**OOPS (Object-Oriented Programming System)**

OOPS is a way of programming where we organize code into "objects." Think of objects as things in real life (like a car or a person) that have:

1. **Attributes (properties):** Things the object has. Example: A car has a color, brand, and speed.
2. **Methods (actions):** Things the object can do. Example: A car can drive, stop, or honk.

**Main Concepts in OOPS:**

1. **Class**: A blueprint for creating objects.
   * Example: A blueprint of a car (not the car itself).
2. **Object**: A specific instance of a class.
   * Example: Your red Honda car (based on the car blueprint).

**Encapsulation**

Encapsulation is like a protective shield around the object. It bundles data (attributes) and methods (functions) into a single unit and restricts outside access to sensitive information.

* + To hide internal details.
  + To protect data from being modified accidentally.

**Inheritance**

Inheritance allows one class (child) to reuse the properties and methods of another class (parent). It's like a parent passing down their traits to a child.

* + Code reuse.
  + Simplifies complex code by breaking it into hierarchies.

**Polymorphism**

Polymorphism means "many forms." Allowing different classes to use the same method in different ways.

* + To simplify code when working with different object types.
  + Makes it easier to extend the system in the future.

**Abstraction**

Abstraction means showing only the essential details and hiding unnecessary information. Think of driving a car—We don’t need to know how the engine works, just how to use the controls.

* + Simplifies code by focusing on what an object **does** rather than **how** it does it.
  + Reduces complexity for users.

**List**

A list is like a shopping list. It’s a collection of items where:

* Items are ordered.
* You can add, remove, or change items.
* It allows duplicates.

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

print(fruits[0]) # Output: apple

# Adding an item

fruits.append("orange")

print(fruits) # Output: ['apple', 'banana', 'cherry', 'orange']

# Changing an item

fruits[1] = "kiwi"

print(fruits) # Output: ['apple', 'kiwi', 'cherry', 'orange']

# Removing an item

fruits.remove("cherry")

print(fruits) # Output: ['apple', 'kiwi', 'orange']

**Tuple**

* A tuple is like a list, but you **cannot change** it. Once created, it’s fixed.

coordinates = (10, 20, 30)

print(coordinates[1]) # Output: 20

# You cannot change a tuple

# coordinates[1] = 50 # This will give an error

**Set**

A set is like a bag of items where:

* The order doesn’t matter.
* Duplicates are not allowed.

**Dictionary**

A dictionary is like a real dictionary. It stores data as **key-value pairs**, where:

* Keys are like words (unique).
* Values are like meanings (can be anything).

numbers = {1, 2, 3, 3, 4}

print(numbers) # Output: {1, 2, 3, 4} (duplicates removed)

# Adding an item

numbers.add(5)

print(numbers) # Output: {1, 2, 3, 4, 5}

# Removing an item

numbers.remove(3)

print(numbers) # Output: {1, 2, 4, 5}

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Feature** | **List** | **Tuple** | **Set** | **Dictionary** |
| **Ordered** | Yes | Yes | No | No |
| **Mutable** | Yes | No | No | Yes |
| **Duplicates** | |  | | --- | |  |  |  | | --- | | Allowed | | Allowed | Not Allowed | Keys: Not Allowed, Values: Allowed |
| **Use Case** | General-purpose collections | Fixed collections | Unique items | Key-value pairs |
| **Index** | Numeric indices (0, 1, 2...) -> fruits[0] | Numeric indices (0, 1, 2...) ->fruits[0] | Not supported | Keys (e.g., "name", "age") ->  Stud[“name’] |