1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 1 1 0 1

</ Table of contents

1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 0 1 1 1 1 1 0 1

Why Reuse Code?

1. Save time (don't write the same thing again and again)

2. Avoid mistakes (if you copy-paste, you might forget to update one copy)

3. Make changes easier (change in one place updates everywhere)

- **The OOP Way Reusability = Maintainability**

- **Lego Block Analogy**

- **Key Points to Remember**
  - Don't Repeat Yourself (DRY): Write code once and reuse it
  - Classes are templates: They define how objects should work
  - Inheritance helps reuse: Child classes get all features of parent classes
  - Changes are easier: Fix or improve in one place, all users benefit
- **Reusability makes your code**
  - Shorter
  - Easier to maintain
  - Less prone to errors
  - More flexible for future changes

1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 0 1 1 1 1 1 0 1

Concept of Inheritance

- Allows a child class to reuse properties and methods from a parent class, making code shorter and easier to maintain.

- **Why to use inheritance?**
  - Avoid rewriting the same code (reusability)
  - Organize code better (hierarchy)
  - Easier to modify (change parent → affects all children)

- **Basic syntax:**
  - A child class inherits from a parent class using (ParentClass).

```
class Parent:
    pass
class Child(Parent):
    pass
```

```
class Animal

class Dog(Animal)
```

- **IS-A relationship:** Inheritance follows an "IS-A" relationship:
  - A Dog is a type of Animal
  - A Car is a type of Vehicle

1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 0 1 1 1 1 1 0 1

Types of Inheritance

1. Single Inheritance:
   - Child inherits from one parent only.
     e.x. Car inherits from Vehicle

2. Multilevel Inheritance:
   - child inherits from a parent, which itself inherits from another class.
     e.x. C inherits from B, and B inherits from A

3. Multiple Inheritance:
   - A child class can inherit from multiple parents.
     e.x. Smartphone inherits from Phone and Camera

- **Accessing Parent Methods & Attributes:**
  - A child can use all methods from the parent

- **Using `super()` to Call Parent Methods**
  - If you want to **extend** a parent method,

1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 0 1 1 1 1 1 0 1

Method Overriding

- Inheritance allows a child class to reuse parent class methods. But sometimes, we want to:
  - **Change** how a method works in the child (**Overriding**)
  - **Extend** the parent method instead of replacing it (**super()**)
- **What is Method Overriding?**
  - When a child class defines the same method as the parent, it overrides the parent's version.
- **Why Override Methods?**
  - To customize behavior for the child class.
  - Example: All animals make sounds, but dogs bark, cats meow, etc.
- **Using super()**
  - Don’t fully replace the parent method—just add more to it.
  - super() lets us call the parent’s method inside the child.
- **Constructor Inheritance:**
  - If the parent has an __init__, the child must call it to inherit properties.

1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 0 1 1 1 1 1 0 1

Multilevel inheritance

- **Why Use It?**
  - Reuse code step-by-step.
  - Organize classes logically

- **Multilevel inheritance is like a family tree in programming:**
  - A grandparent class passes features to a parent class.
  - The parent class then passes those features (plus its own) to a child class.

- **How It Works?**
  1. Class A (Grandparent) → Class B (Parent) → Class C (Child).
  2. Class C gets all methods from B and A.

- **Example: Family of Vehicles**
  - Think of vehicles:
  - Vehicle (Grandparent) → Car (Parent) → Electric Car (Child)

- Chain of inheritance: A → B → C.
- ✔ Child class (C) gets everything from B and A.
- ✔ Avoid too many levels (2-3 is okay; more gets messy).

1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 0 1 1 1 1 1 0 1

Polymorphism

- Polymorphism means "many forms".

1. Method Polymorphism:
   - Different classes can have the same method name, but each does something unique.
   - Why to use?
     - can loop through different objects and call the same method name.
     - Each object automatically uses its own version.
2. Duck Typing in Python:

   - Python doesn’t care about the class type, only if the method exists.
   - If an object has make_sound(), Python will run it—even if the classes are unrelated.

3. Polymorphism in Built-in Python Functions:
   - Python uses polymorphism everywhere.

1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 0 1 1 1 1 1 0 1

Lab: Vehicle Management System

1. Base Class (Vehicle):
   - Properties: name, speed (set in __init__).
   - Methods:
     1. start_engine(): Returns "Engine started".
     2. describe(): Returns f"{name} moves at {speed} km/h".
2. Single Inheritance (Car inherits Vehicle):
   - New property: brand.
   - Override describe():
   - Use super() to extend the parent’s method.
   - Returns f"{name} moves at {speed} km/h (Brand: {brand})".
3. Multilevel Inheritance (Electric Car inherits Car)
   - New property: battery_range.
     Override start_engine(): Returns "Electric motor activated!".
4. Polymorphism (Duck Typing)
   - Unrelated Bicycle class:
     - Same methods (start_engine(), describe()).
     - Implements its own behavior.

1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 0 1 1 1 1 1 0 1