying elements easily while maintaining order.

2️⃣ Tuple (`tuple`) – Immutable Data Storage

✅ **Use Case:** Storing **GPS coordinates** (latitude, longitude) that shouldn't be modified.

```
location = (13.4125, 103.8667)  # Angkor Wat coordinates
print(f"Latitude: {location[0]}, Longitude: {location[1]}")
```

🔹 **Why?** Tuples prevent accidental changes, making them ideal for read-only data.

3️⃣ Set (`set`) – Unique Elements & Fast Lookups

✅ **Use Case:** Removing duplicate emails from a mailing list.

```
emails = {"user1@gmail.com", "user2@gmail.com", "user1@gmail.com"}
print(emails)  # Output: {'user1@gmail.com', 'user2@gmail.com'}
```

🔹 **Why?** Sets automatically remove duplicates and allow fast membership checks (`"email" in emails`).

4️⃣ Dictionary (`dict`) – Key-Value Mapping

✅ **Use Case:** Storing **product prices** in an e-commerce app.

```
products = {
    "iPhone": 999.99,
    "MacBook": 1299.99,
    "AirPods": 199.99
}
print(products["MacBook"])  # Output: 1299.99
```

◇ **Why?** Dictionaries allow **fast lookups** and meaningful key-value mappings.

TL;DR – Choosing the Right Data Structure

✔ **List:** Ordered, flexible storage (e.g., student names, ordered items).
✔ **Tuple:** Immutable, fixed data (e.g., GPS coordinates, database records).
✔ **Set:** Unique, unordered collection (e.g., unique emails, unique usernames).
✔ **Dictionary:** Fast key-value lookups (e.g., product prices, user profiles).