📝 Clock App - Xây dựng Ứng dụng Đồng Hồ với Python

✍️ **Tác giả: Đặng Kim Thi**

🔥 Giải thích cách làm từ tổng quan đến chi tiết

1️⃣ Tổng quan về bài toán

Bài toán yêu cầu xây dựng một ứng dụng **đồng hồ số có thể hiển thị và thay đổi thời gian** bằng cách nhập giá trị vào giao diện người dùng (GUI). Để đạt được điều này, ta sử dụng:

- **Tkinter**: Thư viện tích hợp sẵn trong Python để tạo giao diện đồ họa.
- **Lập trình hướng đối tượng (OOP)**: Chia nhỏ ứng dụng thành các lớp có nhiệm vụ riêng biệt, giúp mã nguồn dễ bảo trì.

2️⃣ Kiến trúc chương trình

Để đảm bảo **mã nguồn gọn gàng, dễ hiểu**, ta chia chương trình thành **2 tệp chính**:

Tệp	Chức năng
`StillClock.py`	Tạo và vẽ đồng hồ số. Chứa các phương thức để cập nhật giờ, phút, giây.
`DisplayClock.py`	Giao diện người dùng với các ô nhập liệu để chỉnh thời gian của đồng hồ.

3️⃣ Triển khai chi tiết từng phần

📌 Phần 1: `StillClock.py` - Xây dựng lớp đồng hồ

🔷 **Nhiệm vụ của lớp `StillClock`:**

- **Vẽ mặt đồng hồ** bằng `Canvas`.
- **Hiển thị kim giờ, kim phút, kim giây** dựa trên giá trị thời gian hiện tại.
- **Hỗ trợ thay đổi thời gian** khi có lệnh từ `DisplayClock.py`.

💡 **Cách triển khai**

1. **Tạo lớp `StillClock`** kế thừa từ `Canvas` (bảng vẽ trong Tkinter).
2. **Lấy thời gian hệ thống** (sử dụng `datetime.now()`).
3. **Vẽ mặt đồng hồ và kim** dựa trên công thức **lượng giác** (tọa độ cực chuyển sang tọa độ Descartes).
4. **Thêm các phương thức `setHour()`, `setMinute()`, `setSecond()`** để cập nhật thời gian.

📌 **Cách hoạt động của `StillClock`:**

- Lớp `StillClock` mở rộng từ `Canvas`, nghĩa là `StillClock` hoạt động như một `Canvas`.

- Hàm khởi tạo của `StillClock` sẽ gọi hàm khởi tạo của `Canvas`, sau đó khởi tạo các biến `hour`, `minute`, `second` bằng cách gọi `setCurrentTime()`.
- Các biến `hour`, `minute`, `second` có các phương thức getter và setter để lấy và cập nhật giá trị của chúng. Khi cập nhật, hàm `drawClock()` sẽ được gọi để vẽ lại đồng hồ.
- Phương thức `setCurrentTime()` lấy thời gian hiện tại bằng cách gọi `datetime.now()`, sau đó cập nhật lại kim đồng hồ.
- Hàm `drawClock()` lấy kích thước của `Canvas`, xác định kích thước của kim giờ, phút, giây và vẽ đồng hồ bằng các phương thức vẽ của `Canvas`.

📌 **Công thức tính tọa độ đầu cuối của kim đồng hồ**:

- Để tính vị trí của kim đồng hồ, ta dùng công thức chuyển từ **góc quay sang tọa độ x, y** trong mặt phẳng Descartes:

  [x = x_0 + r \times \cos(\theta)] [y = y_0 + r \times \sin(\theta)]

  Với:

  - (x_0, y_0) là tâm đồng hồ (giả sử (100,100)).
  - (r) là độ dài kim.
  - (\theta) là góc quay của kim (theo đơn vị radian).
  - (\theta = \frac{\text{số đơn vị} \times 360}{tổng đơn vị} - 90) (để xoay đúng hướng, cần trừ đi 90 độ).

  Ví dụ:

  - Kim giờ: (\theta = \frac{\text{giờ} \times 30}{12} - 90)
  - Kim phút: (\theta = \frac{\text{phút} \times 6}{60} - 90)
  - Kim giây: (\theta = \frac{\text{giây} \times 6}{60} - 90)

🔗 **Tham khảo thêm về hệ tọa độ và cách chuyển đổi tại**: Wikipedia - Hệ tọa độ

📌 Phần 2: `DisplayClock.py` - Xây dựng giao diện nhập thời gian

🔷 **Nhiệm vụ của `DisplayClock`**:

- **Tạo cửa sổ Tkinter** để chứa giao diện đồng hồ.
- **Tạo các ô nhập liệu** cho **giờ, phút, giây**.
- **Có nút `Set Time`** để cập nhật lại đồng hồ.

📌 **Cách triển khai**:

1. **Tạo cửa sổ Tkinter** (`window = Tk()`).
2. **Gọi `StillClock` để vẽ đồng hồ** (`self.clock = StillClock(window)`).
3. **Thêm ô nhập liệu** (`Entry`) để nhập giờ, phút, giây.
4. **Thêm nút `Set Time`**, khi nhấn vào sẽ gọi hàm `setNewTime()` để cập nhật đồng hồ.

4️⃣ Chạy chương trình

Để chạy chương trình, làm theo các bước sau:

1️⃣ Lưu các file

- `StillClock.py`
- `DisplayClock.py`

2️⃣ Chạy lệnh

```
python DisplayClock.py
```

⏳ Ứng dụng sẽ mở giao diện đồng hồ, cho phép chỉnh sửa thời gian.

5️⃣ Kết luận

🔷 Cách tiếp cận này có lợi ích gì?

Ưu điểm	Lợi ích
Tổ chức mã tốt	Mỗi lớp có trách nhiệm riêng, dễ bảo trì.
Sử dụng OOP	Tận dụng kế thừa và đóng gói để quản lý đồng hồ dễ dàng.
Giao diện đẹp, trực quan	Tkinter giúp xây dựng ứng dụng dễ dàng.

💡 **Tóm lại**, chương trình được xây dựng theo hướng **mô-đun hóa** với cách tổ chức hợp lý, giúp dễ mở rộng và bảo trì.