# Object-Oriented Programming (OOP)

OOP is a programming paradigm based on objects and classes.

FOUR PILLARS OF OOP:

1. ENCAPSULATION
   - Bundling data and methods together
   - Data hiding using access modifiers
   - Provides abstraction and security

   Example (Python):
```
class BankAccount:
    def __init__(self):
        self.__balance = 0  # Private variable
    def deposit(self, amount):
        self.__balance += amount
    def get_balance(self):
        return self.__balance
```
2. INHERITANCE
   - Acquiring properties from parent class
   - Promotes code reusability
   - Types: Single, Multiple, Multilevel, Hierarchical, Hybrid

   Example (Python):
```
class Animal:
    def speak(self):
        pass
class Dog(Animal):
    def speak(self):
        return "Woof!"
class Cat(Animal):
    def speak(self):
        return "Meow!"
```
3. POLYMORPHISM
   - Same interface, different implementations
   - Types:
     a) Compile-time (Method Overloading)
     b) Runtime (Method Overriding)

   Example (Python):
```
class Shape:
    def area(self):
        pass
class Circle(Shape):
    def __init__(self, radius):
        self.radius = radius
    def area(self):
        return 3.14 * self.radius ** 2
  class Rectangle(Shape):
    def __init__(self, length, width):
        self.length = length
        self.width = width
    def area(self):
        return self.length * self.width
```

4. ABSTRACTION
   - Hiding implementation details
   - Showing only essential features
   - Abstract classes and interfaces

   Example (Python):

```
  from abc import ABC, abstractmethod
  class Vehicle(ABC):
    @abstractmethod
    def start(self):
        pass
  class Car(Vehicle):
    def start(self):
        return "Car started"
```

CLASSES AND OBJECTS:

Class: Blueprint/template

Object: Instance of a class

Example:

```
class Student:
    def __init__(self, name, roll):
        self.name = name
        self.roll = roll
    def display(self):
        print(f"Name: {self.name}, Roll: {self.roll}")
# Creating objects
s1 = Student("John", 101)
s2 = Student("Alice", 102)
```

CONSTRUCTOR AND DESTRUCTOR:

Constructor: Special method called when object is created

- Initializes object
- __init__() in Python, ClassName() in Java/C++

Destructor: Called when object is destroyed

- Cleanup operations
- __del__() in Python, ~ClassName() in C++

ACCESS MODIFIERS:

1. Public: Accessible everywhere
2. Private: Accessible only within class
3. Protected: Accessible in class and subclasses

Python:

- Public: self.var
- Protected: self._var
- Private: self.__var

STATIC MEMBERS:

- Shared by all objects
- Accessed using class name
- Example: Math.PI, Counter.count

DESIGN PATTERNS:

1. Singleton: Only one instance
2. Factory: Object creation logic
3. Observer: Event handling
4. Strategy: Interchangeable algorithms
5. Decorator: Add functionality dynamically

BENEFITS OF OOP:

? Modularity: Organized code

? Reusability: Inheritance, composition

? Flexibility: Polymorphism

? Maintainability: Easy to update

? Security: Encapsulation