When you create a subclass that **inherits** from a parent class, you can **override** (replace) methods or attributes of the parent class by redefining them in the child class.

**🔹 1. Basic Method Override**

class **Animal**:

def speak(self):

return "Some sound"

class Dog(**Animal**): # Dog inherits from Animal

def speak(self): # override the parent **method**

return "Woof!"

class Cat(Animal):

def speak(self): # override again

return "Meow"

dog = Dog()

cat = Cat()

print(dog.speak()) # Woof!

print(cat.speak()) # Meow

Here Dog and Cat both **override** the speak() method from Animal.

**🔹 2. Overriding Attributes**

class **Vehicle**:

wheels = 4

class Bike(**Vehicle**):

wheels = 2 # override attribute

print(Vehicle.wheels) # 4

print(Bike.wheels) # 2

**🔹 3. Inherit an attribute from Parent class**

Sometimes you don’t want to completely replace the parent method, but rather extend it. You can call the parent version with super().

class **Person**:

**def greet(self):**

return "Hello!"

class Student(**Person**):

def greet(self):

**parent\_greet = super().greet**()#transfer Parent method to child class.

return **parent\_greet** + " I'm a student."

s = Student()

print(s.greet())

# Hello! I'm a student.

**🔹 4. Overriding the constructor (\_\_init\_\_)**

class **Animal**:

**def \_\_init\_\_(self, name):**

self.name = name

class Dog(**Animal**):

**def \_\_init\_\_(self, name, breed):**

super().\_\_init\_\_(name) # call parent constructor

self.breed = breed

dog = Dog("Buddy", "Labrador")

print(dog.name, dog.breed) # Buddy Labrador

✅ So, overriding means:

* If a **child class** has a method/attribute with the same name as in the parent, Python will use the child’s version first.
* If you still want parent behavior → use super().