Object-Oriented Programming in Python

Building Better Code with Classes and Objects

What is OOP?

Object-Oriented Programming organizes code around **objects** that contain:

- **Data** (attributes/properties)
- **Behavior** (methods/functions)

Four Pillars of OOP:

1. Encapsulation
2. Abstraction
3. Inheritance
4. Polymorphism

Classes and Objects

A **class** is a blueprint, an **object** is an instance of that class.

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

    def bark(self):
        return f"{self.name} says Woof!"

# Creating objects
buddy = Dog("Buddy", 3)
luna = Dog("Luna", 5)

print(buddy.bark())  # Buddy says Woof!
print(luna.bark())   # Luna says Woof!
```

Encapsulation

Hide internal data and provide controlled access

```
class BankAccount:
    def __init__(self, owner, balance):
        self.owner = owner
        self.__balance = balance  # Private attribute

    def deposit(self, amount):
        if amount > 0:
            self.__balance += amount

    def get_balance(self):
        return self.__balance

account = BankAccount("Bryan", 1000)
account.deposit(500)
print(account.get_balance())  # 1500
```

Inheritance

Create new classes based on existing ones

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

    def play(self):
        return f"Playing the {self.name}"

class Guitar(Instrument):
    def __init__(self, name, strings):
        super().__init__(name)
        self.strings = strings

    def strum(self):
        return f"Strumming {self.strings} strings"

guitar = Guitar("acoustic", 6)
print(guitar.play())   # Playing the acoustic
```

Polymorphism

Same method name, different behavior

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

class Rectangle(Shape):
    def __init__(self, width, height):
        self.width = width
        self.height = height

    def area(self):
        return self.width * self.height

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

    def area(self):
        return 3.14 * self.radius ** 2

# Use shapes uniformly
shapes = [Rectangle(5, 3), Circle(4)]
for shape in shapes:
```

Best Practices

- **Single Responsibility**: Each class should do one thing well
- **Use Encapsulation**: Keep data private, provide methods for access
- **Clear Naming**: Use descriptive names for classes and methods
- **Initialize Properly**: Always set up attributes in `__init__`

Summary

The Four Pillars:

- **Encapsulation**: Hide data, control access
- **Abstraction**: Simplify complexity
- **Inheritance**: Reuse code through parent classes
- **Polymorphism**: Same interface, different behavior