Object-Oriented Programming (OOP) Basics in Python

AN INTRODUCTION TO CLASSES, OBJECTS, AND CORE CONCEPTS

What is OOP?

OOP stands for Object-Oriented Programming

Focuses on "objects" that encapsulate data and behavior

Encourages:

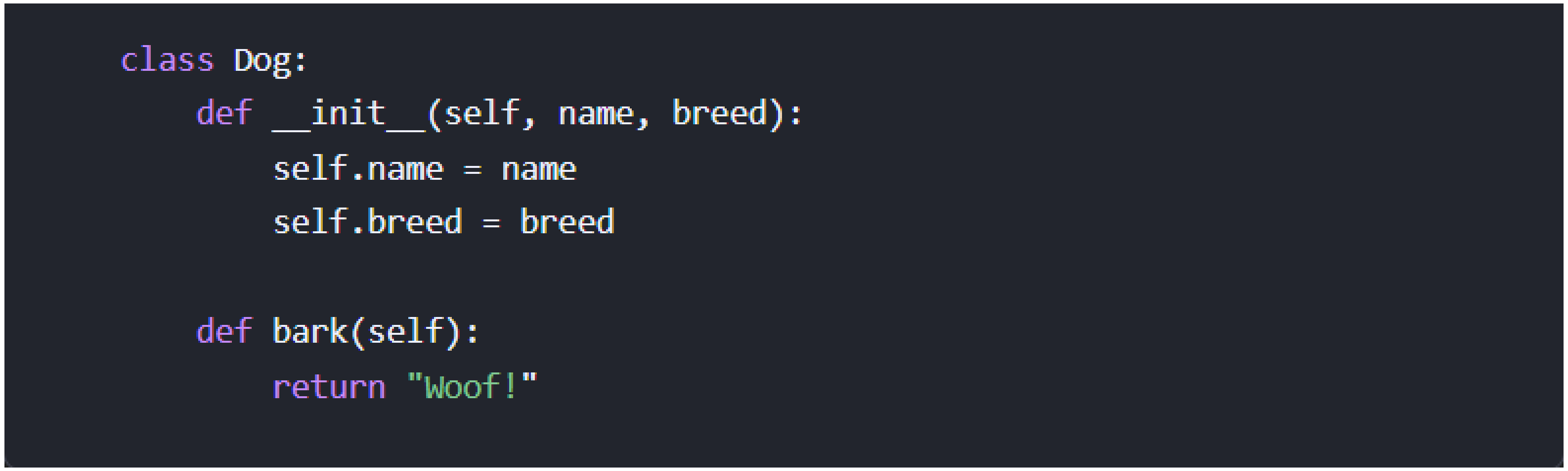
- Code reusability
- Modularity
- Maintainability

Classes

Blueprints for creating objects.

Define the attributes and methods that objects of that class will have.

Example:



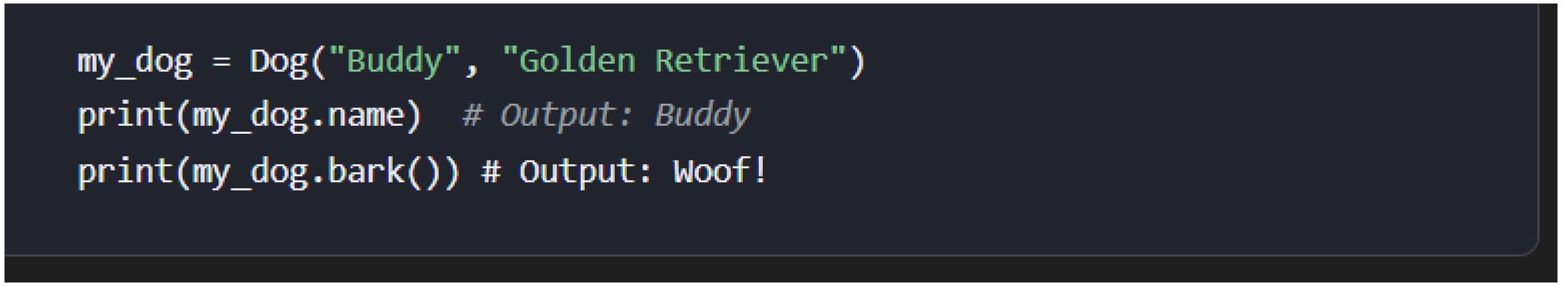
```python
class Dog:
    def __init__(self, name, breed):
        self.name = name
        self.breed = breed

    def bark(self):
        return "Woof!"
```

Objects

Instances of classes.

Represent real-world entities or concepts.

Example:



```
my_dog = Dog("Buddy", "Golden Retriever")
print(my_dog.name)  # Output: Buddy
print(my_dog.bark()) # Output: Woof!
```

Encapsulation

Bundles data and methods inside a class

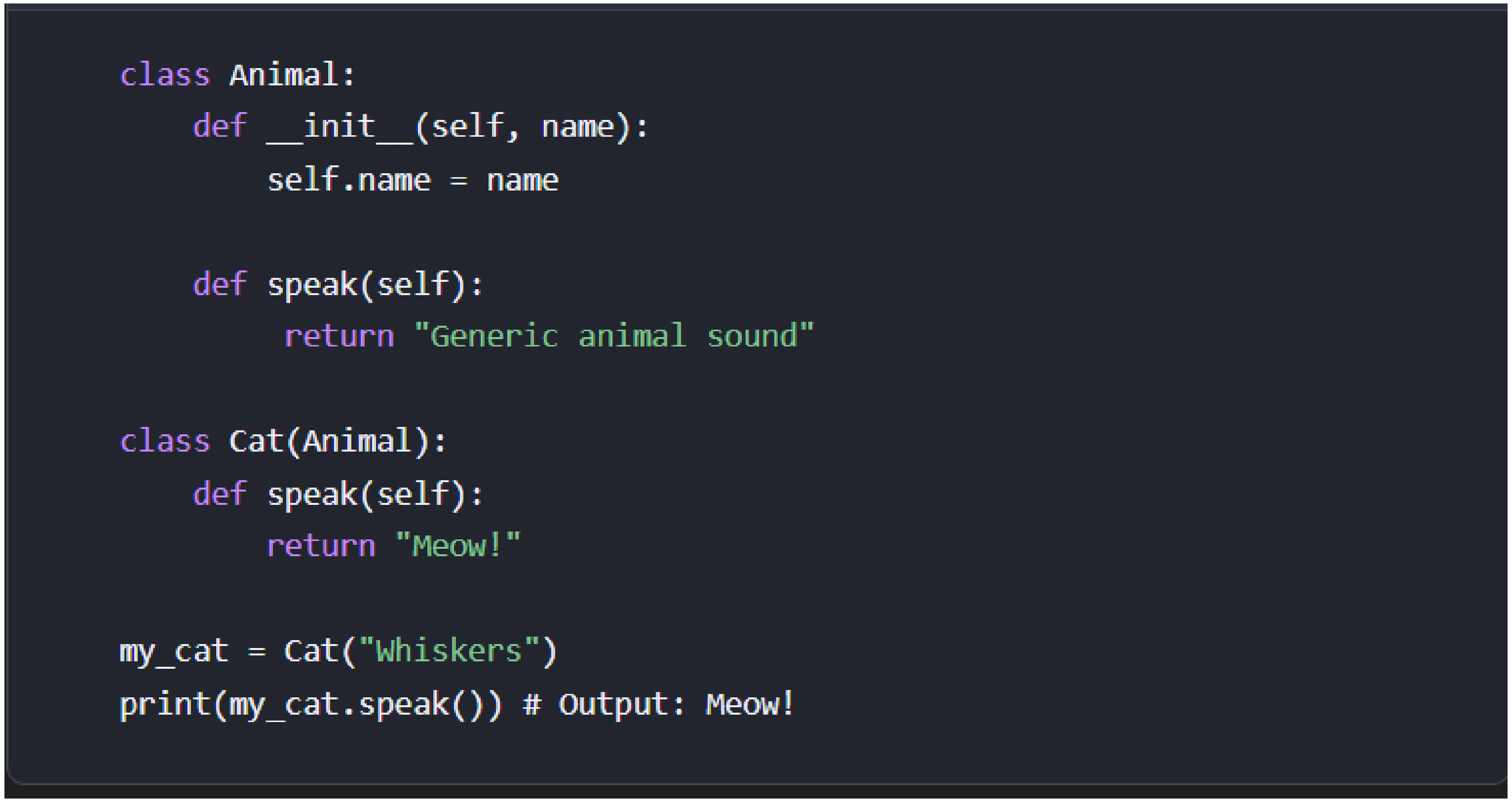
Restricts external access to data

Access Modifiers:

- Public: self.name
- Protected: self._name
- Private: self.__name

Inheritance

Let me explain with an example:



```python
class Animal:
    def __init__(self, name):
        self.name = name

    def speak(self):
         return "Generic animal sound"

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

my_cat = Cat("Whiskers")
print(my_cat.speak()) # Output: Meow!
```

Polymorphism

Different classes respond differently to the same method

Achieved through method overriding

Allows flexibility and extensibility

Abstraction

Hides complex details, shows only essentials

Achieved via abstract classes (abc module)

Simplifies usage and maintenance

Benefits of OOP

- ✅ Code Reusability
- ✅ Modularity & Maintainability
- ✅ Data Protection
- ✅ Flexibility through Polymorphism
- ✅ Real-World Modeling

Summary

OOP is a powerful paradigm for organizing and managing code

Python supports all major OOP principles:

- Classes
- Objects
- Encapsulation
- Inheritance
- Polymorphism
- Abstraction

Thank You!

Questions?