Task 1: Algorithmic Trading Adventure

Scenario : Alex, a budding programmer and finance enthusiast, embarks on an algorithmic trading adventure with a budget of $5000. Their mission? To develop a tool that leverages Python to make informed decisions in the stock market, using a class-based approach for flexibility.

Task Steps:

Initializing Class: Create a class to encapsulate the trading strategy, allowing initialization with parameters such as symbol, from date, and to date. **Example: class_Name("AAPL","2018-01-01","2023-12-31")**

Setting the Stage: Install the yfinance library to access historical market data.

Data Acquisition: Download historical data for the specified symbol within the provided date range.

Data Cleanup: Filter out duplicate data points and handle NaN values by forward filling.

Analytical Insights: Compute the moving averages for 50 and 200 days.

Golden Opportunity: Identify the golden cross, signaling a bullish trend, and take a buying position.

Investment Strategy: Determine the maximum quantity of shares to purchase within the $5000 budget.

Timely Actions: When the golden cross reverses, sell the position and close the trade. When you are in a position you can't buy other stock.

Final Touches: Forcefully close the position on the last row if a position is still open.

Evaluation: Calculate profits or losses to assess the success of the trading strategy.

Through this adventure, Alex not only hones their programming skills but also gains valuable insights into the dynamic world of finance, all while managing their investments wisely

Task 2: Samsung Phone Advisor

Scenario: Imagine you are building a **smart assistant** for a tech review platform. The platform's goal is to help users make informed decisions when buying Samsung smartphones. Users want both **detailed specs** and **natural-language reviews or recommendations**.

Scenario Flow:

1. **Data Collection:**
    - Your system first **scrapes Samsung phone data** (20–30 models) from GSMArena
    - Scraped data is **stored in PostgreSQL**, including model name, release date, display, battery, camera, RAM, storage, and price.
2. **User Interaction:**
    - Users ask questions naturally, such as:
        - *"What are the specs of Samsung Galaxy S23 Ultra?"*
        - *"Compare Galaxy S23 Ultra and S22 Ultra for photography."*
        - *"Which Samsung phone has the best battery under $1000?"*
3. **Unified RAG + Multi-Agent System:**

    Step 1 – RAG Module:
    - Retrieves **structured specifications** from PostgreSQL
    - Answers **direct factual questions** about phones
4. **Step 2 – Multi-Agent System:**
    - **Agent 1 – Data Extractor:** Pulls relevant phone data from PostgreSQL based on the query

- **Agent 2 – Review Generator:** Generates **comparative analysis or recommendations** in natural language

5. **Response Composition:**
    - The system **unifies outputs** from RAG and multi-agent agents to deliver a **complete answer**

Example Answer:

Samsung Galaxy S23 Ultra has a 6.8" AMOLED display, 5000mAh battery, and 200MP rear camera.
Compared to Galaxy S22 Ultra, S23 Ultra has better camera performance and battery life, making it the recommended choice for photography and long usage.

6. **API Access via FastAPI:**
    - Users send natural-language queries to a single endpoint:
        - `/ask` → returns specifications, reviews, or comparisons depending on the query

Input Example:

```
{ "question": "Compare Samsung Galaxy S23 Ultra and S22 Ultra" }
```

Output Example:

```
{ "answer": "Samsung Galaxy S23 Ultra has better camera and battery life than S22 Ultra. Display is similar. Overall, S23 Ultra is recommended for photography and long usage." }
```

Result:
The user has a **single, unified system**: they ask a question in natural language, and the system automatically retrieves **specifications** and generates **reviews or recommendations**, leveraging the **RAG module and multi-agent reasoning**, all powered by **PostgreSQL** and served via **FastAPI**.

Submission Instructions

1. **Must submit both tasks**
2. Submit both tasks in a **public GitHub repository** with a clear README explaining how to run each project.
3. Include a **Google Drive link** to a short **video demonstration** showing both tasks in action.
4. Ensure the repository contains all required files, scripts, and any dataset needed to run the projects.
5. Verify that both tasks run successfully and independently before submission.
6. Submit **both the GitHub link and the video link** together for evaluation.