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

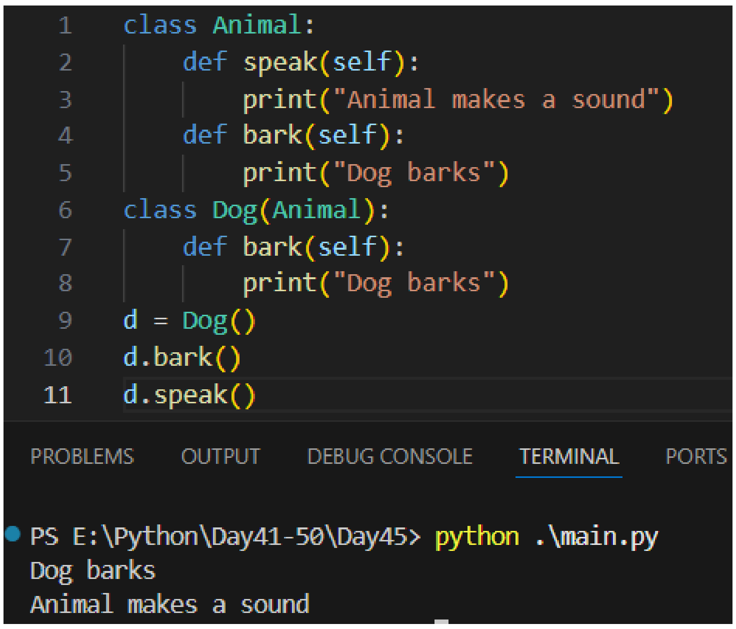
What is Single Inheritance?

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:



```
class Animal:
    def speak(self):
        print("Animal makes a sound")
    def bark(self):
        print("Dog barks")
class Dog(Animal):
    def bark(self):
        print("Dog barks")
d = Dog()
d.bark()
d.speak()
```

```
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.

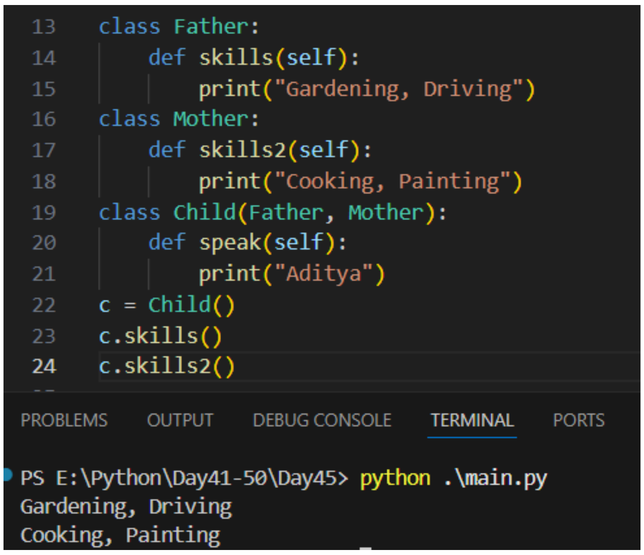
```
Parent1     Parent2
   \       /
    \     /
     Child
```

Syntax of Multiple Inheritance

class Child(Parent1, Parent2):
  pass

The order of parent classes matters.

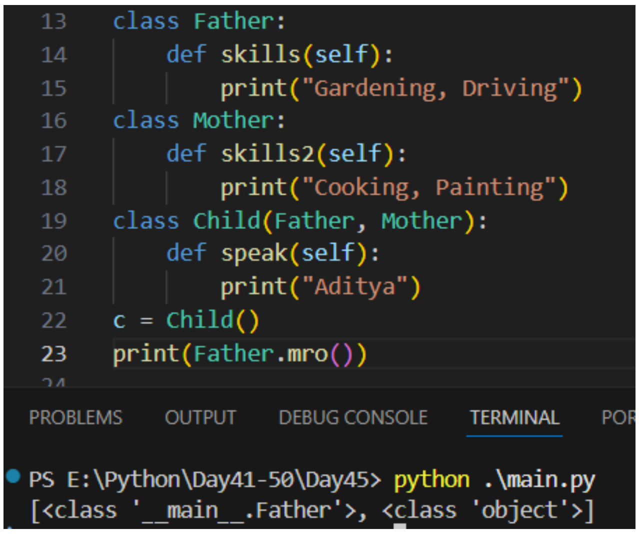
Example: Python checks parent classes from left to right



```python
class Father:
    def skills(self):
        print("Gardening, Driving")
class Mother:
    def skills2(self):
        print("Cooking, Painting")
class Child(Father, Mother):
    def speak(self):
        print("Aditya")
c = Child()
c.skills()
c.skills2()
```

```
PS E:\Python\Day41-50\Day45> python .\main.py
Gardening, Driving
Cooking, Painting
```

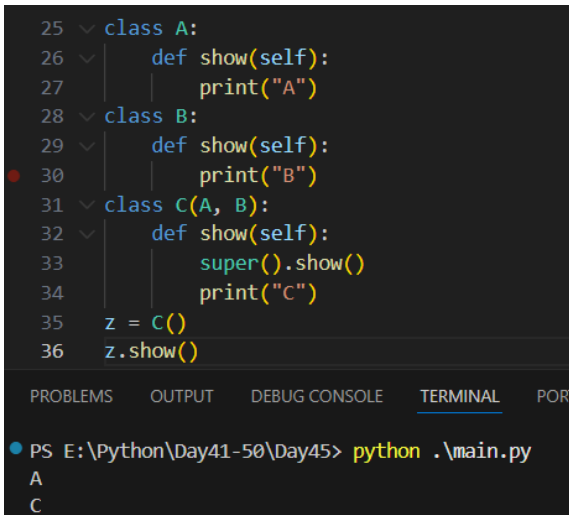
Example: Method Resolution Order (MRO)->MRO defines the order in which Python looks for methods.



```python
class Father:
    def skills(self):
        print("Gardening, Driving")
class Mother:
    def skills2(self):
        print("Cooking, Painting")
class Child(Father, Mother):
    def speak(self):
        print("Aditya")
c = Child()
print(Father.mro())
```

```
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.



```python
class A:
    def show(self):
        print("A")
class B:
    def show(self):
        print("B")
class C(A, B):
    def show(self):
        super().show()
        print("C")
z = C()
z.show()
```

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

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:

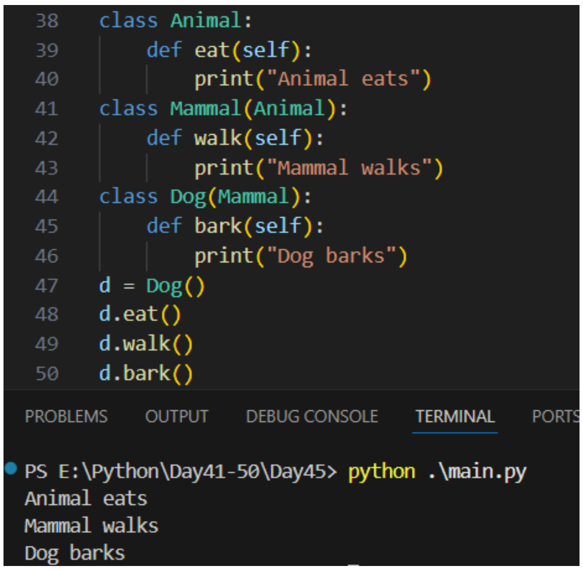
What is Multilevel Inheritance?

A class is derived from another derived class.

Syntax of Multilevel Inheritance

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

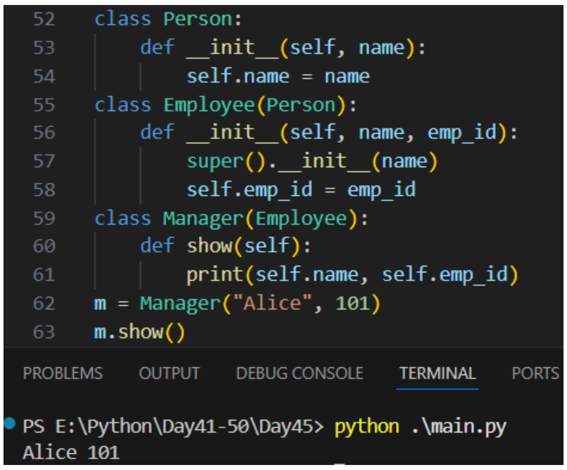
Example:



```
class Animal:
    def eat(self):
        print("Animal eats")
class Mammal(Animal):
    def walk(self):
        print("Mammal walks")
class Dog(Mammal):
    def bark(self):
        print("Dog barks")
d = Dog()
d.eat()
d.walk()
d.bark()
```

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

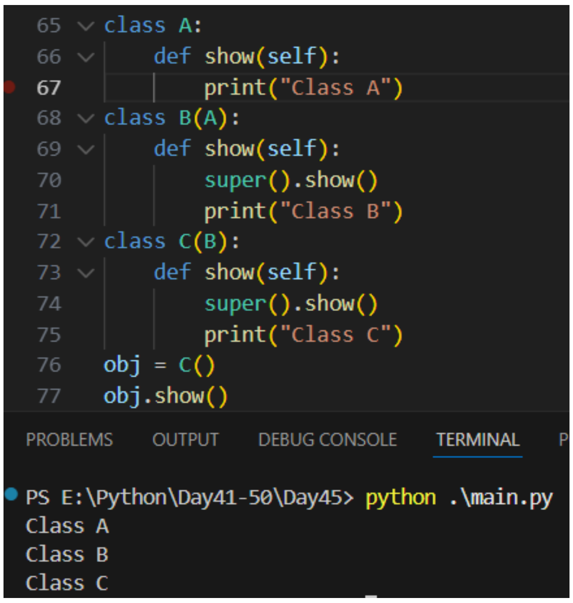
Example: inheriting attributes. Attributes flow through levels.



```
class Person:
    def __init__(self, name):
        self.name = name
class Employee(Person):
    def __init__(self, name, emp_id):
        super().__init__(name)
        self.emp_id = emp_id
class Manager(Employee):
    def show(self):
        print(self.name, self.emp_id)
m = Manager("Alice", 101)
m.show()
```

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

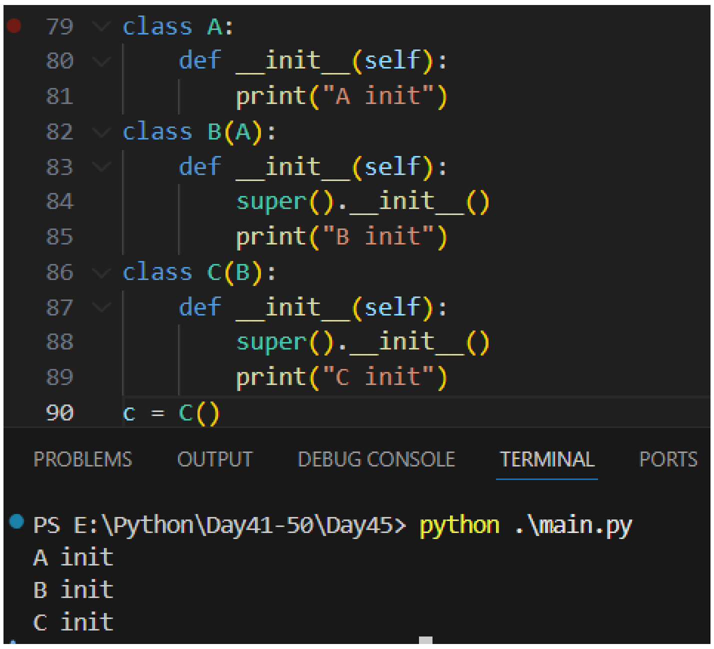
Example: Using super() in Multilevel Inheritance. super() helps call the immediate parent's method, following the inheritance chain.



```python
class A:
    def show(self):
        print("Class A")
class B(A):
    def show(self):
        super().show()
        print("Class B")
class C(B):
    def show(self):
        super().show()
        print("Class C")
obj = C()
obj.show()
```

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

Example: Multilevel Inheritance with Constructors



```python
class A:
    def __init__(self):
        print("A init")
class B(A):
    def __init__(self):
        super().__init__()
        print("B init")
class C(B):
    def __init__(self):
        super().__init__()
        print("C init")
c = C()
```

```
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?

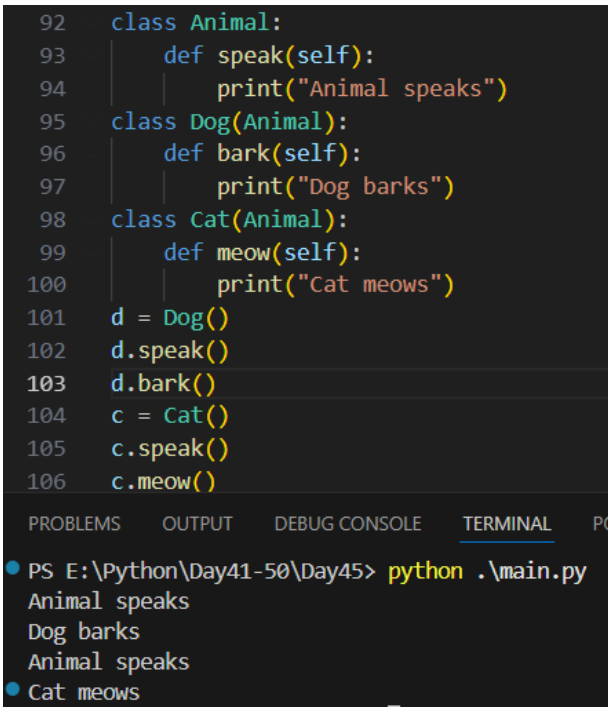
Hierarchical Inheritance occurs when: One parent class is inherited by multiple child classes

```
      Parent
     /    \
  Child1   Child2
```

Syntax:

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

Example:



```python
class Animal:
    def speak(self):
        print("Animal speaks")
class Dog(Animal):
    def bark(self):
        print("Dog barks")
class Cat(Animal):
    def meow(self):
        print("Cat meows")
d = Dog()
d.speak()
d.bark()
c = Cat()
c.speak()
c.meow()
```

```
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

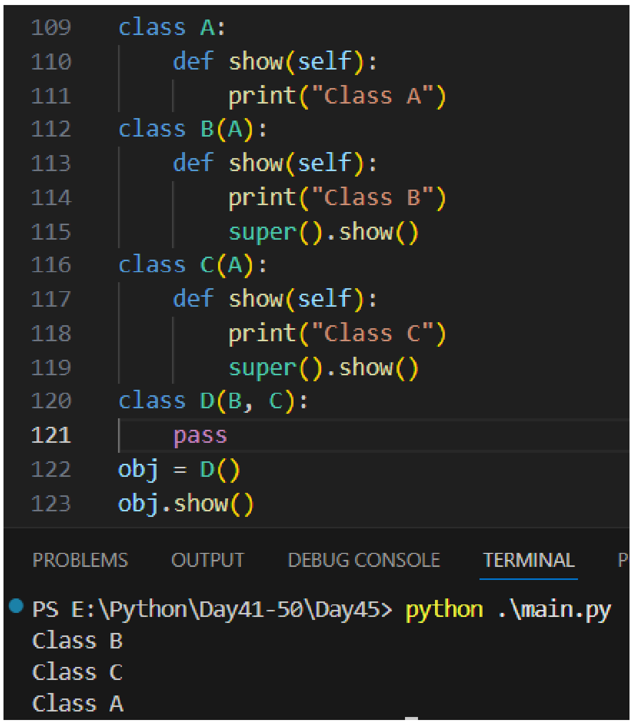
Usually involves:

- Multiple Inheritance
- Multilevel Inheritance
- Hierarchical Inheritance

Common Hybrid Structure

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

Example:



```python
class A:
    def show(self):
        print("Class A")
class B(A):
    def show(self):
        print("Class B")
        super().show()
class C(A):
    def show(self):
        print("Class C")
        super().show()
class D(B, C):
    pass
obj = D()
obj.show()
```

```
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--