**Inheritance in Python Programming**

**Inheritance** is a fundamental concept in object-oriented programming (OOP) that allows a class (child class) to inherit attributes and methods from another class (parent class). This promotes code reuse, organization, and extensibility.

**1. Types of Inheritance in Python**

Python supports different types of inheritance:

**(a) Single Inheritance**

A child class inherits from a single parent class.

# Parent Class

class Animal:

def speak(self):

print("This animal makes a sound")

# Child Class inheriting from Animal

class Dog(Animal):

def bark(self):

print("Dog barks")

# Creating an object of Dog class

dog = Dog()

dog.speak() # Inherited method

dog.bark() # Method of Dog class

**(b) Multiple Inheritance**

A child class inherits from more than one parent class.

# Parent Class 1

class Father:

def skill(self):

print("Father is good at painting")

# Parent Class 2

class Mother:

def skill(self):

print("Mother is good at singing")

# Child Class inheriting from both parents

class Child(Father, Mother):

pass

child = Child()

child.skill() # Calls the method from the first inherited class (Father)

**Note:** Python follows the **Method Resolution Order (MRO)** to determine which method to call in case of conflicts.

**(c) Multilevel Inheritance**

A chain of inheritance where a class is derived from another derived class.

class Grandparent:

def grandparent\_info(self):

print("I am the grandparent")

class Parent(Grandparent):

def parent\_info(self):

print("I am the parent")

class Child(Parent):

def child\_info(self):

print("I am the child")

child = Child()

child.grandparent\_info() # Inherited from Grandparent

child.parent\_info() # Inherited from Parent

child.child\_info() # Own method

**(d) Hierarchical Inheritance**

Multiple child classes inherit from the same parent class.

class Vehicle:

def vehicle\_info(self):

print("This is a vehicle")

class Car(Vehicle):

def car\_info(self):

print("This is a car")

class Bike(Vehicle):

def bike\_info(self):

print("This is a bike")

car = Car()

bike = Bike()

car.vehicle\_info()

car.car\_info()

bike.vehicle\_info()

bike.bike\_info()

**(e) Hybrid Inheritance**

A combination of multiple types of inheritance.

class A:

def method\_A(self):

print("Class A method")

class B(A):

def method\_B(self):

print("Class B method")

class C(A):

def method\_C(self):

print("Class C method")

class D(B, C): # Multiple and multilevel inheritance

def method\_D(self):

print("Class D method")

d = D()

d.method\_A()

d.method\_B()

d.method\_C()

d.method\_D()

**2. Method Overriding in Inheritance**

A child class can override a method of the parent class to provide a new implementation.

class Parent:

def show(self):

print("Parent class method")

class Child(Parent):

def show(self):

print("Child class method") # Overriding method

child = Child()

child.show() # Calls the overridden method

**3. Using super() in Inheritance**

The super() function allows access to the parent class’s methods inside the child class.

class Parent:

def show(self):

print("This is Parent class")

class Child(Parent):

def show(self):

super().show() # Calling parent class method

print("This is Child class")

child = Child()

child.show()

**Output:**

This is Parent class

This is Child class

**4. Private and Protected Members in Inheritance**

* **Protected Members (\_variable)**: Can be accessed in child classes.
* **Private Members (\_\_variable)**: Cannot be directly accessed in child classes.

class Parent:

def \_\_init\_\_(self):

self.\_protected = "Protected Variable"

self.\_\_private = "Private Variable"

class Child(Parent):

def display(self):

print(self.\_protected) # Accessible

# print(self.\_\_private) # Not Accessible

child = Child()

child.display()

**5. Abstract Classes and Inheritance**

Using **abstract classes** to enforce method implementation in child classes.

from abc import ABC, abstractmethod

class Animal(ABC):

@abstractmethod

def make\_sound(self):

pass # Must be implemented in subclasses

class Dog(Animal):

def make\_sound(self):

print("Dog barks")

dog = Dog()

dog.make\_sound()

**Key Takeaways**

1. **Inheritance** allows code reuse and better structure.
2. **Different types** of inheritance include single, multiple, multilevel, hierarchical, and hybrid.
3. **Method overriding** lets child classes modify inherited behavior.
4. **super()** helps call parent methods in child classes.
5. **Abstract classes** enforce implementation of essential methods.