Inheritance in Python: Simple, Complex and All Inheritance Types

# 1. Introduction

Inheritance allows a class (child class) to acquire the properties and behaviors (methods) of another class (parent class). It promotes code reusability and establishes relationships between different classes. Python supports various types of inheritance: Single, Multiple, Multilevel, Hierarchical, and Hybrid.

# 2. Simple Example: Single Inheritance

A child class inherits from one parent class.

class Animal:  
 def sound(self):  
 return "Some sound"  
  
class Dog(Animal):  
 def sound(self):  
 return "Bark"  
  
d = Dog()  
print(d.sound())

# 3. Multilevel Inheritance

A class is derived from a child class which is already derived from a base class.

class Animal:  
 def category(self):  
 return "Mammal"  
  
class Dog(Animal):  
 def breed(self):  
 return "Labrador"  
  
class Puppy(Dog):  
 def age(self):  
 return "2 months"  
  
p = Puppy()  
print(p.category(), p.breed(), p.age())

# 4. Multiple Inheritance

A class inherits from more than one parent class.

class Father:  
 def skills(self):  
 return "Gardening"  
  
class Mother:  
 def skills(self):  
 return "Cooking"  
  
class Child(Father, Mother):  
 def skills(self):  
 return f"{Father.skills(self)} & {Mother.skills(self)} & Painting"  
  
c = Child()  
print(c.skills())

# 5. Hierarchical Inheritance

Multiple child classes inherit from a single parent class.

class Animal:  
 def move(self):  
 return "Moves"  
  
class Dog(Animal):  
 def bark(self):  
 return "Bark"  
  
class Cat(Animal):  
 def meow(self):  
 return "Meow"  
  
d = Dog()  
c = Cat()  
print(d.move(), d.bark())  
print(c.move(), c.meow())

# 6. Hybrid Inheritance

Combination of two or more types of inheritance.

class A:  
 def do(self):  
 return "A"  
  
class B(A):  
 def do(self):  
 return "B"  
  
class C:  
 def do(self):  
 return "C"  
  
class D(B, C):  
 def do(self):  
 return B.do(self) + " + " + C.do(self)  
  
d = D()  
print(d.do())

# 7. Complex Example: Organization Roles

Using inheritance to model roles and responsibilities in an organization.

class Employee:  
 def \_\_init\_\_(self, name):  
 self.name = name  
  
 def get\_details(self):  
 return f"Name: {self.name}"  
  
class Manager(Employee):  
 def get\_role(self):  
 return "Manages team"  
  
class Developer(Employee):  
 def get\_role(self):  
 return "Writes code"  
  
class TechLead(Manager, Developer):  
 def get\_role(self):  
 return "Manages and codes"  
  
t = TechLead("Alice")  
print(t.get\_details())  
print(t.get\_role())

# 8. Conclusion

Inheritance is a powerful concept that enhances code modularity and reusability. Python's flexibility in supporting multiple types of inheritance allows for complex system modeling. When using multiple inheritance, care must be taken to handle the method resolution order (MRO) correctly.