⚙️ Method Overloading & Overriding (OOP Concepts)

🧠 Theory

🔹 Method Overloading

Definition:
Method Overloading means defining multiple methods with the **same name** but **different numbers or types of parameters**.

However, **Python does not support true method overloading** like Java or C++.
Instead, it can be achieved through:

- **Default arguments**
- **Variable-length arguments** (*args / **kwargs)
  These allow a single method to handle different kinds of input.

Purpose:

- Makes methods more flexible.
- Simplifies code readability by allowing one method name to handle multiple cases.

💻 Example — Method Overloading

```
class Calculator:
    def add(self, a=0, b=0, c=0):
        """Simulates method overloading using default parameters."""
        return a + b + c


# Creating object
calc = Calculator()

print("Sum of one number:", calc.add(5))
print("Sum of two numbers:", calc.add(5, 10))
print("Sum of three numbers:", calc.add(5, 10, 15))
```

Output:

Sum of one number: 5

Sum of two numbers: 15

Sum of three numbers: 30

🔹 **Method Overriding**

Definition:
Method Overriding occurs when a **child class redefines a method** from its **parent class** using the same name and parameters.

Purpose:

- Allows a subclass to **customize or extend** behavior from the parent class.
- Enables **runtime polymorphism**, where the method that runs depends on the object's type.

💻 **Example — Method Overriding**

```
class Animal:
    def sound(self):
        print("Some generic animal sound")


class Dog(Animal):
    def sound(self):
        print("Dog barks: Woof!")


# Creating objects
a = Animal()
d = Dog()


a.sound()   # Calls parent method
d.sound()   # Calls overridden method in child class
```

Output:

Some generic animal sound

Dog barks: Woof!

🔹 **Using super() in Method Overriding**

super() is used to call the **parent class's method** inside the **child class**,
allowing the child to reuse or extend parent functionality.

Example:

```
class Vehicle:
    def start(self):
        print("Vehicle started")


class Car(Vehicle):
    def start(self):
        super().start()  # Call parent class method
        print("Car engine running smoothly!")


car = Car()
car.start()
```

Output:

Vehicle started

Car engine running smoothly!

✅ **Summary Table**

Concept	Relationship	Purpose	Behavior
Overloading	Within same class	Handle multiple input types/counts	Achieved via default or variable arguments
Overriding	Parent–Child classes	Redefine or extend parent method	Achieved at runtime

Concept	Relationship	Purpose	Behavior
super()	Used in child class	Access parent method	Enables code reusability