Method Overriding in Python | Python Tutorial - Day #74

Day 74 - Method Overriding in Python

Method overriding is a powerful feature in object-oriented programming that allows you to redefine a method in a derived class. The method in the derived class is said to override the method in the base class. When you create an instance of the derived class and call the overridden method, the version of the method in the derived class is executed, rather than the version in the base class.
In Python, method overriding is a way to customize the behavior of a class based on its specific needs. For example, consider the following base class:

```
class Shape:
    def area(self):
        pass
```

In this base class, the area method is defined, but does not have any implementation. If you want to create a derived class that represents a circle, you can override the area method and provide an implementation that calculates the area of a circle:

```
class Circle(Shape):
    def __init__(self, radius):
        self.radius = radius

    def area(self):
        return 3.14 * self.radius * self.radius
```

In this example, the Circle class inherits from the Shape class, and overrides the area method. The new implementation of the area method calculates the area of a circle, based on its radius.
It's important to note that when you override a method, the new implementation must have the same method signature as the original method. This means that the number and type of arguments, as well as the return type, must be the same.
Another way to customize the behavior of a class is to call the base class method from the derived class method. To do this, you can use the super function. The super function allows you to call the base class method from the derived class method, and can be useful when you want to extend the behavior of the base class method, rather than replace it.

For example, consider the following base class:

```
class Shape:
    def area(self):
        print("Calculating area...")
```

In this base class, the area method prints a message indicating that the area is being calculated. If you want to create a derived class that represents a circle, and you also want to print a message indicating the type of shape, you can use the super function to call the base class method, and add your own message:

```
class Circle(Shape):
    def __init__(self, radius):
        self.radius = radius

    def area(self):
        print("Calculating area of a circle...")
        super().area()
        return 3.14 * self.radius * self.radius
```

In this example, the Circle class overrides the area method, and calls the base class method using the super function. This allows you to extend the behavior of the base class method, while still maintaining its original behavior.
In conclusion, method overriding is a powerful feature in Python that allows you to customize the behavior of a class based on its specific needs. By using method overriding, you can create more

robust and reliable code, and ensure that your classes behave in the way that you need them to. Additionally, by using the super function, you can extend the behavior of a base class method, rather than replace it, giving you even greater flexibility and control over the behavior of your classes.

Previous Next

CodeWithHarry