**✅ Defining List and List Slicing & Use of Tuple Data Type**

**🔷 1. Lists in Python**

A **list** is a built-in **sequence data type** in Python that can hold **multiple values in a single variable**. These values can be of any type: integers, strings, floats, or even other lists. Lists are **ordered** and **mutable**, meaning their content can be changed after creation.

**✅ Defining Lists**

Lists are defined using square brackets [ ].

# Examples

numbers = [1, 2, 3, 4, 5]

fruits = ["apple", "banana", "cherry"]

mixed = [1, "hello", 3.14, True]

nested = [1, [2, 3], 4]

**✅ Accessing Elements**

Elements are accessed using **index numbers**, starting from 0.

fruits = ["apple", "banana", "cherry"]

print(fruits[0]) # apple

print(fruits[-1]) # cherry (last item)

**🔷 2. List Slicing**

**Slicing** means extracting a specific portion from the list.

**🔹 Syntax:**

list[start:stop:step]

* start: Index to begin slicing (inclusive)
* stop: Index to end slicing (exclusive)
* step: Interval (optional)

**🔹 Examples:**

nums = [0, 1, 2, 3, 4, 5]

print(nums[1:4]) # [1, 2, 3]

print(nums[:3]) # [0, 1, 2]

print(nums[3:]) # [3, 4, 5]

print(nums[::-1]) # [5, 4, 3, 2, 1, 0] (reverse list)

**🔷 3. List Operations**

Python provides powerful list manipulation tools:

| **Operation** | **Description** |
| --- | --- |
| append(x) | Adds x at the end |
| insert(i, x) | Adds x at index i |
| remove(x) | Removes first occurrence of x |
| pop(i) | Removes item at index i |
| sort() | Sorts the list |
| reverse() | Reverses the list |
| len(list) | Returns number of items |
| in / not in | Check membership |

**🔹 Example:**

colors = ["red", "blue", "green"]

colors.append("yellow") # ['red', 'blue', 'green', 'yellow']

colors.remove("blue") # ['red', 'green', 'yellow']

**🔷 4. Tuple in Python**

A **tuple** is another built-in sequence type similar to a list, but **immutable** — meaning **once created, you cannot change its content**.

**✅ Defining Tuples**

Tuples are defined using **parentheses ( )**:

coordinates = (10, 20)

person = ("Alice", 25, "Engineer")

If defining a tuple with one item, add a comma:

single = (5,) # Correct

not\_tuple = (5) # This is just an integer

**✅ Accessing Tuple Elements**

Just like lists, tuples use indexing and slicing.

person = ("Alice", 25, "Engineer")

print(person[0]) # Alice

print(person[-1]) # Engineer

print(person[0:2]) # ('Alice', 25)

**🔷 5. Tuple vs List (Key Differences)**

| **Feature** | **List ([])** | **Tuple (())** |
| --- | --- | --- |
| Mutability | Mutable (changeable) | Immutable (fixed) |
| Performance | Slower | Faster |
| Methods | Many built-in methods | Fewer methods |
| Use Case | When data changes | When data is fixed |

**✅ Why Use Tuples?**

* To protect data from being accidentally changed.
* Faster execution in loops or functions.
* Can be used as keys in dictionaries (lists cannot).

**🔷 6. Tuple Packing and Unpacking**

**🔹 Packing:**

Combining multiple values into a single tuple.

info = ("Alice", 25, "Engineer")

**🔹 Unpacking:**

Extracting values from a tuple into separate variables.

name, age, job = info

print(name) # Alice

print(age) # 25

**🔷 7. Nested Tuples and Lists**

Tuples and lists can contain other tuples or lists:

nested\_list = [1, [2, 3], [4, 5]]

nested\_tuple = (1, (2, 3), (4, 5))

Access like this:

print(nested\_list[1][0]) # 2

print(nested\_tuple[2][1]) # 5

**✅ Summary**

* **List**: Ordered, mutable collection. Use [ ]. Powerful for dynamic data handling.
* **Tuple**: Ordered, immutable collection. Use ( ). Safer and faster for fixed data.
* **Slicing** lets you extract parts from lists or tuples.
* Choose **list** when data changes, **tuple** when data is constant.