**BÀI TẬP VỀ NHÀ – LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG VỚI PYTHON**

**1. Phân biệt class attribute và instance attribute**

* **Class attribute** (thuộc tính lớp):
  + Được khai báo trực tiếp bên trong thân lớp, không nằm trong các phương thức.
  + Mỗi đối tượng của lớp đều chia sẻ chung giá trị của thuộc tính này.
  + Ví dụ:
  + class SinhVien:
  + truong = "Đại học ABC" # Class attribute

Nếu ta thay đổi SinhVien.truong, tất cả các đối tượng của lớp SinhVien (nếu chưa ghi đè) cũng sẽ có giá trị mới.

* **Instance attribute** (thuộc tính thể hiện):
  + Được khai báo và khởi tạo trong phương thức khởi tạo \_\_init\_\_.
  + Mỗi đối tượng được tạo ra có bộ thuộc tính riêng, không bị ảnh hưởng bởi các đối tượng khác.
  + Ví dụ:
  + class SinhVien:
  + def \_\_init\_\_(self, ten, tuoi):
  + self.ten = ten # Instance attribute
  + self.tuoi = tuoi # Instance attribute

**2. So sánh cách thực thi access modifiers giữa Python với C#/Java**

* **Trong Python:**
  + Python không có cơ chế access modifiers (public, protected, private) cứng nhắc như C# hay Java.
  + Ta chỉ sử dụng quy ước:
    - **Public:** Các thuộc tính/method thông thường (ví dụ: ten, tuoi).
    - **Protected:** Đặt tên với dấu gạch dưới ở đầu (ví dụ: \_diem), báo hiệu đây là thuộc tính "được bảo vệ" (không nên dùng bên ngoài, nhưng vẫn có thể truy cập).
    - **Private:** Đặt tên với hai dấu gạch dưới (ví dụ: \_\_ma\_sv), Python sẽ thực hiện "name mangling" để hạn chế truy cập trực tiếp từ bên ngoài.
* **Trong C#/Java:**
  + Các ngôn ngữ này cung cấp các access modifiers rõ ràng:
    - **public:** Cho phép truy cập từ mọi nơi.
    - **protected:** Chỉ được truy cập từ bên trong lớp và lớp con.
    - **private:** Chỉ được truy cập trong nội bộ lớp.
  + Việc kiểm tra các access modifiers được thực hiện tại thời điểm biên dịch, do đó an toàn hơn về mặt đóng gói dữ liệu.

**3. Ý nghĩa và cách sử dụng các decorator: @property, @classmethod, @staticmethod**

* **@property:**
  + Dùng để biến một phương thức thành thuộc tính.
  + Giúp việc truy cập thuộc tính trở nên trực quan hơn, đồng thời có thể thêm logic kiểm tra khi lấy hoặc gán giá trị.
  + Ví dụ:
  + class HinhTron:
  + def \_\_init\_\_(self, ban\_kinh):
  + self.\_ban\_kinh = ban\_kinh
  + @property
  + def ban\_kinh(self):
  + return self.\_ban\_kinh
  + @ban\_kinh.setter
  + def ban\_kinh(self, value):
  + if value > 0:
  + self.\_ban\_kinh = value
  + else:
  + raise ValueError("Bán kính phải > 0")
* **@classmethod:**
  + Phương thức có tham số đầu tiên là cls đại diện cho lớp.
  + Có thể truy cập và thay đổi các thuộc tính của lớp.
  + Thường dùng để tạo các factory method (hàm tạo đối tượng theo cách khác).
  + Ví dụ:
  + class SinhVien:
  + @classmethod
  + def khoi\_tao\_mac\_dinh(cls):
  + return cls("Không tên", 18)
* **@staticmethod:**
  + Phương thức không nhận tham số đặc biệt nào (không có self hay cls).
  + Hoạt động như một hàm thông thường, chỉ đặt trong lớp để tổ chức code.
  + Ví dụ:
  + class ToanHoc:
  + @staticmethod
  + def cong(a, b):
  + return a + b

**4. So sánh @classmethod và @staticmethod**

* **@classmethod:**
  + Tham số đầu tiên là cls (đại diện cho lớp), do đó có thể truy cập và thay đổi các thuộc tính của lớp.
  + Thường dùng khi cần tạo ra các phương thức khởi tạo (factory method) hoặc cần làm việc với dữ liệu chung của lớp.
* **@staticmethod:**
  + Không nhận tham số đặc biệt nào, không phụ thuộc vào trạng thái của lớp hay đối tượng.
  + Thường dùng cho các hàm tiện ích, mà logic của hàm không liên quan đến lớp hay instance cụ thể.

**5. So sánh kế thừa và đa hình giữa Python với C#/Java**

* **Tương đồng:**
  + Cả Python, C# và Java đều cho phép kế thừa (inheritance), tức là lớp con có thể kế thừa thuộc tính và phương thức của lớp cha.
  + Đều hỗ trợ đa hình (polymorphism): lớp con có thể ghi đè các phương thức của lớp cha để cung cấp hành vi riêng.
* **Khác biệt:**
  + **Python:**
    - Là ngôn ngữ động, không cần khai báo kiểu dữ liệu.
    - Cho phép đa thừa kế (một lớp có thể kế thừa từ nhiều lớp cha).
    - Kiểm soát truy cập dựa trên quy ước (không có cơ chế cứng nhắc như private/protected trong C#/Java).
  + **C#/Java:**
    - Là ngôn ngữ tĩnh, cần khai báo kiểu dữ liệu rõ ràng.
    - Java chỉ hỗ trợ kế thừa đơn (một lớp chỉ có thể kế thừa từ một lớp cha) nhưng có thể thực hiện đa thừa kế thông qua interface; C# cũng tương tự.
    - Hỗ trợ access modifiers cứng nhắc, giúp kiểm soát truy cập và bảo vệ dữ liệu của lớp.