# *Assignment*

Lab 13 – Code Refactoring: Improving Legacy Code with AI

**HALLTICKET NUMBER :** 2503A52416

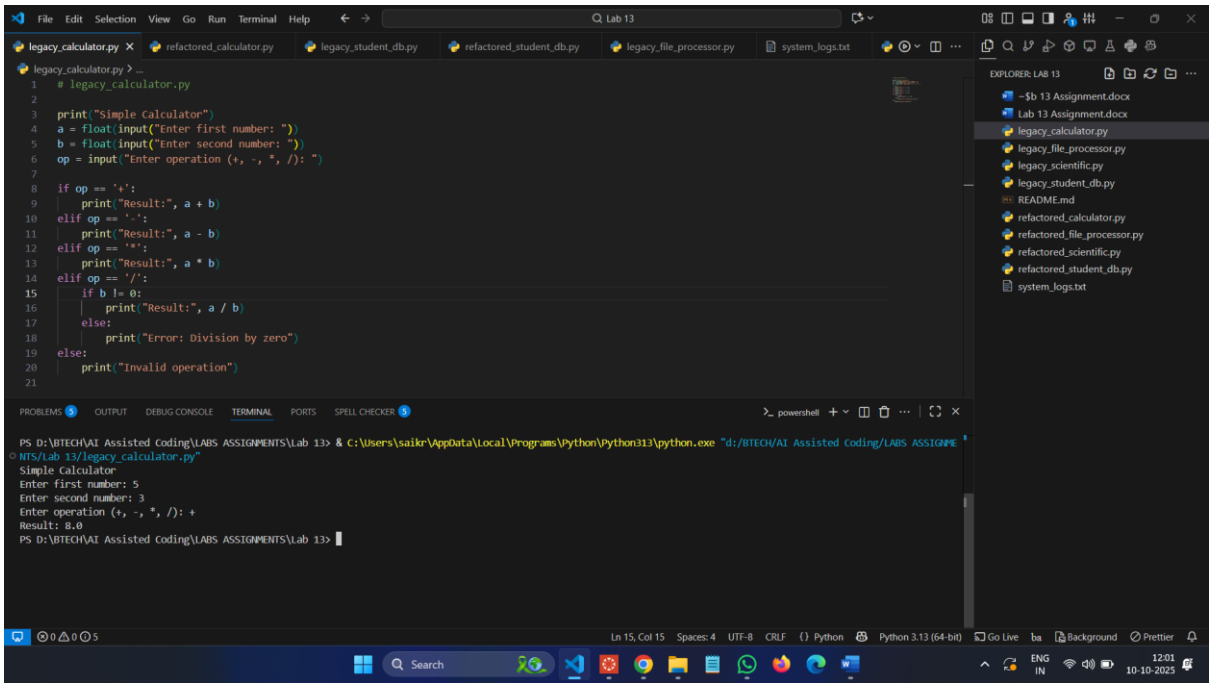
**Task 1: Refactoring a Legacy Calculator Script**

**Scenario:**

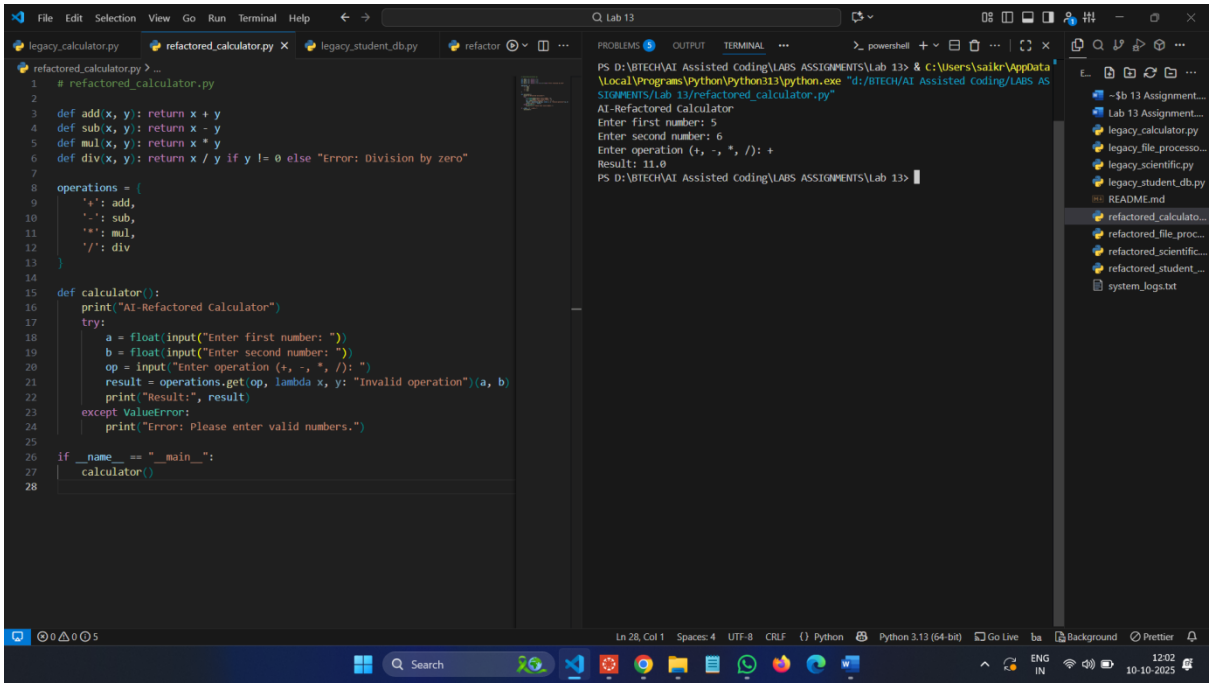
A university has a legacy Python script for a basic calculator that uses long, repetitive if-else statements for each operation. The code is difficult to maintain.

- Upload the calculator script to a GitHub repository.
- Use **GitHub Copilot** to suggest a more modular and cleaner version (e.g., functions, dictionary-based mapping).
- Compare the AI-suggested refactoring with the original code and document improvements

**Legacy Code &Output :**



## Refactored Code &Output :



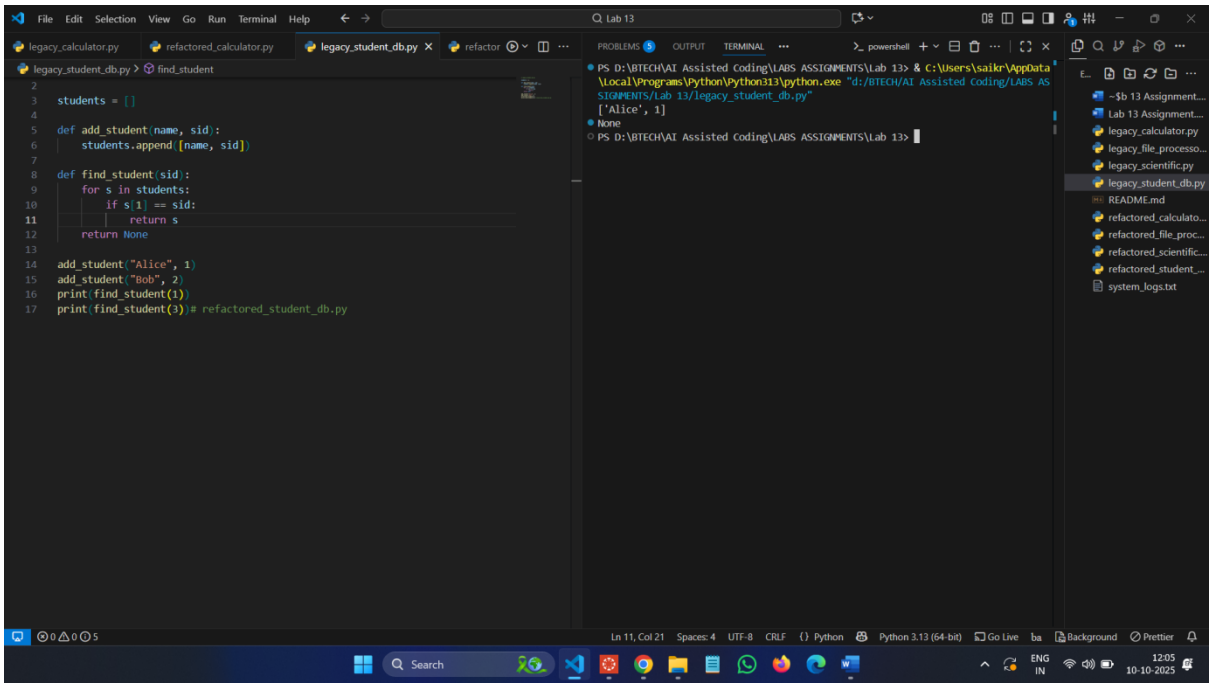

## Task 2: Modernizing a Student Database Program

**Scenario:**

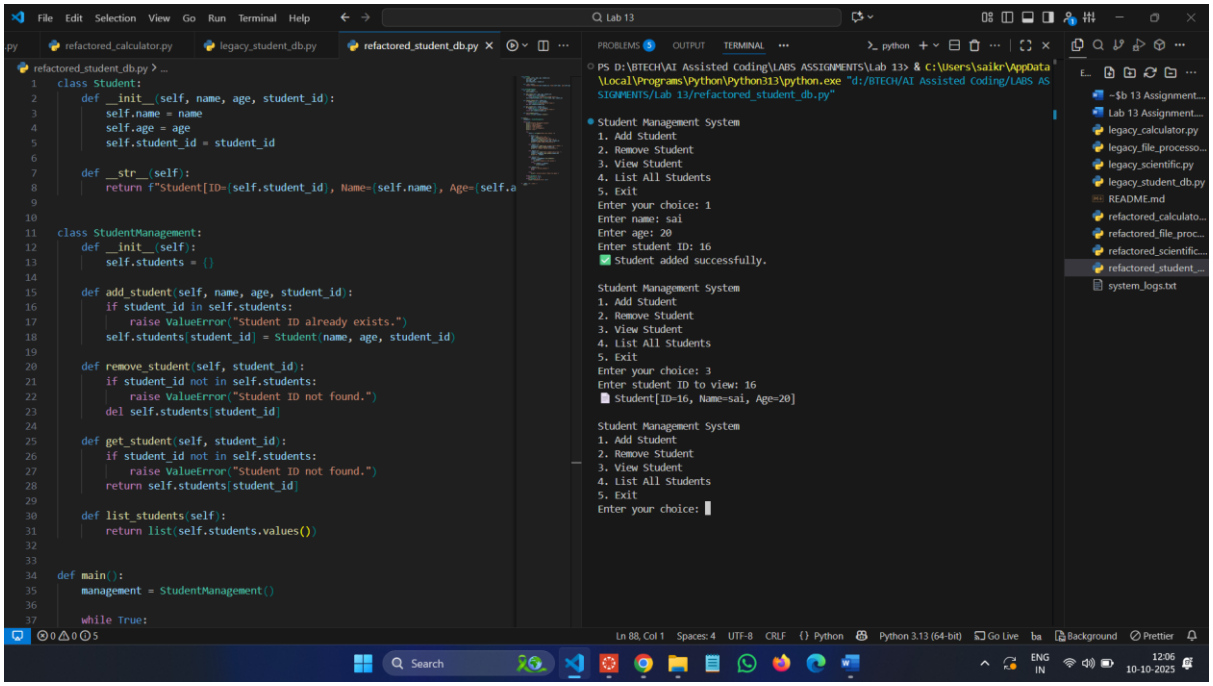
An old student management program uses procedural code with global
variables and no error handling. The program frequently crashes when handling incorrect inputs.

• Push the legacy code into your GitHub repo.
• Ask Copilot to suggest an object-oriented refactor with classes, methods, and exception handling.
• Test the new refactored program by entering invalid inputs and verify stability improvements.

# Legacy Code &Output :




# Refactored Code &Output :



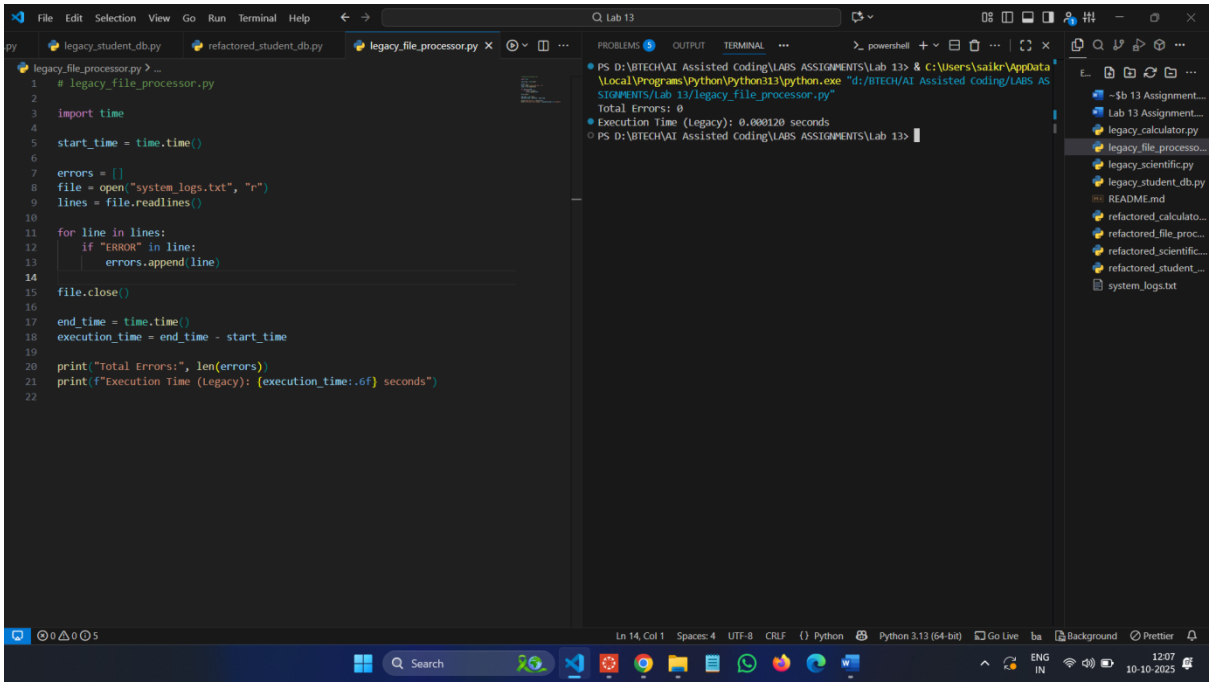

**Task 3: Optimizing Performance in File Processing**

**Scenario:**

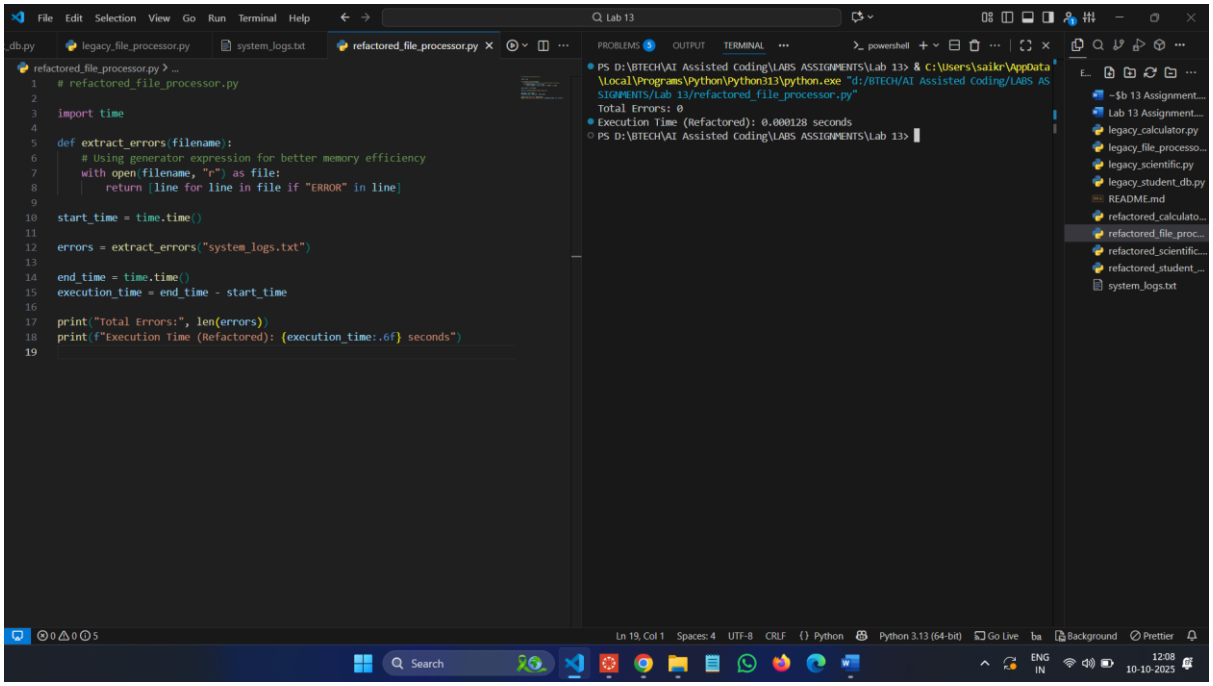
A company's file-processing script reads large log files line by line using inefficient loops, causing delays.

• Commit the original file-processing script to GitHub.
• Use Copilot suggestions to replace inefficient loops with more optimized approaches (e.g., list comprehension, built-in functions, generators).
• Compare the execution time of legacy vs. refactored versions and document the performance gains.

**Legacy Code &Output :**




# Refactored Code &Output :



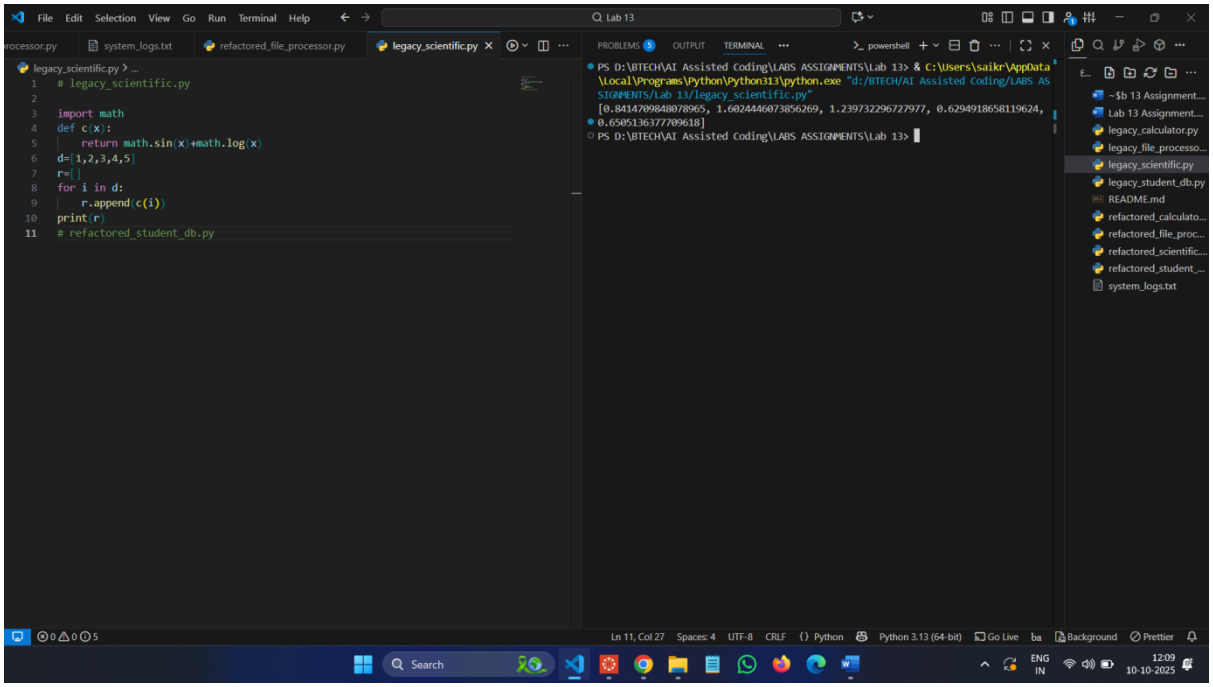

## Task 4: Enhancing Readability and Documentation

**Scenario:**

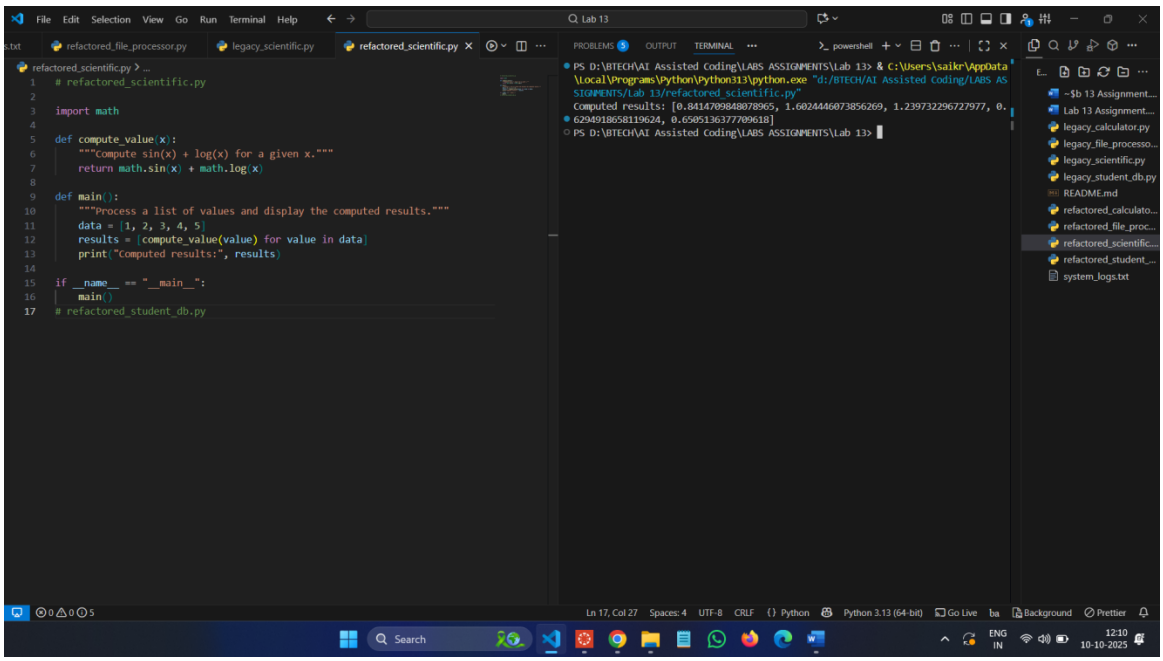
A research group has shared a scientific computation script with minimal comments, inconsistent naming, and poor readability.

• Upload the legacy code to GitHub.
• Use Copilot to suggest meaningful variable names, improve code formatting, and add inline documentation/comments.
• Generate an AI-assisted README.md file for the project explaining usage, inputs, and outputs.

## Legacy Code &Output :




## Refactored Code &Output :




## Observation

1. The legacy programs were unorganized, repetitive, and lacked error handling.

2. After refactoring with AI tools, the codes became cleaner and easier to understand.
3. Using functions, classes, and comprehensions improved performance and readability.
4. Error handling and documentation made the programs more reliable.
5. Overall, the refactored versions are modern, efficient, and easier to maintain.