In Python, the concepts of **method overriding** and **method overloading** work similarly to other object-oriented languages, but there are a few differences in how they are implemented. Let's go over them in detail:-

1. Method Overloading in Python

Unlike languages like Java or C++, Python does not support method overloading in the traditional sense. In those languages, method overloading means defining multiple methods with the same name but different parameters. However, Python doesn't allow multiple methods with the same name in a class; only the last definition of a method is retained.

That said, we can still simulate method overloading by using default arguments, variable-length argument lists (`*args` and `**kwargs`), or conditional logic inside a single method.

Example (Simulating Method Overloading):

```
class MathOperations:
    # Simulate overloading by using *args for variable number of arguments
    def add(self, *args):
        return sum(args)

# Usage
math = MathOperations()
print(math.add(2, 3))       # Output: 5 (two arguments)
print(math.add(2, 3, 4))    # Output: 9 (three arguments)
print(math.add(1, 2, 3, 4)) # Output: 10 (four arguments)
```

In this example, the `add` method can accept any number of arguments due to `*args`, and it behaves like an overloaded method by summing the numbers passed to it.

2. Method Overriding in Python

Method overriding occurs when a subclass provides a **specific implementation** of a method that is already defined in its **parent class**. The method in the child class has the **same name**, **same parameters**, and **same return type** as the method in the parent class.

When a method is overridden in a subclass, the subclass's version of the method gets executed, not the parent class's version.

Example (Method Overriding in Python):

```
class Animal:
    def sound(self):
        print("Animal makes a sound")

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

class Cat(Animal):
    def sound(self):
        print("Cat meows")

# Usage
animal = Animal()
dog = Dog()
cat = Cat()

animal.sound()  # Output: Animal makes a sound
dog.sound()     # Output: Dog barks
cat.sound()     # Output: Cat meows
```

In this example:

- The `Dog` and `Cat` classes override the `sound` method of the `Animal` class to provide their own implementations.
- When `sound()` is called on an instance of `Dog`, the overridden `sound` method in the `Dog` class is executed.

Summary of Method Overloading and Overriding in Python:

Feature	Method Overloading	Method Overriding
Definition	Same method name, but different parameters (simulated using `*args` or default arguments)	Same method name, same parameters
Usage	Not directly supported, use variable-length arguments or default arguments	Replace the parent class method with a subclass implementation
Resolution	At runtime (dynamic)	At runtime (dynamic)
Inheritance	Not required for overloading	Involves inheritance (subclass/parent class)
Python Specific	Python does not support multiple methods with the same name, overloading is	Python supports method overriding naturally

Feature	Method Overloading	Method Overriding
	simulated	

In conclusion:

- **Method overloading** in Python is simulated using features like `*args` and `**kwargs` but isn't directly supported by the language syntax.
- **Method overriding** works in Python as expected, where a subclass can provide a custom implementation of a method that already exists in its parent class.