🧩 Short Answer:

✅ **Yes — method overriding is a form of *runtime polymorphism*.**

🧠 Let’s break it down

1️⃣ Polymorphism means:

> “Many forms” — same method name, but different behavior depending on the object.

It’s a **broad concept** that can appear in multiple ways (both compile-time and runtime, depending on the language).

2️⃣ Method Overriding

> When a **child class** defines a method that already exists in its **parent class**, and gives it a new implementation.

So when you call that method on an object, **Python decides at runtime** which version to use (the parent’s or the child’s).

That’s exactly **runtime polymorphism**.

⚙️ Example

```
class Car:
    def fuel_type(self):
        return "Petrol or Diesel"

class ElectricCar(Car):
    def fuel_type(self):
        return "Electric Charge"

vehicles = [Car(), ElectricCar()]
```

```python
for v in vehicles:
    print(v.fuel_type())
```

🧾 Output:

```
Petrol or Diesel
Electric Charge
```

Here:

- Both objects respond to the same message (`fuel_type()`),
- But **each behaves differently** → *polymorphism in action.*

🧠 So in summary

Concept	Description
Polymorphism	The *concept* — same interface, different behavior.
Method Overriding	A *way to implement* polymorphism using inheritance.
When it happens	At **runtime**, when the program decides which version to call.

⚙️ For contrast — Python also supports:

- **Overloading (compile-time polymorphism)** → same function name, different parameters.
  But Python doesn't truly support overloading like Java or C++; it uses *default arguments* or `*args` instead.

✅ So yes:

> Method overriding is one of the main techniques used to achieve **polymorphism** in Object-Oriented Programming — specifically **runtime polymorphism**.