Object-Oriented Programming

Inheritance and Method Resolution Order (MRO)

Hà Minh Dũng

February 2026

Chapter 1

Inheritance

1.1 Definition

Inheritance is one of the core principles of Object-Oriented Programming (OOP). It allows a class called a child class to inherit attributes and methods from another class called a parent class.

Instead of rewriting common functionality, a child class can reuse and extend the behavior of its parent class.

1.2 Basic Syntax

In Python, inheritance is defined by placing the parent class inside parentheses when declaring the child class.

Example

```
class Animal:
    def eat(self):
        print("The animal is eating")

class Dog(Animal):
    def bark(self):
        print("Woof woof")
```

In this example:

- Dog inherits from Animal
- Dog can use both eat() and bark()

Testing

```
d = Dog()
d.eat()
d.bark()
```

Output

```
The animal is eating
Woof woof
```

1.3 Method Overriding

A subclass can redefine a method inherited from its parent class. This process is called method overriding.

Example

```
class Animal:
    def sound(self):
        print("Animal makes a sound")

class Dog(Animal):
    def sound(self):
        print("Dog: Woof")
```

Testing

```
d = Dog()
d.sound()
```

Output

```
Dog: Woof
```

Here, the Dog class overrides the sound() method of the Animal class.

1.4 Using `super()`

The `super()` function allows a subclass to call a method from its parent class.

Example

```
class Animal:
    def sound(self):
        print("Animal makes a sound")
class Dog(Animal):
    def sound(self):
        super().sound()
        print("Dog: Woof")
```

Output

```
Animal makes a sound
Dog: Woof
```

Note: The super() function ensures that the original behavior from the parent class is preserved while extending it.

Chapter 2

Multiple Inheritance

Python supports multiple inheritance, meaning a class can inherit from more than one parent class.

Example

```
class Animal:
    def eat(self):
        print("Animal eats")

class Walker:
    def move(self):
        print("Walking on land")

class Swimmer:
    def move(self):
        print("Swimming in water")

class Duck(Animal, Walker, Swimmer):
    pass
```

Testing

```
d = Duck()
d.eat()
d.move()
```

Output

```
Animal eats
Walking on land
```

Note: Python resolves methods from left to right in the inheritance declaration. However, when the inheritance structure becomes more complex, Python follows a specific rule called the Method Resolution Order (MRO).

Chapter 3

Method Resolution Order (MRO)

3.1 Definition

Method Resolution Order (MRO) defines the order in which Python searches for a method or attribute in a hierarchy of classes.

When a method is called, Python searches in the following order:

- The current class
- Parent classes (from left to right)
- Parent classes of parent classes
- The base `object` class

3.2 Viewing the MRO

Python provides built-in methods to inspect the MRO:

```
1 print(C.__mro__)
```

or

```
1 print(C.mro())
```

3.3 Example of MRO (Single Inheritance)

```
class Animal:
    pass

class Dog(Animal):
    pass

print(Dog.__mro__)
```

Output

```
(<class '__main__.Dog'>,<class '__main__.Animal'>, <class 'object
   '>)
```

This means Python searches in the following order:
Dog → Animal → object

3.4 Example of MRO (Multiple Inheritance)

```
class Animal:
    pass

class Walker:
    pass

class Swimmer:
    pass

class Duck(Animal, Walker, Swimmer):
    pass

print(Duck.__mro__)
```

Output

```
(<class '__main__.Duck'>,<class '__main__.Animal'>,<class '
   __main__.Walker'>,<class'__main__.Swimmer'>,<class 'object'>)
```

This means Python searches in the following order:
Duck → Animal → Walker → Swimmer → object