SteinnLabs – Backend Intern Assignment

Sequential Multi-Agent Financial Computation System (Python)

Instructions

You are required to build a **simple sequential multi-agent system** in Python to perform financial calculations. The focus of this assignment is on **logic, correctness, and agent orchestration**, not UI or deployment.

Mandatory Instructions

1. **Record a video explaining each step** of your implementation before progressing to next step.
2. **Rename each video file** according to the step number (e.g., Step_0.mp4, Step_1.mp4, etc.).
3. **Create a new folder** named exactly as:

   Python_FirstName_LastName
4. Inside this folder, include:
   - A **ZIP file containing all your Python code**
   - All the **step-wise explanation videos**

Additional Notes

- GitHub submission is **optional** (You can add github link in the main.py).
- You may use **CrewAI, LangChain, LangGraph, or any open-source Python agent framework**.
- The mathematical logic **must be implemented in Python code**, not delegated to the LLM.
- Agents must run **sequentially** (not parallel).

Step-Wise Task Breakdown

Step 0: Environment Setup & Agent Framework Initialization

Task

- Set up a Python virtual environment.
- Choose and install an agent framework (CrewAI / LangChain / LangGraph / similar).
- Create a minimal project structure.
- Implement a simple test agent that:
    - Accepts an input
    - Returns a modified output

Input Example

Input: 5

Expected Output

Output: 10

(Example: value doubled by the agent)

Step 1: Agent Design & Logic Explanation

Task

Implement **three agents** with clear responsibilities:

1. **ArithmeticAgent**
    - Performs:
        - Addition
        - Subtraction
        - Division
    - Must handle invalid cases (e.g., division by zero)

2. **PercentageAgent**
    - Calculates percentage values
    - Example: 10% of 200 → 20
3. **AuditAgent**
    - Validates intermediate results
    - Logs each step
    - Rejects invalid states (negative values, invalid operations)

Each agent must:

- Have a single responsibility
- Accept structured input
- Return structured output

You must clearly explain:

- What the agent does
- Why it exists
- What it does **not** handle

Input Example:

```
{
  "value": 200,
  "percentage": 10
}
```

Expected Output:

```
{
  "value": 20,
  "agent": "PercentageAgent",
  "status": "success"
}
```

Step 2: Sequential Agent Orchestration

Task

- Implement a **Coordinator / Orchestrator** that:
    - Executes agents **one after another**
    - Passes output from one agent as input to the next
- The coordinator:
    - Must not perform any mathematical operations
    - Must stop execution if the AuditAgent fails

Execution Flow

Initial Value

→ PercentageAgent

→ ArithmeticAgent

→ AuditAgent

Input Example

Initial Value: 1000

Percentage: 10%

Expected Output

Audit Passed

Final Value: 1100

Step 3: Compound Interest Without Formula

Task

Compute **compound interest over multiple years** using **sequential agent execution only**.

Given

- Principal (P)

- Annual Interest Rate (R%)
- Time (T years)

Rules

- You **must not** use:
    - Compound interest formula
    - Single-step calculations
- Interest must be calculated **year by year**.

Required Yearly Flow

For each year:

1. PercentageAgent → calculates interest
2. ArithmeticAgent → adds interest to balance
3. AuditAgent → validates and logs result

Agents must remain **stateless**.
The coordinator manages the loop and state passing.

Input Example

Principal: 1000

Rate: 10%

Time: 3 years

Expected Output

Year 1 → 1100

Year 2 → 1210

Year 3 → 1331

Logs must clearly show:

- Agent name
- Input
- Output
- Year number

Submission Folder Structure

Python_FirstName_LastName/

|

|— code.zip

| └— (All Python source code)

|

|— Step_0.mp4

|— Step_1.mp4

|— Step_2.mp4

└— Step_3.mp4