

Day 45



Single Inheritance in Python:

What is Inheritance?

Inheritance is an OOP (Object-Oriented Programming) concept where one class acquires properties and methods of another class.

It helps in:

- Code reuse
- Better organization
- Easier maintenance

What is Single Inheritance?

One child class inherits from exactly one parent class

Basic Syntax of Single Inheritance

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

Example:

```
1  class Animal:  
2      def speak(self):  
3          print("Animal makes a sound")  
4      def bark(self):  
5          print("Dog barks")  
6  class Dog(Animal):  
7      def bark(self):  
8          print("Dog barks")  
9  d = Dog()  
10 d.bark()  
11 d.speak()
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

● PS E:\Python\Day41-50\Day45> **python .\main.py**
Dog barks
Animal makes a sound

Multiple Inheritance in Python:

What is Multiple Inheritance?

A single child class inherits from more than one parent class.

Parent1 Parent2



Syntax of Multiple Inheritance

```
class Child(Parent1, Parent2):  
    pass
```

The order of parent classes matters.

Example: Python checks parent classes from left to right

```
13  class Father:  
14      def skills(self):  
15          print("Gardening, Driving")  
16  class Mother:  
17      def skills2(self):  
18          print("Cooking, Painting")  
19  class Child(Father, Mother):  
20      def speak(self):  
21          print("Aditya")  
22  c = Child()  
23  c.skills()  
24  c.skills2()  
--  
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS  
● PS E:\Python\Day41-50\Day45> python .\main.py  
Gardening, Driving  
Cooking, Painting
```

Example: Method Resolution Order (MRO)->MRO defines the order in which Python looks for methods.

```
13  class Father:  
14      def skills(self):  
15          print("Gardening, Driving")  
16  class Mother:  
17      def skills2(self):  
18          print("Cooking, Painting")  
19  class Child(Father, Mother):  
20      def speak(self):  
21          print("Aditya")  
22  c = Child()  
23  print(Father.mro())  
--  
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS  
● PS E:\Python\Day41-50\Day45> python .\main.py  
[<class '__main__.Father'>, <class 'object'>]
```

Example: Using super() in Multiple Inheritance-> super() follows MRO, not direct parent.

```
25 < class A:
26 |     def show(self):
27 |         print("A")
28 < class B:
29 |     def show(self):
30 |         print("B")
31 < class C(A, B):
32 |     def show(self):
33 |         super().show()
34 |         print("C")
35 z = C()
36 z.show()
```

The screenshot shows a code editor interface with a dark theme. The code in the editor is as follows:

```
25 < class A:
26 |     def show(self):
27 |         print("A")
28 < class B:
29 |     def show(self):
30 |         print("B")
31 < class C(A, B):
32 |     def show(self):
33 |         super().show()
34 |         print("C")
35 z = C()
36 z.show()
```

Below the code editor, there is a terminal window showing the output of running the script:

```
PS E:\Python\Day41-50\Day45> python .\main.py
A
C
```

The terminal tabs at the top are labeled: PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL, and PORTS. The TERMINAL tab is currently selected.

Summary:

- ✓ One child, multiple parents
- ✓ Method conflicts resolved by MRO
- ✓ super() follows MRO
- ✓ Diamond problem handled safely
- ✓ Powerful but should be used carefully

Multilevel Inheritance in Python:

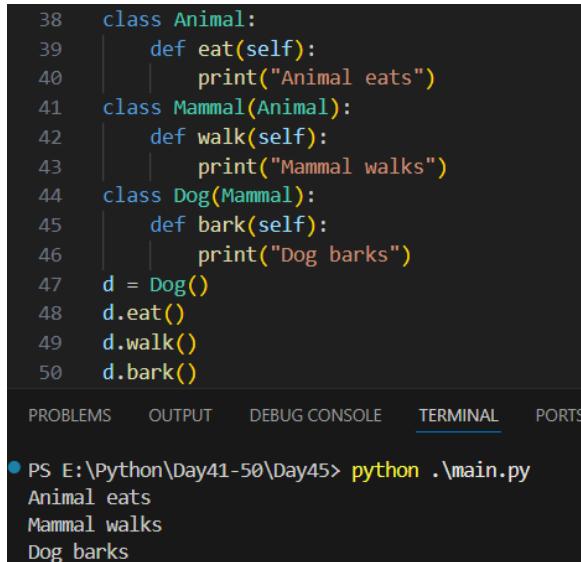
What is Multilevel Inheritance?

A class is derived from another derived class.

Syntax of Multilevel Inheritance

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

Example:

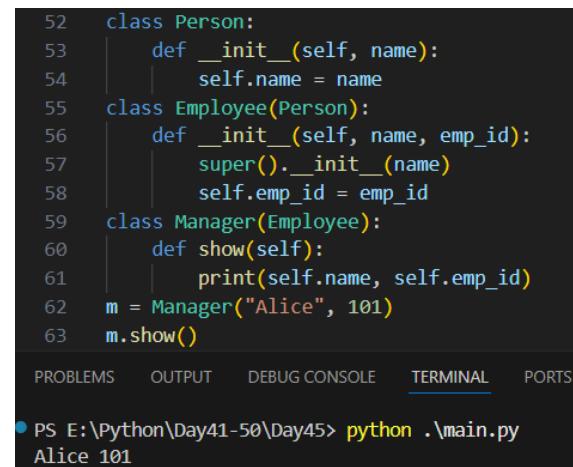


```
38     class Animal:  
39         def eat(self):  
40             print("Animal eats")  
41     class Mammal(Animal):  
42         def walk(self):  
43             print("Mammal walks")  
44     class Dog(Mammal):  
45         def bark(self):  
46             print("Dog barks")  
47     d = Dog()  
48     d.eat()  
49     d.walk()  
50     d.bark()
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

● PS E:\Python\Day41-50\Day45> python .\main.py
Animal eats
Mammal walks
Dog barks

Example: inheriting attributes. Attributes flow through levels.



```
52     class Person:  
53         def __init__(self, name):  
54             self.name = name  
55     class Employee(Person):  
56         def __init__(self, name, emp_id):  
57             super().__init__(name)  
58             self.emp_id = emp_id  
59     class Manager(Employee):  
60         def show(self):  
61             print(self.name, self.emp_id)  
62     m = Manager("Alice", 101)  
63     m.show()
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

● PS E:\Python\Day41-50\Day45> python .\main.py
Alice 101

Example: Using super() in Multilevel Inheritance. super() helps call the immediate parent's method, following the inheritance chain.

The screenshot shows a code editor with Python code demonstrating multilevel inheritance. The code defines three classes: A, B(A), and C(B). Class A has a show() method that prints "Class A". Class B(A) overrides show() by calling super().show() and then printing "Class B". Class C(B) overrides show() by calling super().show() and then printing "Class C". When an object of class C is created and its show() method is called, it prints "Class A", "Class B", and "Class C" in that order. Below the code, a terminal window shows the command "python .\main.py" and its output: "Class A", "Class B", and "Class C".

```
65 < class A:
66   < def show(self):
67     < print("Class A")
68 < class B(A):
69   < def show(self):
70     < super().show()
71     < print("Class B")
72 < class C(B):
73   < def show(self):
74     < super().show()
75     < print("Class C")
76 obj = C()
77 obj.show()

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL P

● PS E:\Python\Day41-50\Day45> python .\main.py
Class A
Class B
Class C
```

Example: Multilevel Inheritance with Constructors

The screenshot shows a code editor with Python code demonstrating multilevel inheritance with constructors. It defines three classes: A, B(A), and C(B). Each class has an __init__ method that prints a message. Class B(A) calls the __init__ method of class A using super().__init__(). Class C(B) calls the __init__ method of class B using super().__init__(). When an object of class C is created, it prints "A init", "B init", and "C init" in that order. Below the code, a terminal window shows the command "python .\main.py" and its output: "A init", "B init", and "C init".

```
79 < class A:
80   < def __init__(self):
81     < print("A init")
82 < class B(A):
83   < def __init__(self):
84     < super().__init__()
85     < print("B init")
86 < class C(B):
87   < def __init__(self):
88     < super().__init__()
89     < print("C init")
90 c = C()

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

● PS E:\Python\Day41-50\Day45> python .\main.py
A init
B init
C init
```

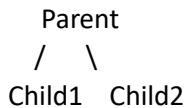
Summary:

- ✓ Multilevel inheritance forms a chain of classes
- ✓ Child inherits from parent, parent from grandparent
- ✓ super() ensures proper method calls
- ✓ Supports overriding
- ✓ Best for natural hierarchies

Hybrid and Hierarchical Inheritance in Python:

What is Hierarchical Inheritance?

Hierarchical Inheritance occurs when: One parent class is inherited by multiple child classes



Syntax:

```
class Parent:
    pass
class Child1(Parent):
    pass
class Child2(Parent):
    pass
```

Example:

```
92     class Animal:
93         def speak(self):
94             print("Animal speaks")
95     class Dog(Animal):
96         def bark(self):
97             print("Dog barks")
98     class Cat(Animal):
99         def meow(self):
100            print("Cat meows")
101 d = Dog()
102 d.speak()
103 d.bark()
104 c = Cat()
105 c.speak()
106 c.meow()
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

● PS E:\Python\Day41-50\Day45> python .\main.py

Animal speaks
Dog barks
Animal speaks
● Cat meows

Hybrid Inheritance

Hybrid Inheritance is: A combination of two or more types of inheritance

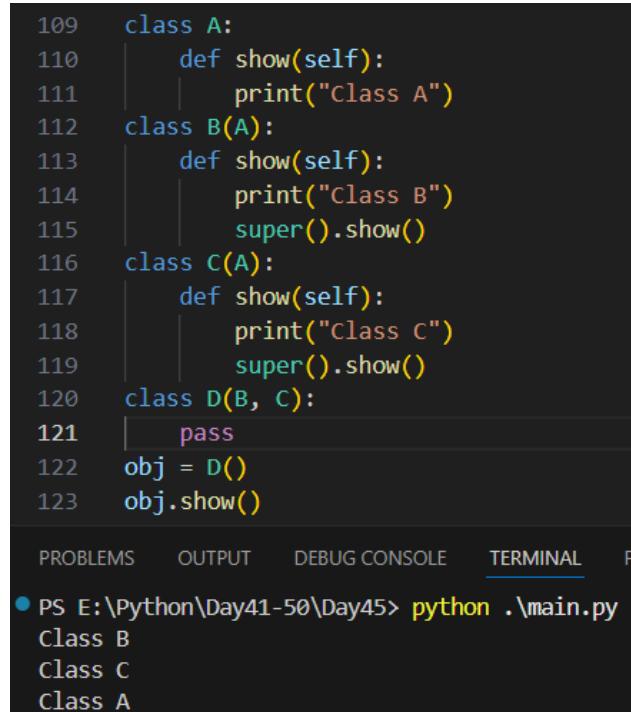
Usually involves:

- Multiple Inheritance
- Multilevel Inheritance
- Hierarchical Inheritance

Common Hybrid Structure

```
A  
/\  
B C  
 \/  
 D
```

Example:



```
109  class A:  
110      def show(self):  
111          print("Class A")  
112  class B(A):  
113      def show(self):  
114          print("Class B")  
115          super().show()  
116  class C(A):  
117      def show(self):  
118          print("Class C")  
119          super().show()  
120  class D(B, C):  
121      pass  
122  obj = D()  
123  obj.show()
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL P

● PS E:\Python\Day41-50\Day45> python .\main.py
Class B
Class C
Class A

Final Summary:

Hierarchical Inheritance

- One parent, many children
- Simple and clean
- No ambiguity

Hybrid Inheritance

- Combination of inheritance types
- Uses MRO to resolve conflicts
- Powerful but complex

--The End--