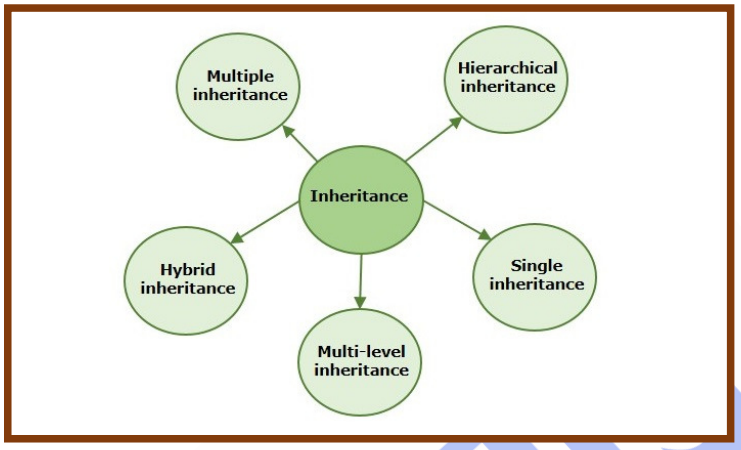
Class Inheritance in Python

Inheritance in Python is an object-oriented programming (OOP) concept that allows a class (called a child or subclass) to inherit the attributes and methods from another class (called a parent or base class).



This promotes code reuse, organization, and hierarchical relationships among classes.

Why Use Inheritance?

Imagine you have a class Person with basic details like name and age.
Then you want to create a class Student that also has a name and age — but with some extra details (like student_id).
Instead of rewriting all the same code again, the Student class can **inherit** from Person.

Basic Syntax of Inheritance

```
class Parent:
    # parent class code

class Child(Parent):
    # child class code
```

Here,
- Parent → base class
- Child → derived class (inherits from Parent)

Class Inheritance in Python

Simple Example of Inheritance

```
# Parent class
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def display(self):
        print(f"Name: {self.name}, Age: {self.age}")

# Child class
class Student(Person):
    def __init__(self, name, age, student_id):
        # Call the parent class constructor
        super().__init__(name, age)
        self.student_id = student_id

    def show(self):
        print(f"Student ID: {self.student_id}")

# Create object of child class
s1 = Student("Aisha", 20, "S101")

# Access parent and child methods
s1.display()  # from Person class
s1.show()     # from Student class
```

Output:

```
Name: Aisha, Age: 20
Student ID: S101
```

Step-by-Step Explanation:

1. **Person** is the **parent class** — it has name, age, and a method display().
2. **Student** is the **child class** — it inherits from Person.
3. Inside Student, we use:
4. super().__init__(name, age)
to call the parent's constructor and initialize those attributes.
5. Then we add a new attribute student_id (unique to Student).
6. Student now has **both** parent and child features.

So, Student inherits display() and all variables from Person.

Class Inheritance in Python

Using super() Function

super() is used to **call the parent class's methods** (usually the constructor).
It helps us avoid writing the parent's name directly and works even if the class hierarchy changes.

Example:
super().__init__(name, age)
calls Person.__init__(self, name, age) internally.

Types of Inheritance in Python:

Type	Description	Example
Single Inheritance	One child inherits from one parent	class B(A)
Multiple Inheritance	One child inherits from multiple parents	class C(A, B)
Multilevel Inheritance	A child inherits from a parent, and another class inherits from that child	A → B → C
Hierarchical Inheritance	Multiple children inherit from one parent	A → B, C
Hybrid Inheritance	Combination of multiple inheritance types	(complex mix)

Type 1: Single Inheritance

A child class inherits from only one parent class. This is the most common type.
Single Inheritance

```
class Animal:
    def sound(self):
        print("Animals make sounds")

class Dog(Animal):
    def bark(self):
        print("Dog barks")

d = Dog()
d.sound()  # from parent
d.bark()   # from child
```

Explanation:
- **Animal** → parent class with method sound().
- **Dog** → child class that inherits from Animal.
- Dog has its own method bark() and **inherits** sound() from Animal.
- When we create d = Dog(), d can use both sound() and bark().

Class Inheritance in Python

Output:
Animals make sounds
Dog barks

Type 2: Multiple Inheritance

A child class inherits from multiple parent classes.
Python supports multiple inheritance (unlike some languages such as Java).

```
class Mother:
    def cooking(self):
        print("Mother can cook")

class Father:
    def driving(self):
        print("Father can drive")

class Child(Mother, Father):
    def play(self):
        print("Child can play")

obj = Child()
obj.cooking()
obj.driving()
obj.play()
```

Explanation:
Child inherits from two parent classes: Mother and Father.
Child has its own method play().
obj can access all three methods: cooking() (from Mother), driving() (from Father), and play() (own method).

Output:
Mother can cook
Father can drive
Child can play

Type 3: Multilevel Inheritance

A class inherits from a class that is already derived from another class.

Code:

```
class Grandfather:
    def property(self):
```

Class Inheritance in Python

```
    print("Has family property")

class Father(Grandfather):
    def car(self):
        print("Has a car")

class Son(Father):
    def bike(self):
        print("Has a bike")

obj = Son()
obj.property()
obj.car()
obj.bike()
```

Explanation:
- Grandfather → top-level parent class with property().
- Father → inherits from Grandfather, adds car().
- Son → inherits from Father, adds bike().
- Son indirectly inherits everything from both Father and Grandfather.
- obj can access property(), car(), and bike().

Output:
```
Has family property
Has a car
Has a bike
```

Type 4: Hierarchical Inheritance
Multiple child classes inherit from the same parent class.

```
class Parent:
    def work(self):
        print("Parent works hard")

class Son(Parent):
    def study(self):
        print("Son studies")

class Daughter(Parent):
    def dance(self):
        print("Daughter dances")

s = Son()
d = Daughter()
```

```
s.work()
d.work()
```

Explanation:

- **One parent (Parent)** → two children (Son and Daughter).
- Both children **inherit the work() method** from Parent.
- Each child has its own unique method (study() for Son, dance() for Daughter).
- s can use work() from parent + study() if called.
- d can use work() from parent + dance() if called.

Output:

```
Parent works hard
Parent works hard
```

Type 5: Hybrid Inheritance

A combination of two or more types of inheritance.

Summary Table of Types

Type	Example	Key Feature
Single	Dog(Animal)	One child inherits one parent
Multilevel	Son(Father(Grandfather))	Chain of inheritance
Multiple	Child(Mother, Father)	One child inherits from multiple parents
Hierarchical	Son(Parent), Daughter(Parent)	One parent, multiple children

Why do we use Inheritance?

- **Code Reusability** - reuse methods and attributes from parent classes.
- **Better Code Organization** - structure code in a logical hierarchy (general → specific).
- **Avoid Code Duplication** - write common code once in the parent class.
- **Easy Maintenance** - update code in one place; changes reflect in all child classes.
- **Method Overriding** - customize or extend parent class behavior in child classes.
- **Extensibility** - easily add new features or classes without affecting existing code.

Class Inheritance in Python

Solved Example: Employees and Managers

```
class Employee:
    def __init__(self, name, salary):
        self.name = name
        self.salary = salary

    def show(self):
        print(f"Employee Name: {self.name}, Salary: {self.salary}")

# Manager inherits Employee
class Manager(Employee):
    def __init__(self, name, salary, department):
        super().__init__(name, salary)
        self.department = department

    def display(self):
        print(f"Manager of Department: {self.department}")

m1 = Manager("John", 80000, "IT")
m1.show()      # from Employee
m1.display()   # from Manager
```

Brief explanation of your code:

Classes:

- **Employee** → Parent/Base class
  - Constructor __init__ initializes name and salary.
  - Method show() prints employee details.
- **Manager** → Child/Derived class
  - Inherits from Employee.
  - Constructor __init__ calls super().__init__(name, salary) to initialize parent attributes.
  - Adds its own attribute department.
  - Method display() prints manager-specific info.

Object Creation:

m1 = Manager("John", 80000, "IT")

- Creates a Manager object with name "John", salary 80000, and department "IT".
- **Parent constructor** runs via super() to set name and salary.
- **Child constructor** sets the department.

Method Calls:

```
m1.show()      # inherited from Employee
```

Class Inheritance in Python

m1.display() # defined in Manager

- m1.show() → prints: Employee Name: John, Salary: 80000
- m1.display() → prints: Manager of Department: IT