

**Inheritance** in Python allows a class (child class) to derive or inherit properties and behaviors (methods and attributes) from another class (parent class). It promotes **code reuse** and establishes a hierarchical relationship between classes.

### Key Points and Syntax of Inheritance in Python

1. **Base (Parent) Class:** The class whose properties are inherited.
2. **Derived (Child) Class:** The class that inherits from the base class.
3. Syntax:

```
python Copy code  
  
class ParentClass:  
    # Parent class methods and attributes  
  
class ChildClass(ParentClass):  
    # Child class methods and attributes
```

### Constructor in Inheritance:

- The child class automatically calls the parent class constructor unless overridden.
- Use `super()` to explicitly call the parent class constructor if the child class overrides it.

### Method Overriding:

- A child class can override methods defined in the parent class.


### Types of Inheritance:

- Single Inheritance
- Multiple Inheritance
- Multilevel Inheritance
- Hierarchical Inheritance
- Hybrid Inheritance

## Examples for Each Key Point

### Single Inheritance

python

 Copy code


```
class Animal:
    def sound(self):
        return "Some sound"

class Dog(Animal):
    def sound(self): # Method overriding
        return "Bark"

dog = Dog()
print(dog.sound()) # Output: Bark
```

### Using `super()` for Constructor

python

 Copy code

```
class Parent:
    def __init__(self, name):
        self.name = name

class Child(Parent):
    def __init__(self, name, age):
        super().__init__(name) # Call parent constructor
        self.age = age

child = Child("Alice", 25)
print(child.name) # Output: Alice
print(child.age) # Output: 25
```

## Multiple Inheritance

Python allows a child class to inherit from multiple parent classes:

python Copy code

```
class Parent1:
    def method1(self):
        return "Parent1 method"

class Parent2:
    def method2(self):
        return "Parent2 method"

class Child(Parent1, Parent2):
    pass

child = Child()
print(child.method1()) # Output: Parent1 method
print(child.method2()) # Output: Parent2 method
```


Note: Python resolves conflicts using the Method Resolution Order (MRO), which is determined by the C3 linearization algorithm.

## Differences Between Python and Java Inheritance

Feature	Python	Java
Support for Multiple Inheritance	Fully supported	Not directly supported (achieved via interfaces).
Method Overriding	Achieved with the same method name in child class	Same, but annotations like <code>@Override</code> are used for clarity.
Access Modifiers	Uses <code>_</code> (protected) and <code>__</code> (private convention); actual enforcement is relaxed	Enforced strictly with <code>private</code> , <code>protected</code> , <code>public</code> .
Parent Class Constructor Call	Use <code>super()</code> to call parent constructor	Same, <code>super()</code> must be the first statement.
MRO (Method Resolution Order)	Uses C3 Linearization for resolving conflicts in multiple inheritance	No MRO in Java as it does not allow multiple inheritance of classes.
Abstract Classes	Can use abstract base classes ( <code>ABC</code> ) from <code>abc</code> module	Uses <code>abstract</code> keyword and interfaces.

## Code Example: Overriding a Method in Inheritance

python

 Copy code

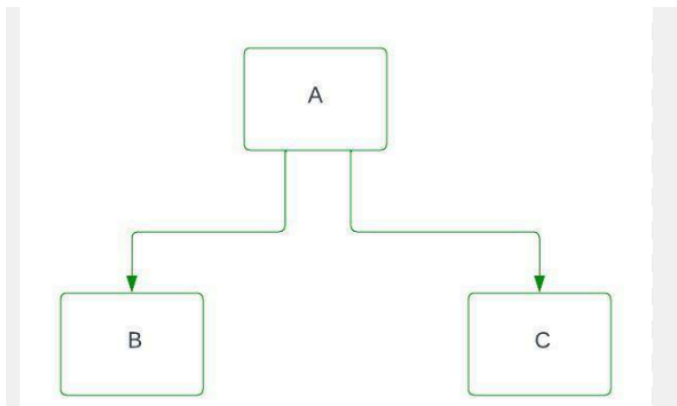
```
class Parent:
    def greet(self):
        return "Hello from Parent!"

class Child(Parent):
    def greet(self): # Overriding the 'greet' method from the Parent class
        return "Hello from Child!"

# Create objects
parent = Parent()
child = Child()

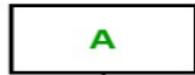
# Call the greet method from both parent and child objects
print(parent.greet()) # Output: Hello from Parent!
print(child.greet())  # Output: Hello from Child!
```

## Hierarchical Inheritance



## Multilevel Inheritance

Base class



Intermediary  
class



Derived class



**A**

**B**

**C**