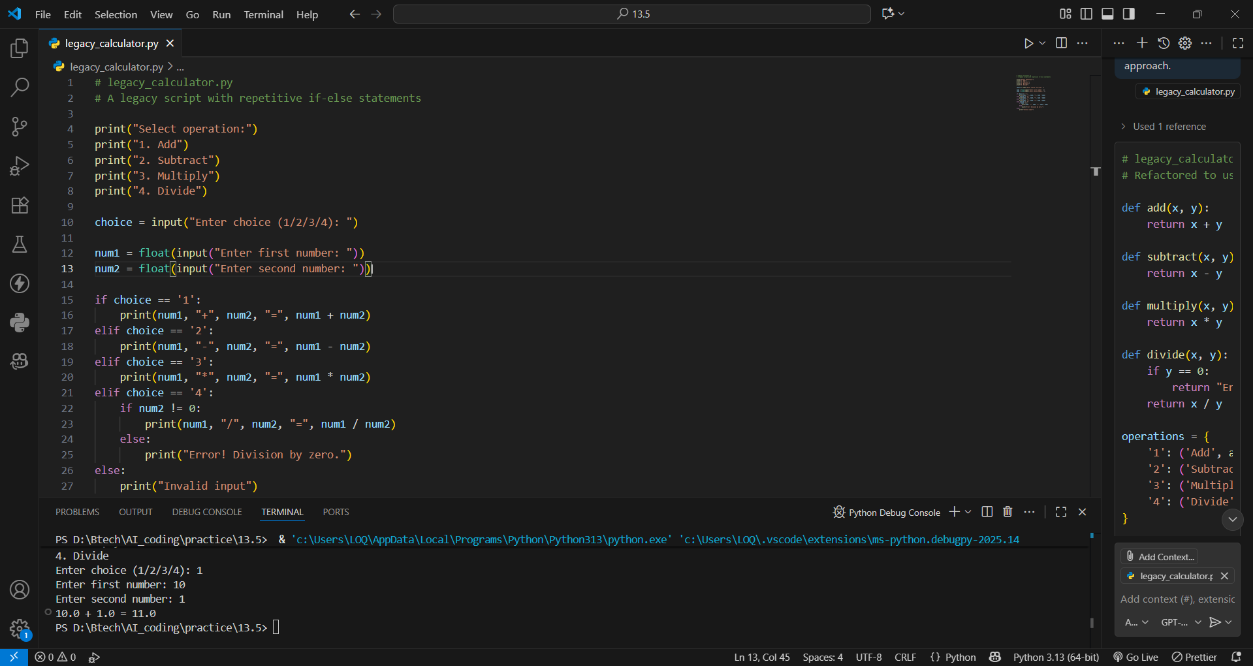
Assignment-13  
  
**Hno:2053A52L16**

**Lab 13: Code Refactoring – Improving Legacy Code with AI Suggestions**

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

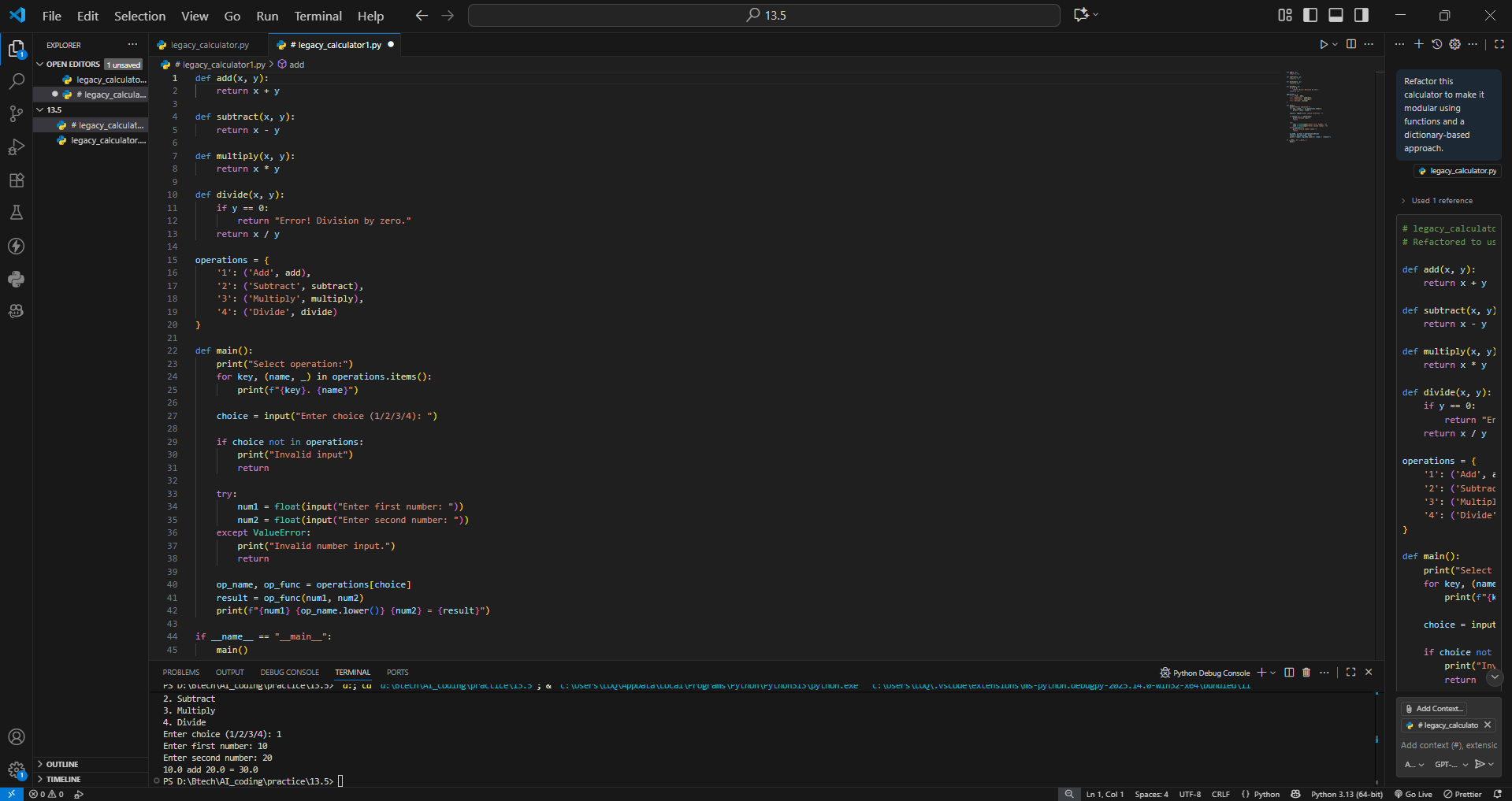
**Prompt:** Refactor this basic calculator code to make it cleaner and modular.  
Use functions for each operation (add, subtract, multiply, divide) and a dictionary to map user choices to these functions.  
Make sure it asks the user for two numbers, performs the selected operation, and handles division by zero errors.

**Output(Before Refactor):**



**Output(After Refactor):**

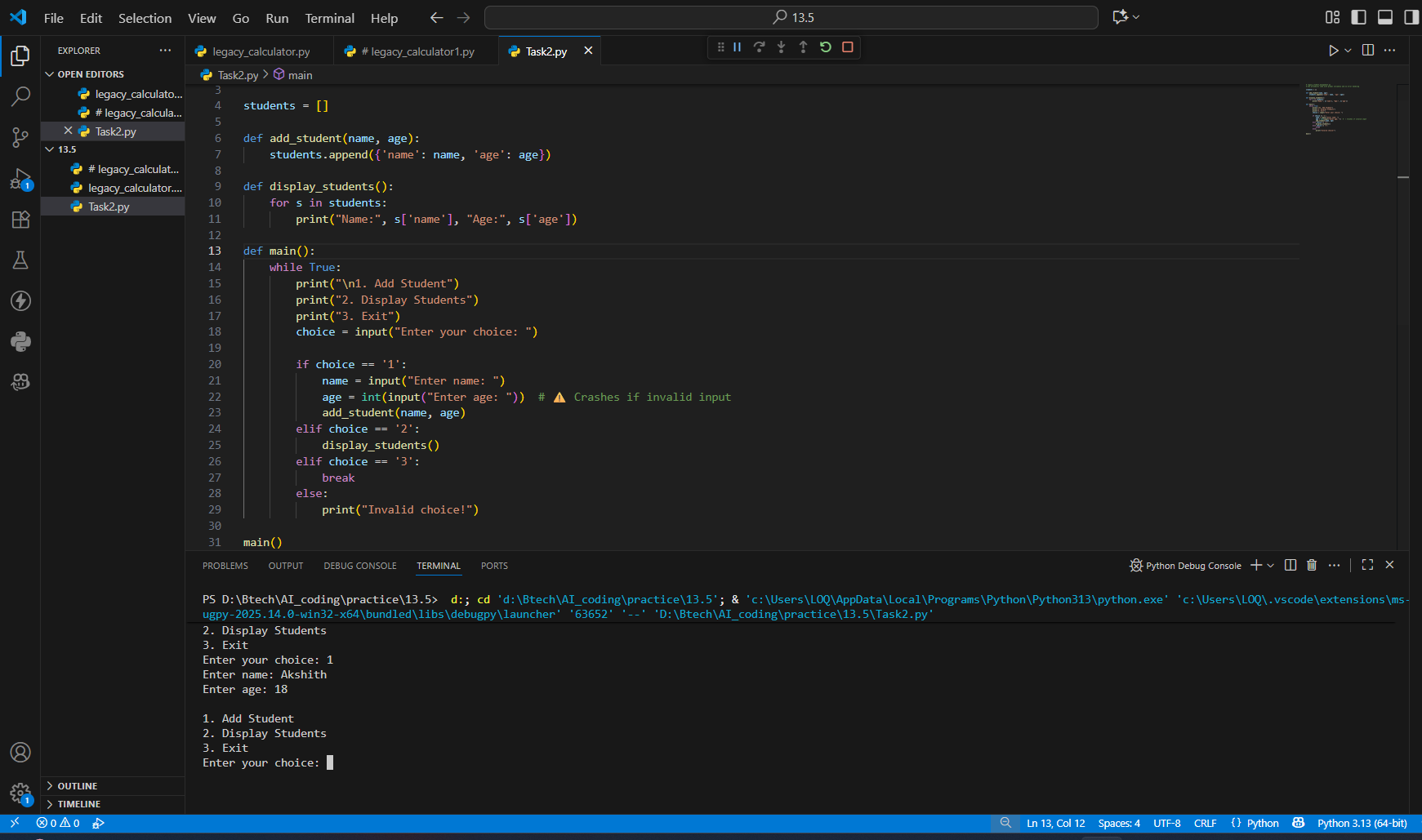
**Prompt:** Refactor this calculator to make it modular using functions and a dictionary-based approach

****

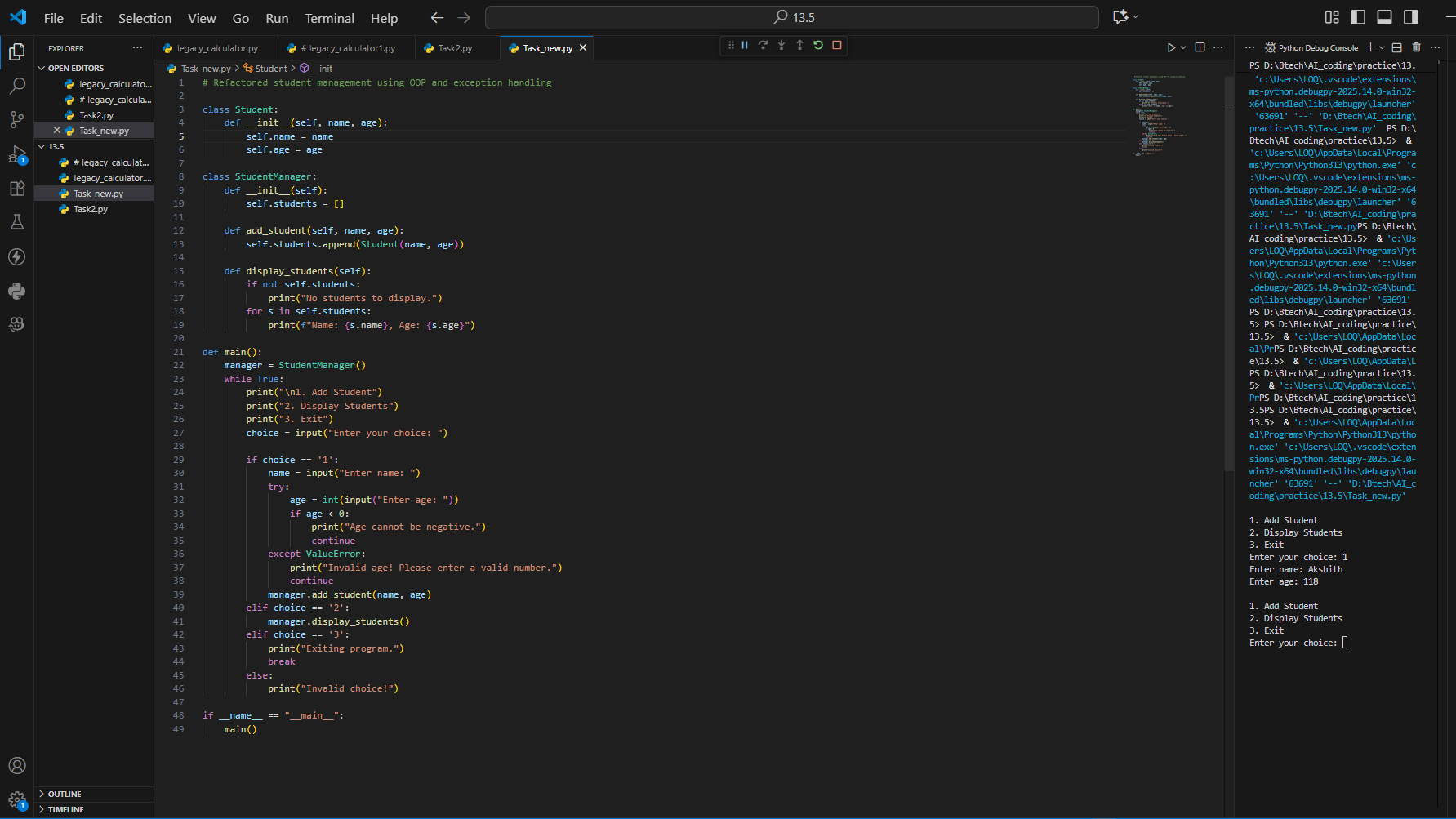
**Observation:**  
Old code had many if-else statements and was hard to maintain.  
Copilot made it modular using functions and a dictionary.  
Now it’s cleaner, easier to read, and simple to add new operations.

**Task 2:** Modernizing a Student Database Program Scenario:

**Output(Before Refactor):**



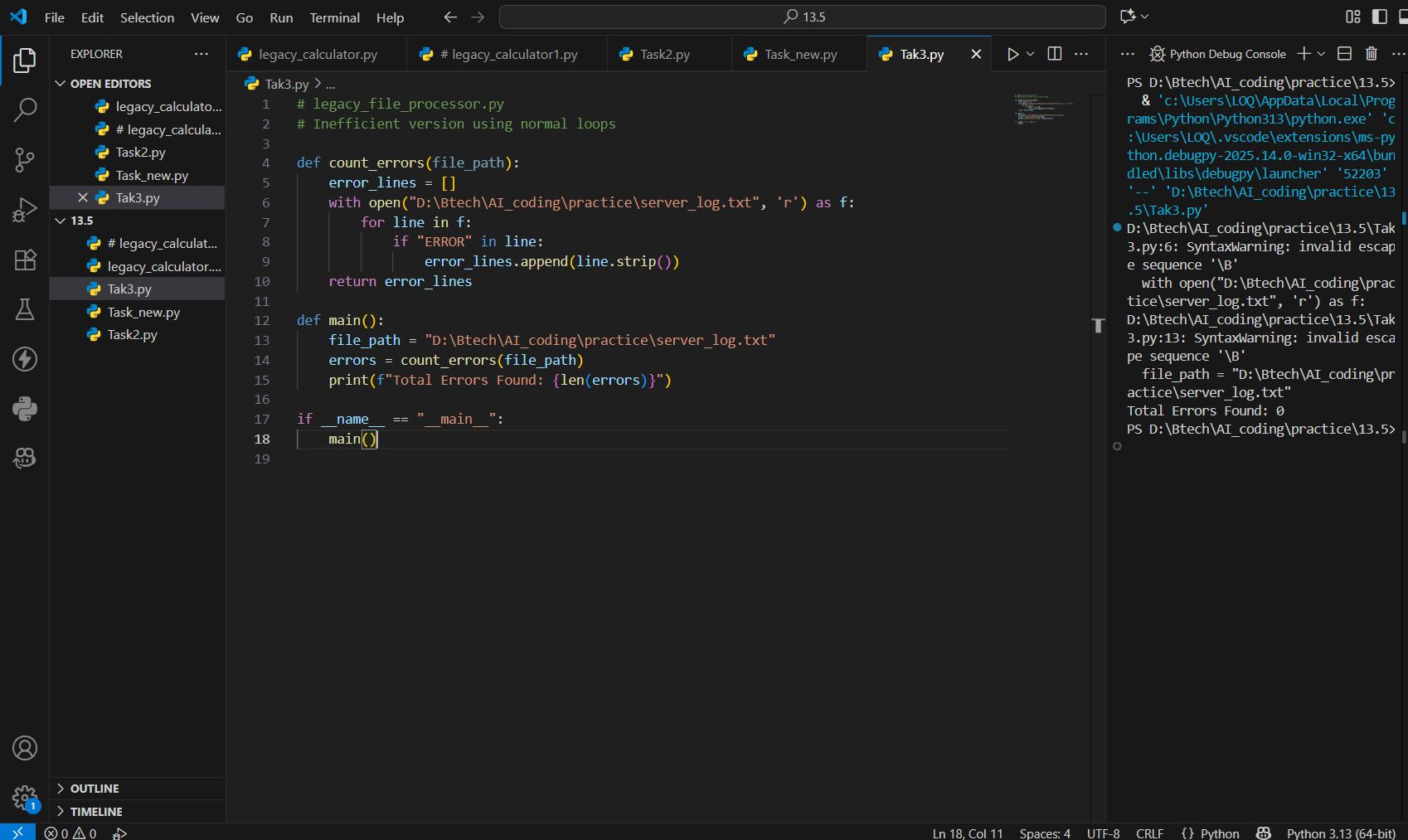
**Output(After Refactor):**



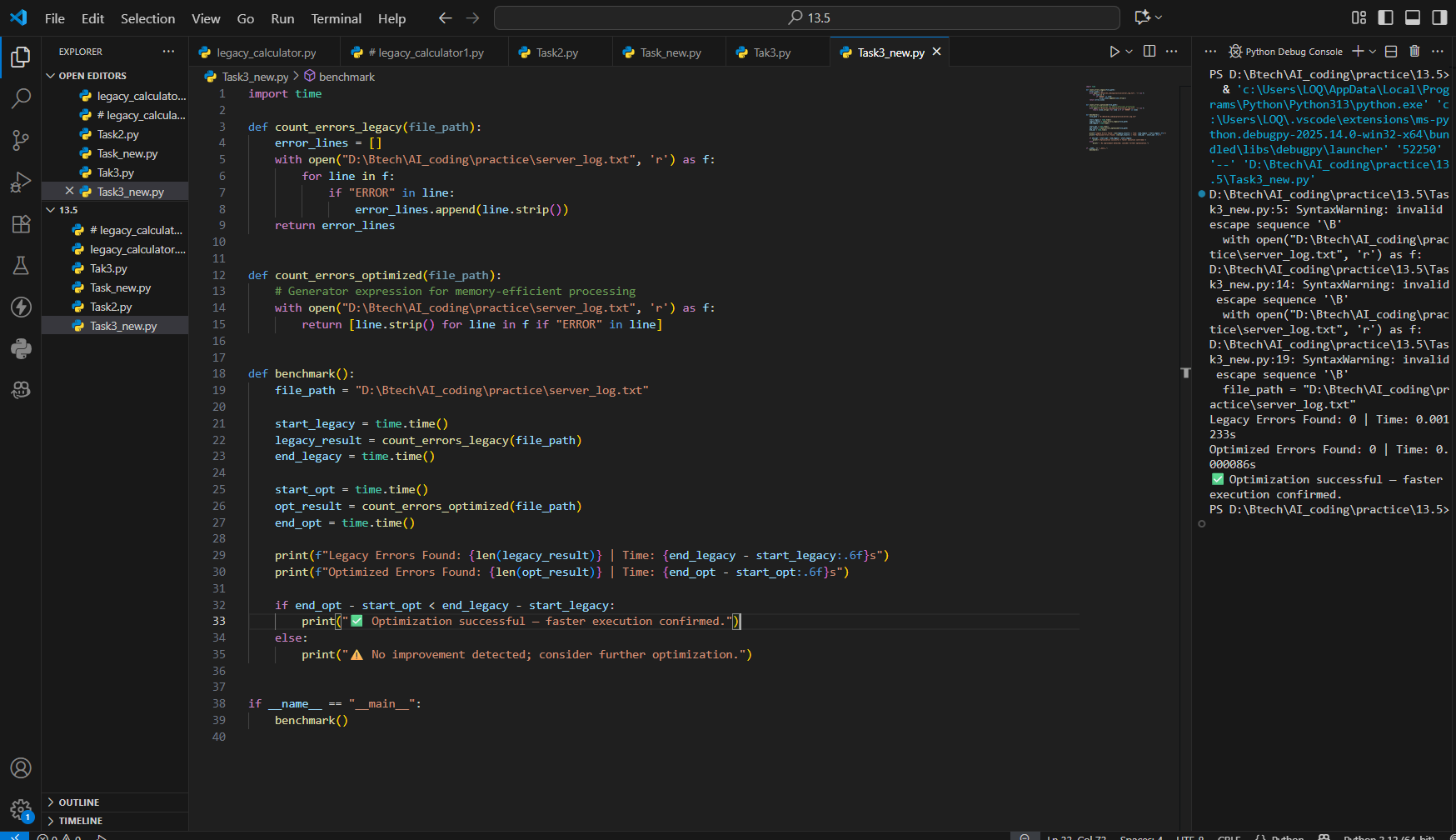
**Observation:**  
Old code used global variables and crashed on wrong input.  
Copilot converted it into classes with error handling.  
Now it’s stable, organized, and user-friendly.

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

**Output(Before Refactor):**

****

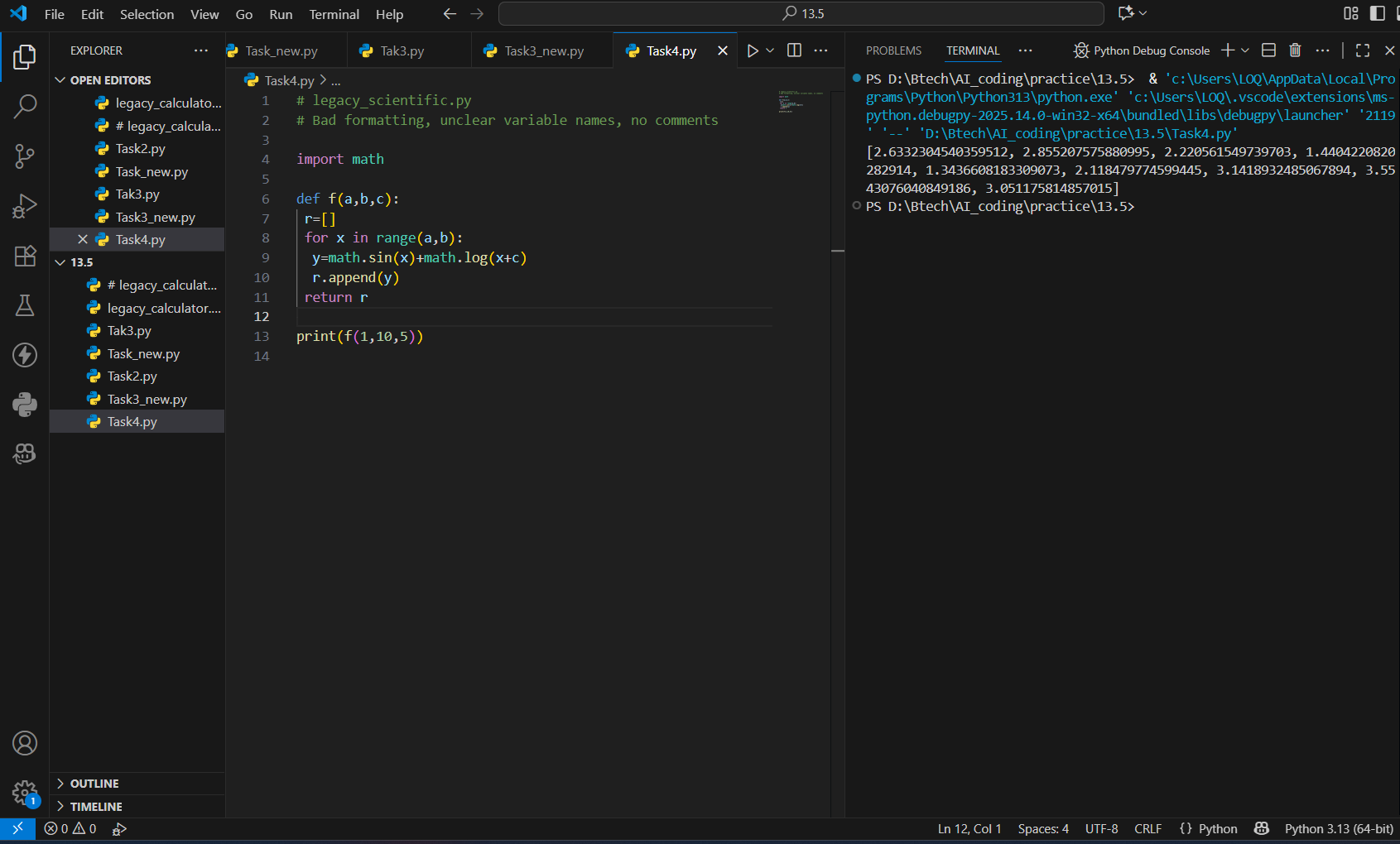
**Output(After Refactor):**

****

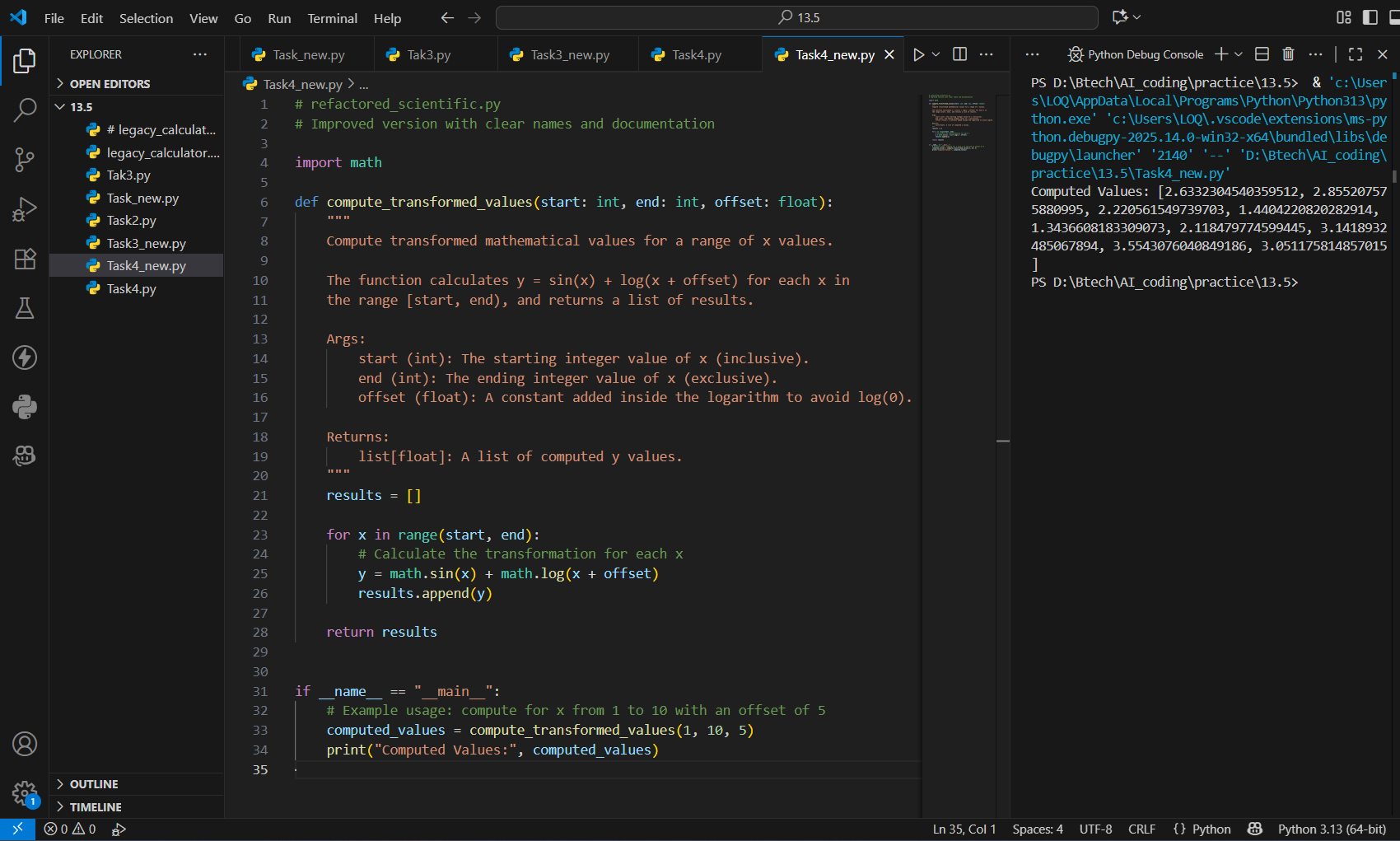
**Observation:**  
Old script read files line by line with slow loops.  
Copilot replaced them with generators and list comprehensions.  
The new version runs faster and uses less memory.

**Task 4**: Enhancing Readability and Documentation Scenario

**Output(Before Refactor):**



**Output(After Refactor):**



**Observation:**  
Old code had poor names, no comments, and was hard to read.  
Copilot improved variable names, formatting, and added comments.  
A README was created to explain usage clearly.