Notes on Object-Oriented Programming (OOP) in Python

Object-Oriented Programming (OOP) is a programming paradigm that uses "objects" to model real-world entities. These objects encapsulate **data** (attributes) and **behavior** (methods), promoting modularity, reusability, and scalability in code.

Key Concepts of OOP

1. Class

- A class is a blueprint or template for creating objects.
- It defines the properties (attributes) and behaviors (methods) an object can have.

Example:

```python
class Car:
    def __init__(self, brand, color):
        self.brand = brand  # Attribute
        self.color = color  # Attribute

    def drive(self):
        return f"The {self.color} {self.brand} is driving!"  # Method
```

---

2. Object

- An object is an instance of a class.
- It is created using the class blueprint and represents a specific entity.

Example:

```python
my_car = Car("Toyota", "Red")
print(my_car.drive())
# Output: The Red Toyota is driving!
```

3. Encapsulation

- Encapsulation is the practice of bundling data (attributes) and methods (functions) into a single unit (class).
- It restricts direct access to some components, providing control over data.

Example:

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

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

    def get_balance(self):
        return self.__balance

account = BankAccount(1000)
account.deposit(500)
print(account.get_balance())  # Output: 1500
```

---

4. Inheritance

- Inheritance allows a class (child class) to acquire properties and behaviors from another class (parent class).
- Promotes code reuse and logical hierarchy.

Example:

```python
class Animal:
    def speak(self):
        return "I make a sound."

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

my_dog = Dog()
print(my_dog.speak())  # Output: Woof!
```

5. Polymorphism

- Polymorphism means "many forms."
- It allows methods in different classes to have the same name but behave differently.

Example:

```python
class Bird:
    def fly(self):
        return "I can fly."

class Penguin(Bird):
    def fly(self):
        return "I cannot fly, but I can swim!"

bird = Bird()
penguin = Penguin()
print(bird.fly())      # Output: I can fly.
print(penguin.fly())   # Output: I cannot fly, but I can swim!
```

---

6. Abstraction

- Abstraction hides implementation details and shows only essential features.
- Achieved using abstract classes and methods (in Python, the `abc` module is used).

Example:

```python
class Shape(ABC):
    @abstractmethod
    def area(self):
        pass

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

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

circle = Circle(5)
print(circle.area())  # Output: 78.5
```

Benefits of OOP

Modularity: Code is organized into classes, making it easier to manage

Reusability: Use inheritance and polymorphism to reuse existing code.

Scalability: Classes and objects make large systems easier to build and extend.

Maintainability: Encapsulation ensures that changes to one part of code don't impact others.

Summary of OOP Pillars

Pillar	***Description***	***Key Points***
Encapsulation	Bundles data and methods into a class	Ensures controlled data access
Inheritance	Enables code reuse by inheriting properties	Promotes logical relationships
Polymorphism	Allows one interface, multiple implementations	Simplifies method extensions
Abstraction	Hides unnecessary details	Focuses on essential functionality

---

Practice Challenge

Write a Python program that demonstrates all four OOP principles using a **School** system where:

- There is a parent class `Person` with child classes `Student` and `Teacher`.
- The `Teacher` class has a method to calculate the salary based on teaching hours.
- The `Student` class has a method to calculate the grade.

Conclusion

OOP is a powerful paradigm that simplifies code design and fosters collaboration. By mastering these principles, you'll be equipped to write scalable, reusable, and maintainable code for complex systems.