Nguyen Cong Phat
I'm a somophore in Computer Science at UIT – VNUHCM with a data-oriented mindset and a strong enthusiasm for learning and innovation

Python OOP Programming

Master Python Object-Oriented Programming (OOP) with Hands-on Projects

Vietnamese Full Name: Nguyen Cong Phat

My portfolio website: https://my-portfolio-website-gamma-lemon.vercel.app/

My github profile: https://github.com/paht2005

Table of Contents

A. Introduction to Object-Oriented Programming (OOP) in Python 2
1. What is a Class? 2
2. What is an Object? 2
3. Attributes and Methods 3
4. Key Pillars of OOP 3
a. Encapsulation 3
b. Inheritance 3
c. Polymorphism 4
d. Abstraction 4
B. Hands-on Python OOP Mini-Projects 4
1. Animal Soundboard 4
2. Simple Bank System 4
3. Employee Management System 5
4. Smart Inventory System 5
5. Library Management System 6
6. Mini ATM Machine 6
7. Secure User Vault 6

A. Introduction to Object-Oriented Programming (OOP) in Python

Let's start with the basics of OOP.

1. What is a Class?

In Python, a **class** is like a blueprint or a template for creating objects. Think of it as a design for something real-world. For example, if you're building a program to manage vehicles, you can define a class called **Car** that contains common features like **brand** and **model.**

2. What is an Object?

- An **object** is a real "thing" created from a class. It's an instance of that class. Once you define the **Car** class, you can create many individual cars (objects) from that blueprint.

- Here's a simple example of a **Car** class and how to create an object:

```
class Car:
  def __init__(self, brand, model):
    self.brand = brand
    self.model = model

# Creating an object (an instance of the Car class) my_car =
Car("Toyota", "Corolla")
```

In this example, **my_car** is an object of the **Car** class. It has its own **brand** set to "Toyota" and **model** set to "Corolla".

3. Attributes and Methods

- Inside classes, we define **attributes** (which are like variables) and **methods** (which are like functions).

- **Attributes**: These describe the characteristics or data of an object. For **my_car**, **brand** and **model** are attributes.
- **Methods**: These define the actions or behaviors an object can perform.

- For instance, we can add a **drive** method to our **Car** class:

```
class Car:
  def __init__(self, brand, model):
    self.brand = brand
    self.model = model
  def drive(self):
    print(f"{self.brand} {self.model} is driving.")

my_car = Car("Toyota", "Corolla")
my_car.drive()

# Output: Toyota Corolla is driving.
```

4. Key Pillars of OOP

OOP is built upon several core concepts:

a. Encapsulation

This is the process of bundling data (attributes) and the methods (functions) that operate on that data into a single unit (the class). It helps in hiding the internal details of how an object works and only showing what's necessary to the user. In our **Car** example, **brand**, **model**, and **drive()** are all encapsulated within the **Car** class.

b. Inheritance

This allows a new class (called a subclass or derived class) to acquire or share properties and behaviors from an existing class (called a base class or parent class). This promotes reusability. For example, a **Dog** class could inherit from a general **Animal** class, gaining common animal traits while also having dog-specific properties.

c. Polymorphism

This refers to the ability to process objects differently depending on their data type or class. In simpler terms, it means "many forms." A common example is having different animal types (Dog, Cat) respond differently when you tell them all to **make_sound()**. Each animal will make its own distinct sound.

d. Abstraction

This means hiding the complex implementation details and showing only the essential functionality to the user. For instance, when you press the accelerator in a car, you don't need to know the intricate engine mechanics; you just know it makes the car go faster. The complexity is abstracted away.

Mastering these concepts will help you write more scalable, organized, and maintainable code, especially in larger projects.

B. Hands-on Python OOP Mini-Projects

To help you solidify your understanding, I've prepared a set of mini-projects. Each project illustrates different OOP concepts and can be run independently.

This Github Project link (source code): https://github.com/paht2005/some_Object-Oriented_miniProjects

1. Animal Soundboard

- **Goal:** Understand **Inheritance** and **Polymorphism.**
- **Description**: This project creates a simple "soundboard" where different animals can make their unique sounds. It demonstrates how child classes (like **Dog**, **Cat**) inherit from a parent class (**Animal**) and override the **make_sound()** method to produce their specific behavior.
- **Key OOP Concepts**:
  - **Inheritance**: **Dog**, **Cat**, **Cow**, **Duck** all inherit from **Animal**.
  - **Polymorphism**: The **make_sound()** method is defined in the parent class and overridden in the child classes, allowing each animal object to produce its distinct sound when the same method is called.
  - **Encapsulation**: Each animal class encapsulates its own sound-making behavior.
- **How to Run**: Execute the **AnimalSound_main.py** file and follow the console instructions. You can add various animals to the soundboard and then play all their sounds.

2. Simple Bank System

- **Goal**: Practice managing object states and handling transactions.
- **Description**: A basic console-based banking application allowing users to create accounts, deposit, withdraw, and view their balance.

- **Key OOP Concepts**:
    - **Class BankAccount**: Represents a bank account, encapsulating attributes like **name** and **balance**, and behaviors such as **deposit** and **withdraw**.
    - **Encapsulation**: The account balance is managed through **deposit** and **withdraw** methods, ensuring data integrity.
- **How to Run**: Run **BankAccount_main.py**. You'll have options to create a new account or log into an existing one.

3. Employee Management System

- **Goal:** Deepen understanding of inheritance and polymorphism in managing different types of objects.
- **Description**: This system allows managing different types of employees (general staff, managers, developers), where each type has its own bonus calculation and information display.
- **Key OOP Concepts:**
    - **Inheritance: Manager** and **Developer** classes inherit from the base **Employee** class.
    - **Polymorphism:** The **calc_bonus()** and **display_info()** methods are overridden in subclasses to provide specific logic for each employee role.
    - **Encapsulation**: Each employee object encapsulates its own data and behavior.
- **How to Run**: Execute **EmployeeManagement_main.py.** You can add employees of various roles and view their detailed information along with their bonuses.

4. Smart Inventory System

- **Goal**: Get familiar with **Class Method** and **Static Method**, and managing collections of objects.
- **Description**: An inventory management system that allows adding products, listing them, processing sales, calculating discounts, and viewing the total inventory count.
- **Key OOP Concepts**:
    - **Class Product**: Represents a product in the inventory.
    - **Class Method (display-total-inventory)**: This method accesses a class-level attribute **(_total_inventory**) to display the total number of items in stock, without needing a specific object instance.
    - **Static Method (apply-discount)**: This method performs a general function (calculating a discount) that doesn't rely on the state of a specific **Product** object or even the class itself.
    - **Encapsulation**: Each product manages its own quantity and price.
- **How to Run**: Run **Smart-Inventory_main.py**. You can add products, sell them, and check the overall inventory.

5. Library Management System

- **Goal**: Understand how to manage object states and interactions between objects.
- **Description**: A library application that allows adding books, listing them, checking out and returning books, and managing their availability status.
- **Key OOP Concepts**:
    - **Class Publication**: Represents a publication (book) with attributes like **name**, **writer**, and **is_checked_out** status.
    - **Class Archive**: Manages a collection of **Publication** objects.
    - **Encapsulation**: Each book object encapsulates its **is_checked_out** state.
- **How to Run**: Execute **LibraryManagement_main.py**. You can add books, view the list, and simulate borrowing and returning them.

6. Mini ATM Machine

- **Goal**: Enhance skills in encapsulation, user account management, and transactions.
- **Description**: Simulates a simple ATM machine with functions for logging in, viewing balance, depositing/withdrawing funds, and changing PIN.
- **Key OOP Concepts:**
    - **Class UserWallet:** Represents a user account with encapsulated attributes like **__pin** (private) and **__funds** (private).
    - **Class SimpleBankATM:** Controls the ATM operations, including account creation and authentication.
    - **Encapsulation:** Sensitive data like PIN and balance are made "private" (**_pin, __funds**) to protect them, accessible only through public methods.
- **How to Run:** Run **miniATM-machine_main.py**. You can create new accounts and perform ATM transactions.

7. Secure User Vault

- **Goal**: Learn about protected and private attributes, and regular expressions for validation.
- **Description**: A straightforward application to manage user profiles, allowing registration, viewing, and updating email information.
- **Key OOP Concepts**:
    - **Class AccountProfile**: Represents a user's profile.
    - **Protected Attribute (_email)**: While Python doesn't have true private attributes, **_email** by convention indicates it should not be accessed directly from outside the class.
    - **Private Attribute (__password)**: Prefixed with double underscores, **__password** is name-mangled, making it harder to access directly and emphasizing its private nature, ensuring password security.
    - **Encapsulation**: User data is encapsulated within **AccountProfile** objects.

- **How to Run**: Execute **UserVault_main.py**. You can register new users, view profiles, and update email addresses.

I hope these projects provide you with a clearer understanding of Object-Oriented Programming in Python and how to apply it in real-world scenarios. Feel free to run each project, examine the source code, and even try to extend them to further hone your skills!