Phase 3: Object-Oriented Programming (OOP)

OOP is a powerful programming paradigm that helps you design and structure your code by modeling real-world entities as "objects."

- **Classes and Objects:**
  - **Class:** A blueprint or template for creating objects. It defines the common attributes (data) and methods (functions/actions) that all objects of that type will have.
  - **Object (Instance):** A specific, concrete instance created from a class. If `Dog` is a class, then `my_dog` and `your_dog` are objects of the `Dog` class, each with their own unique name, age, etc.
  - **`__init__` method:** A special method (constructor) that runs automatically when a new object is created. It's used to initialize the object's attributes.
  - **`self`:** A reference to the current instance of the class. It's the first parameter in all instance methods and allows methods to access the object's own attributes and other methods.
- **Inheritance:**
  - A mechanism that allows a new class (the **child** or **derived** class) to inherit attributes and methods from an existing class (the **parent** or **base** class).
  - This promotes code reuse and establishes a "is-a" relationship (e.g., a `Car` *is a* `Vehicle`).
  - Child classes can also override (redefine) methods from their parent class to provide specific behavior.
- **Polymorphism:**
  - Meaning "many forms." It refers to the ability of different objects to respond to the same method call in their own unique way.
  - For example, if both a `Dog` object and a `Cat` object have a `speak()` method, calling `animal.speak()` will produce a "Woof!" for a dog and a "Meow!" for a cat, even though the method name is the same.
- **Encapsulation and Abstraction:**
  - **Encapsulation:** The practice of bundling data (attributes) and the methods that operate on that data within a single unit (the class). It also involves restricting direct access to some of an object's components, controlling how data is modified (e.g., using "private" attributes, typically prefixed with `__`).
  - **Abstraction:** Hiding the complex implementation details and showing only the essential features or functionalities to the user. When you use an object, you interact with its public methods without needing to know the intricate internal logic.