OOP Principles: Inheritance, Polymorphism, and Encapsulation (1.5 hours) 👩‍🏫👨‍🏫

- **Inheritance** 👨‍👩‍👧: Just like humans inherit traits from their parents, classes can inherit attributes and methods from other classes. This helps reduce code repetition and create a natural hierarchy in your code!
  - **Example:** Imagine a `Vehicle` class with general features (like wheels). We can create subclasses like `Car` and `Bike` that inherit those features!

python

```python
class Vehicle:
    def __init__(self, wheels):
        self.wheels = wheels

class Car(Vehicle):
    pass

car = Car(4)
print(car.wheels)  # Output: 4
```

Polymorphism 🦄: Derived classes can behave differently for the same method inherited from a base class. With polymorphism, a method name can mean different things across multiple classes.

- **Example:** Imagine a `speak()` method. Dogs `bark()`, while cats `meow()`, even though both use `speak()`!

```python
class Dog:
    def speak(self):
        return "Woof!"

class Cat:
    def speak(self):
        return "Meow!"

# Polymorphism in action
for animal in [Dog(), Cat()]:
    print(animal.speak())
```

Encapsulation 🔐: This is the practice of keeping attributes and methods private to prevent unwanted interference from outside the class. It's like hiding your chocolate stash 🍫 from everyone else!

```
class SecretStash:
    def __init__(self):
        self.__chocolates = 10  # Private attribute

    def take_chocolate(self):
        if self.__chocolates > 0:
            self.__chocolates -= 1
            print("One chocolate taken!")
        else:
            print("No chocolates left 😢")

stash = SecretStash()
stash.take_chocolate()
```

In Summary
OOP allows you to organize code in a way that’s fun, reusable, and efficient! As you practice, imagine the real-world objects around you and think of how they could become classes in your code. Whether you’re designing a `Smartphone`, `Pet`, or `Superhero`, OOP gives you the power to build programs that feel like real-world systems.

Happy coding! 🎉