6. Class and Object (OOP Concepts)

❖ Understanding the concepts of classes, objects, attributes, and methods in Python.

1. Class :

A class is a blueprint or template for creating objects. It defines what attributes (data) and methods (behaviors) each object will have.

Syntax:

```
class Person:
    species = "Homo sapiens"  # class
attribute
```

2. Object (Instance) :

An **object** is an **instance** of a class—a concrete entity built from its blueprint. You create objects by "calling" the class.

Syntax:

```
p = Person()
```

This is like baking cookies from a cookie-cutter: each cookie is an individual object, but all follow the same shape.

3. Attributes :

Attributes are variables stored in objects or classes:

• Class Attributes

Shared across all instances. Defined directly in the class body:

Syntax:

```
class Dog:
    species = "Canine"  # class attribute
```

- **Instance Attributes :**

  Specific to each object. Usually defined in __init__() using self:

Syntax:

```
class Dog:
    def __init__(self, name, age):
        self.name = name    # instance attribute
        self.age = age      # instance attribute
dog1 = Dog("Buddy", 3)
dog2 = Dog("Charlie", 5)
```

Each dog has its own name and age, but they share the collective species.

4. Methods

Methods are **functions** defined inside a class that operate on objects:

Syntax:

```
class Dog:

    def __init__(self, name):

        self.name = name


    def bark(self):

        print(f"{self.name} barks!")
```

Calling dog.bark() invokes the method. Internally, Python translates dog.bark() to Dog.bark(dog)—the instance (dog) is passed as the first argument (self)

❖Difference between local and global variables.

Local Variables :

- Declared **inside** a function.
- **Scope**: only that function (or nested inner functions using nonlocal).
- **Lifetime**: exists only while the function runs; destroyed after it returns .
- A local variable **shadows** a global of the same name — the global remains unaffected .

Global Variables

- Declared **outside** any function (module-level).
- **Accessible** from any part of the module, including inside functions (unless shadowed).
- **Lifetime** spans the entire execution of the program (until it ends).
- To **modify** a global variable inside a function, you must use the global keyword, e.g.