🧾 Python Inheritance - Documentation

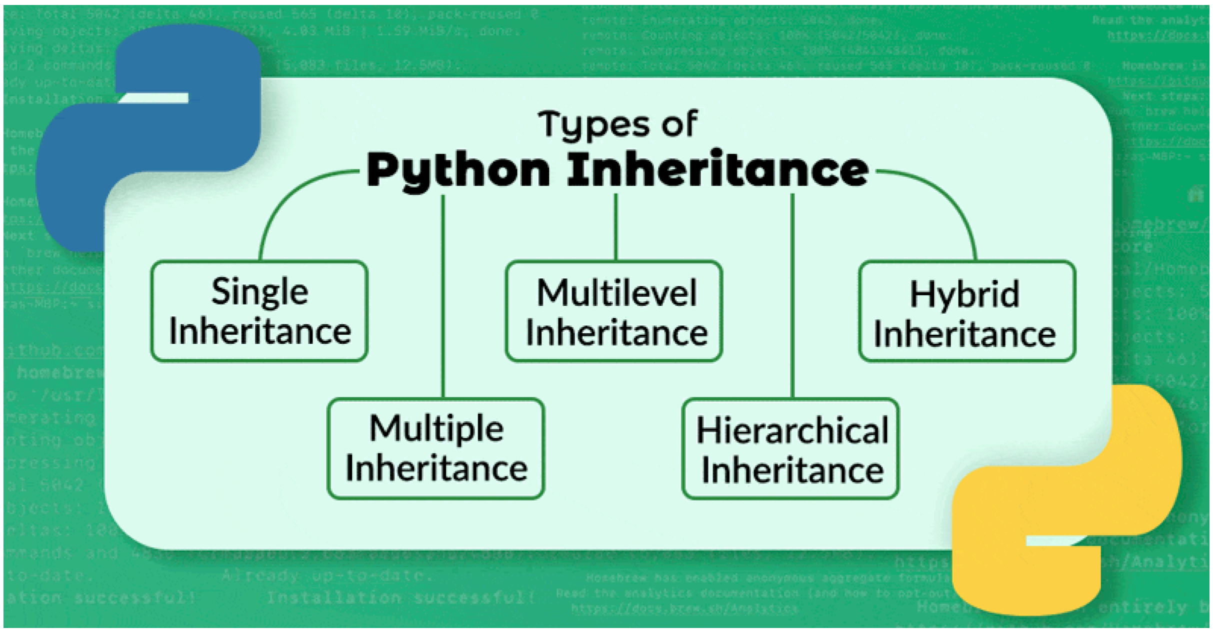
🔰 What is Inheritance?

Inheritance is a fundamental concept in object-oriented programming (OOP) that allows a class (called a **child** or **subclass**) to inherit attributes and methods from another class (called a **parent** or **superclass**).

✅ Why Use Inheritance?•

- **Code Reusability**: Reuse code from existing classes.
- **Extensibility**: Easily add or extend functionality.
- **Organized Structure**: Logical relationship between classes.

🔡 Types of Inheritance in Python



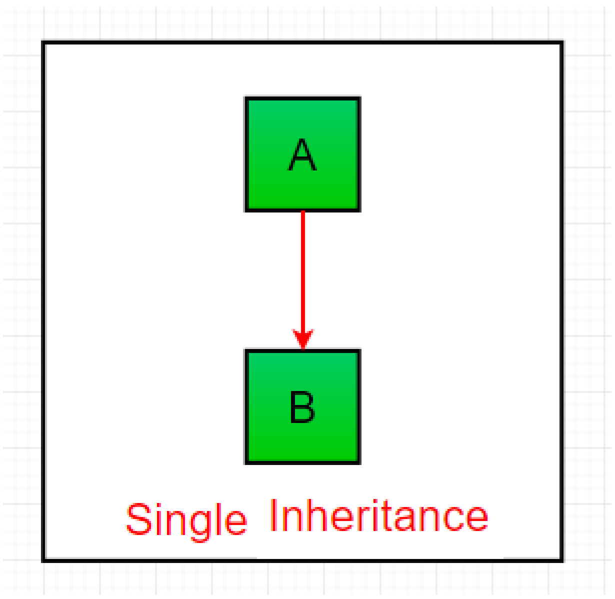
1. **Single Inheritance**
2. **Multilevel Inheritance**
3. **Multiple Inheritance**

4. **Hierarchical Inheritance**
5. **Hybrid Inheritance**

📘 Single Inheritance

💡

> **Single inheritance** involves one *base class* (also known as *superclass*) and one *derived class* (also known as *subclass*). The subclass inherits attributes and methods from the superclass.



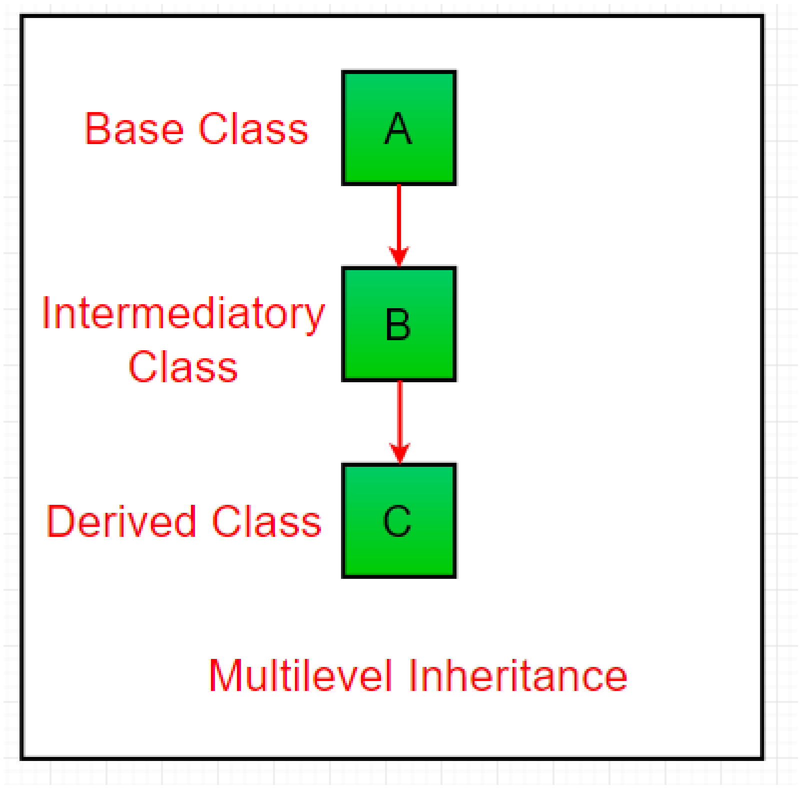
🔷 **Example:**

```
class Animal:
    def speak(self):
        return "Animal speaks"

class Dog(Animal):
    def bark(self):
        return "Dog barks"

dog = Dog()
```

```
print(dog.speak())  # Output: Animal speaks
print(dog.bark())   # Output: Dog barks
```

📘 Multilevel Inheritance

💡 **Multilevel inheritance** occurs when a class is derived from another derived class, forming a multi-level hierarchy. This means there is a chain of inheritance where a subclass becomes a superclass for another subclass.

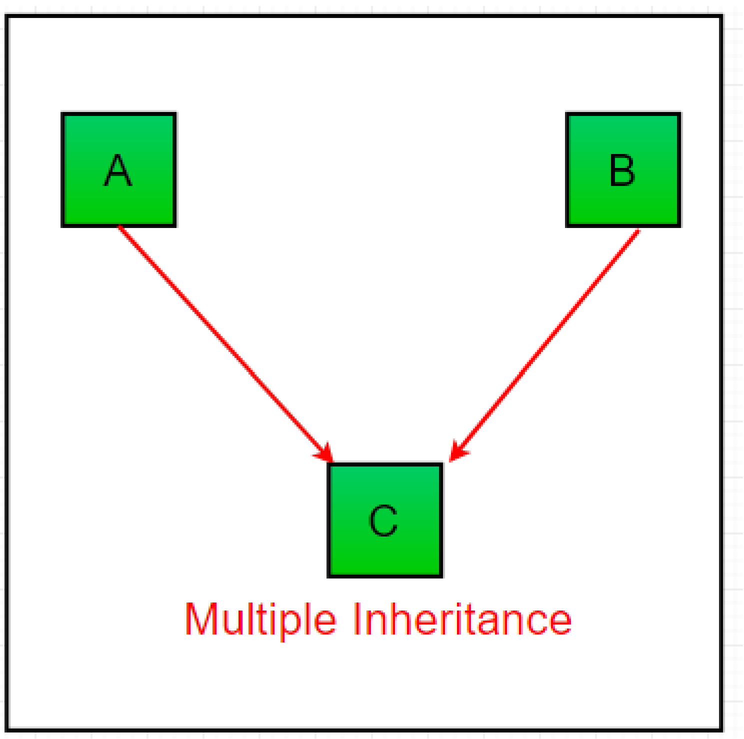


🔷 **Example:**

```
class Animal:
    def speak(self):
        return "Animal speaks"

class Dog(Animal):
    def bark(self):
        return "Dog barks"

class Puppy(Dog):
    def weep(self):
        return "Puppy weeps"

puppy = Puppy()
print(puppy.speak())  # Animal speaks
print(puppy.bark())   # Dog barks
print(puppy.weep())   # Puppy weeps
```

📘 Multiple Inheritance

💡 **Multiple inheritance** allows a class to inherit from more than one base class. This means the derived class can have multiple superclasses. While multiple inheritance is not supported in some languages like Java due to complexities like the Diamond Problem, Python handles it efficiently without confusion.

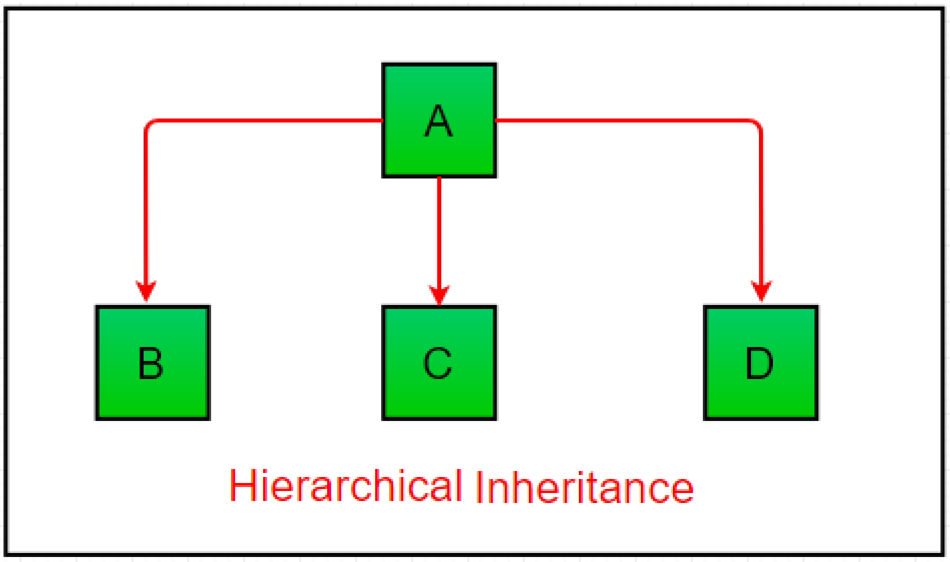


🔷 **Example:**

```
class Father:
    def skills(self):
        return "Gardening, Programming"

class Mother:
    def skills(self):
        return "Cooking, Art"

class Child(Father, Mother):
    def my_skills(self):
        return f"Father's skills: {Father.skills(self)}\nMother's skills: {Mother.sk
ills(self)}"

child = Child()
print(child.my_skills())
```

📘 Hierarchical Inheritance

💡 **Hierarchical inheritance** involves multiple derived classes inheriting from a single base class. This creates a hierarchy where several subclasses share the same superclass.

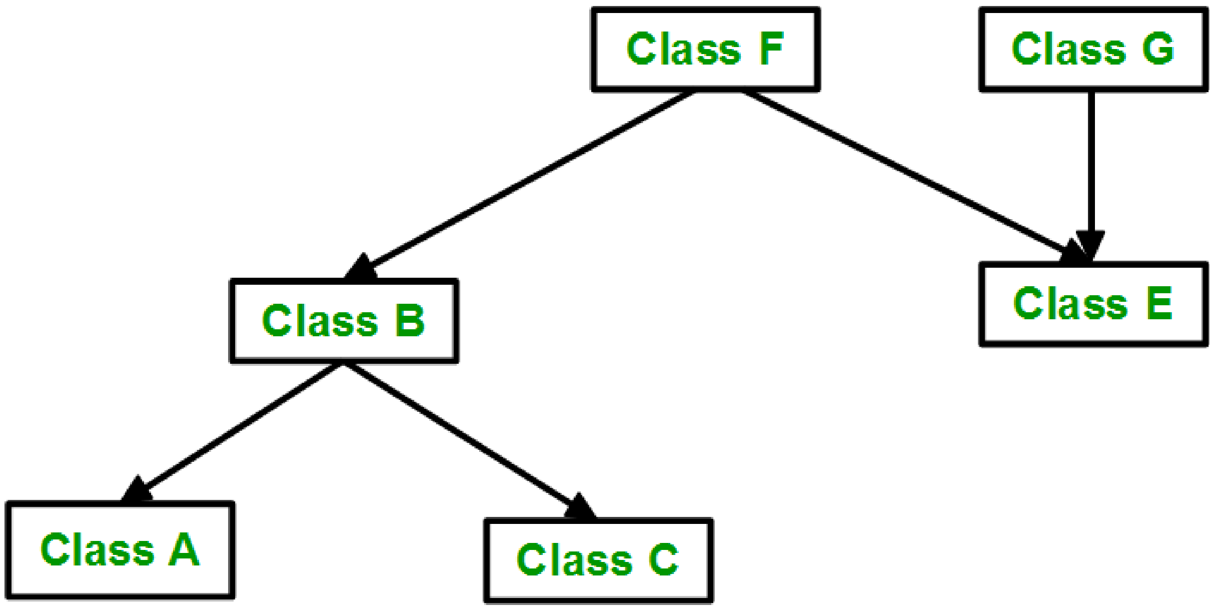


🔷 **Example:**

```
class Animal:
    def move(self):
        return "Can move"

class Dog(Animal):
    def bark(self):
        return "Dog barks"

class Cat(Animal):
    def meow(self):
        return "Cat meows"

dog = Dog()
cat = Cat()
print(dog.move(), dog.bark())
print(cat.move(), cat.meow())
```

📘 Hybrid Inheritance

💡 **Hybrid inheritance** is a combination of two or more types of inheritance. It is commonly used in the industry to model complex relationships. Hybrid inheritance can be a mix of multilevel and multiple inheritance, providing flexibility in designing class hierarchies.



📘 Using super() in Inheritance

🔷 **Example:**

```
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

c = Child("Alice", 10)
print(c.name, c.age)  # Output: Alice 10
```

📘 Method Overriding

🔷 **Example:**

```
class Animal:
    def sound(self):
        return "Generic animal sound"
```

```
class Dog(Animal):
    def sound(self):
        return "Bark!"

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

📘 Abstract Classes (with abc module)

🔷 Example:

```
from abc import ABC, abstractmethod

class Shape(ABC):
    @abstractmethod
    def area(self):
        pass

class Square(Shape):
    def __init__(self, side):
        self.side = side

    def area(self):
        return self.side * self.side

s = Square(4)
print(s.area())  # Output: 16
```

🧠 Best Practices

- Use super() to call parent constructors/methods.
- Avoid unnecessary multiple inheritance if not needed.
- Use abstraction for base class definitions.