

**SR UNIVERSITY**

**Assignment-9.5**

Name : Eldi Arun Kumar

Enrollment No : 2503A51L27

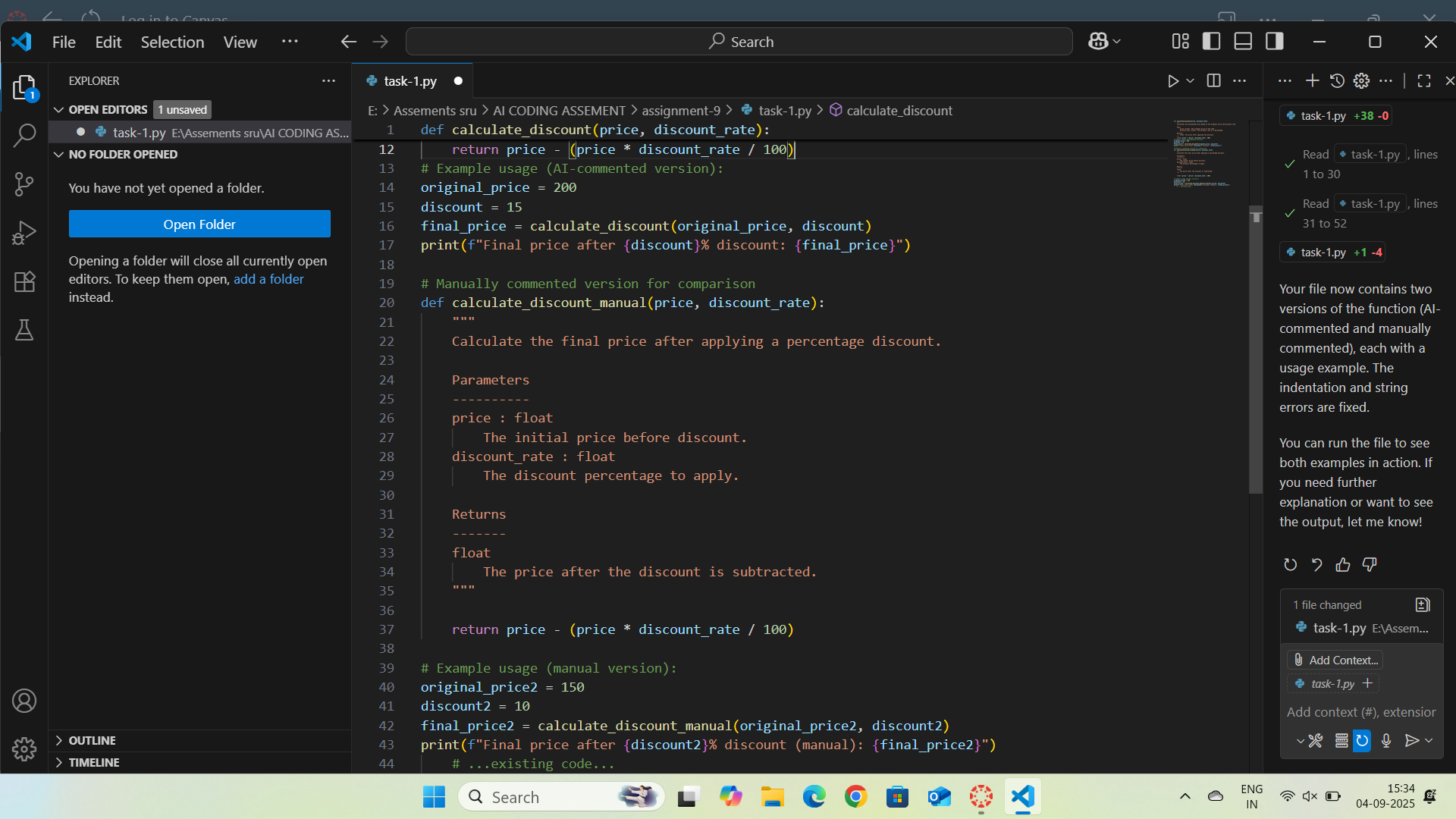
Batch No : 19

**Task Description 1:**

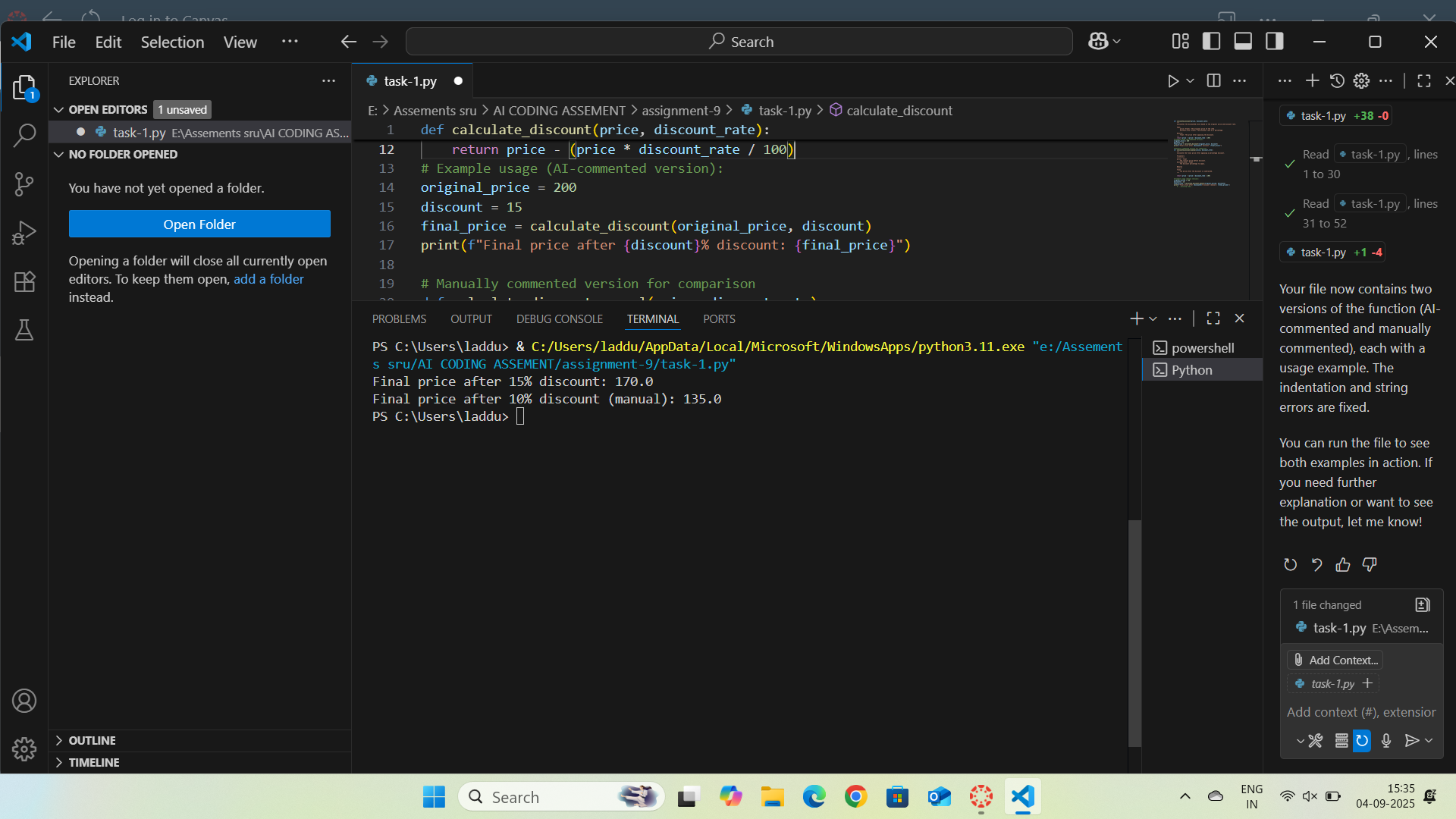
Scenario: You have been given a Python function without comments.  
def calculate\_discount(price, discount\_rate):  
return price - (price \* discount\_rate / 100)  
• Use an AI tool (or manually simulate it) to generate line-by-line  
comments for the function.  
• Modify the function so that it includes a docstring in Google-style  
or NumPy-style format.  
• Compare the auto-generated comments with your manually  
written version.

**Prompt:** Generate line-by-line comments for this Python function, add a Google-style docstring, and compare AI-generated comments with a manual version.

**Code:**

****

**Output:**

****

**Observation:**

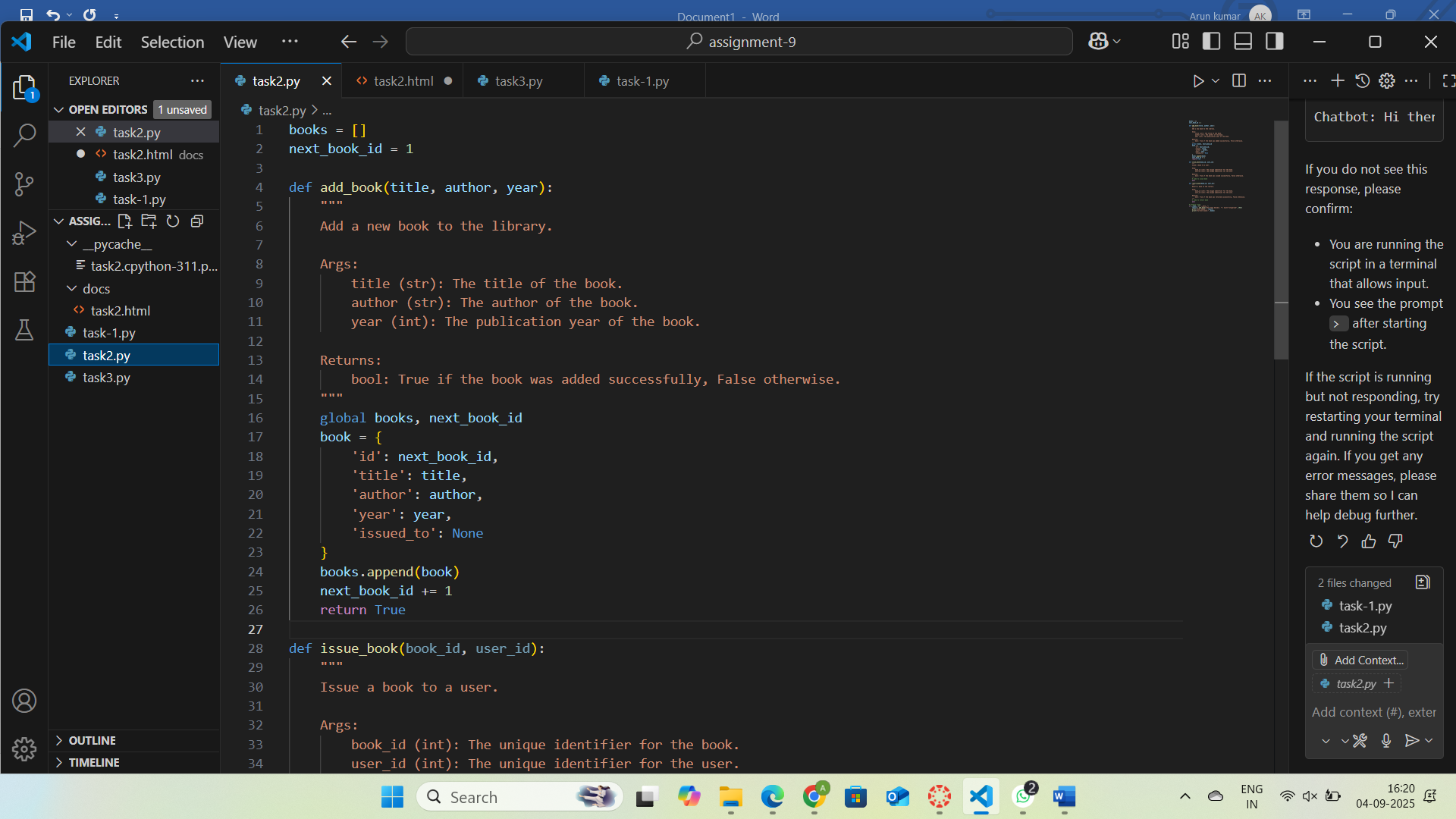
The AI-generated comments were clear but a bit generic, mostly restating the code logic. My manually written comments gave more context, such as why the discount is subtracted and how percentages are calculated. The docstring improved readability by specifying parameters and return values.

**Task Description 2:**

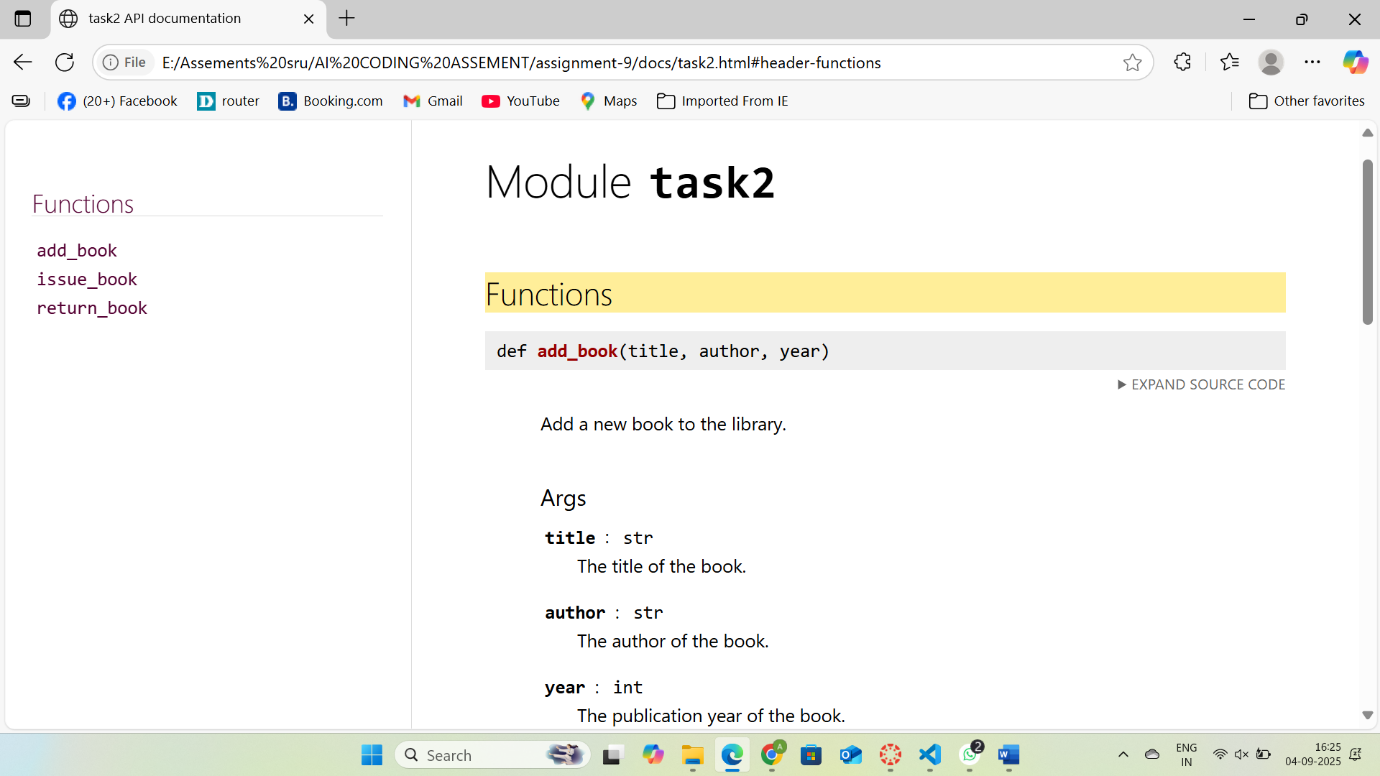
Scenario: A team is building a Library Management System with  
multiple functions.  
def add\_book(title, author, year):  
# code to add book  
pass  
def issue\_book(book\_id, user\_id):  
# code to issue book  
Pass  
• Write a Python script that uses docstrings for each function (with  
input, output, and description).  
• Use a documentation generator tool (like pdoc, Sphinx, or  
MkDocs) to automatically create HTML documentation.  
• Submit both the code and the generated documentation as output.

**Prompt:** Generate a Python script for a Library Management System with functions add\_book and issue\_book, each having Google-style docstrings (inputs, outputs, description). Then generate HTML documentation using pdoc or Sphinx. Provide both the script and the generated docs.

**Code:**



**Output:**

****

**Observation:** The use of docstrings made the functions self-explanatory by clearly defining purpose, inputs, and outputs. Generating HTML documentation with a tool like pdoc or Sphinx ensured consistent, professional documentation without manual formatting. Overall, the process improved readability, maintainability, and ease of use for the Library Management System.

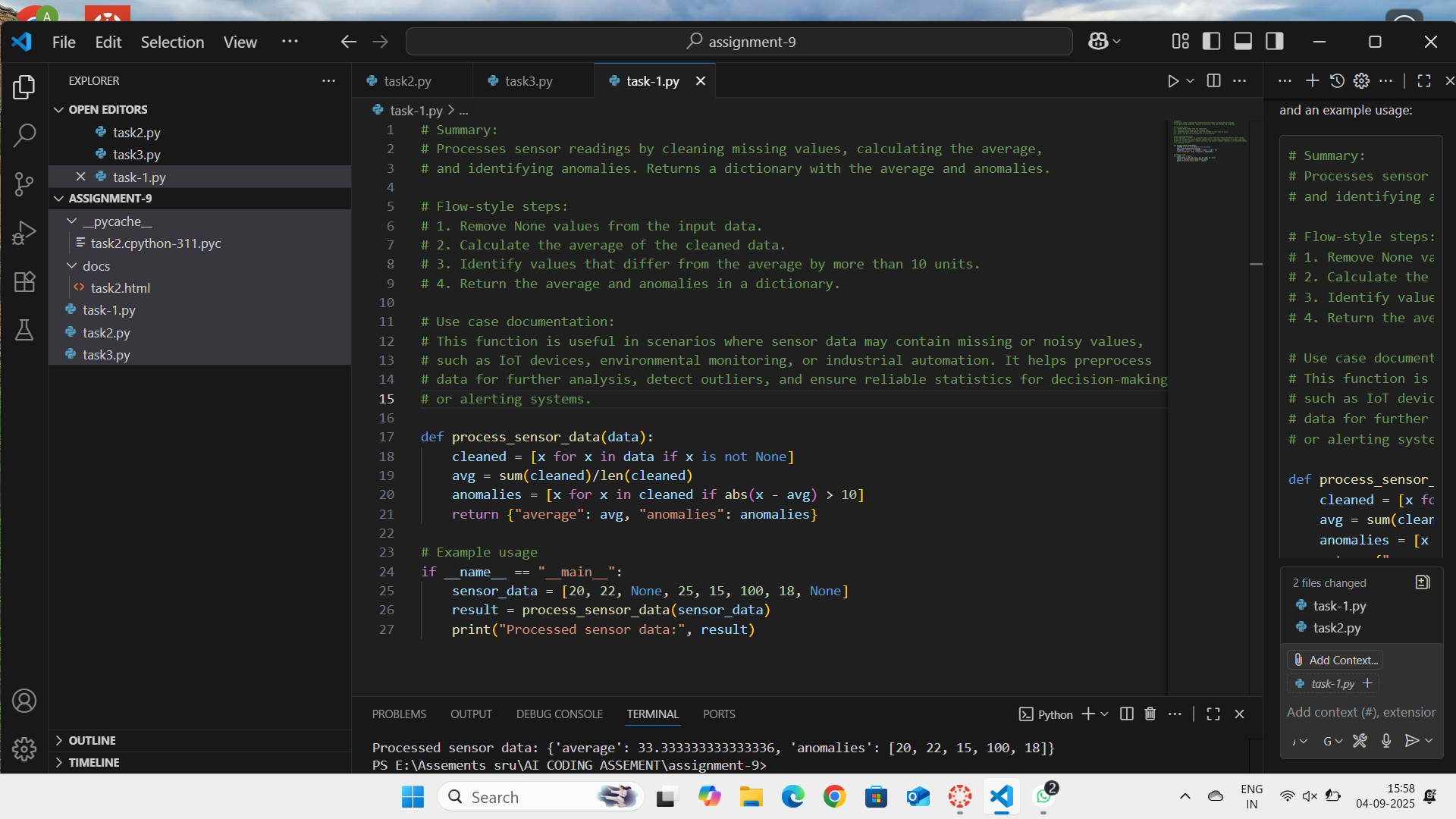
**Task Description 3:**

Scenario: You are reviewing a colleague’s codebase containing long  
functions.

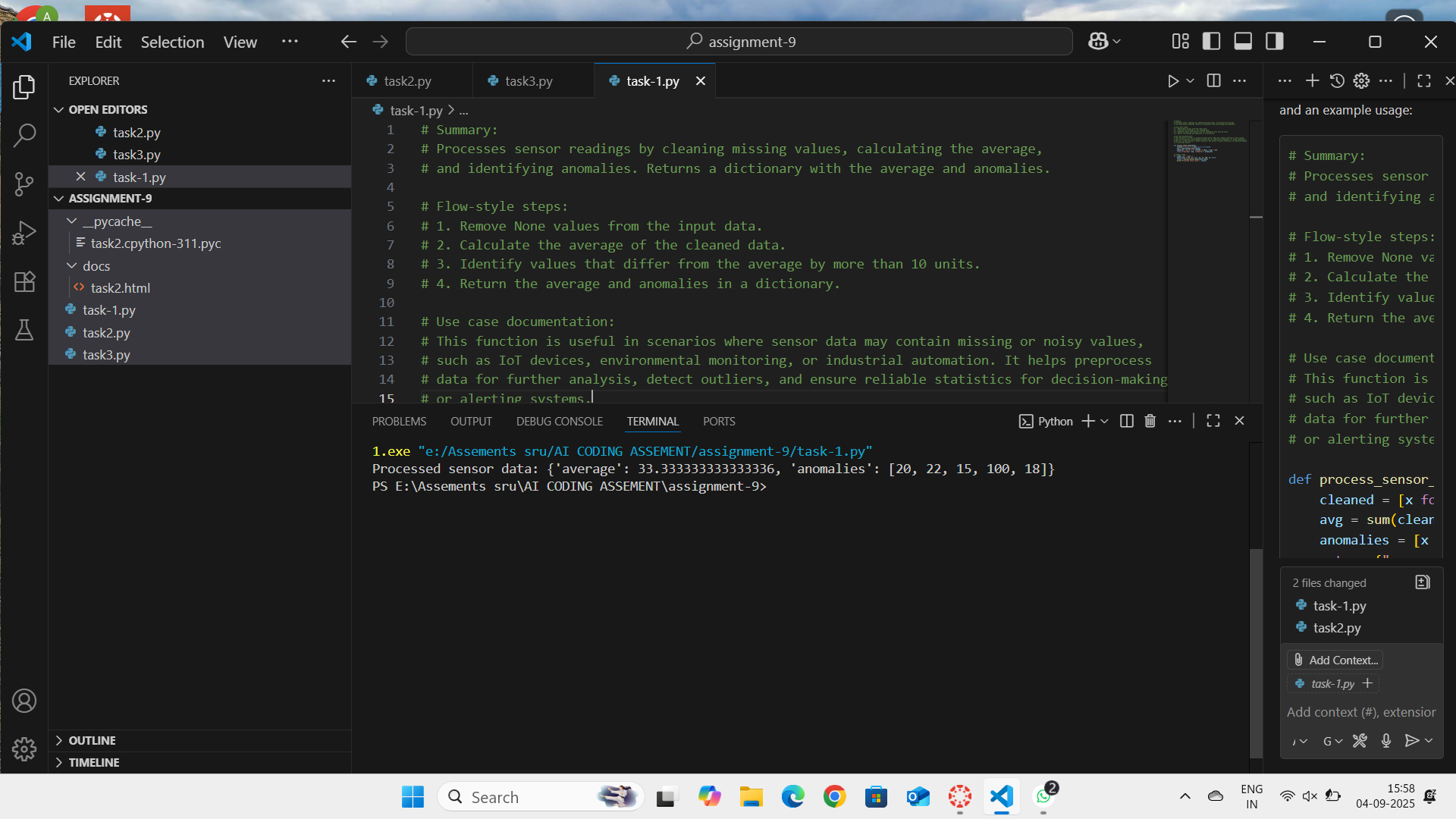
def process\_sensor\_data(data):  
cleaned = [x for x in data if x is not None]  
avg = sum(cleaned)/len(cleaned)  
anomalies = [x for x in cleaned if abs(x - avg) > 10]  
return {"average": avg, "anomalies": anomalies}  
• Generate a summary comment explaining the purpose of the  
function in 2–3 lines.  
• Create a flow-style comment (step-by-step explanation).  
• Write a short paragraph of documentation describing possible use  
cases of this function in real-world scenarios.

**Prompt:** Generate a 2–3 line summary comment for the process\_sensor\_data function, add step-by-step flow comments, and write a short paragraph on real-world use cases of this function.

**Code:**

****

**Output:**

****

**Observation:**

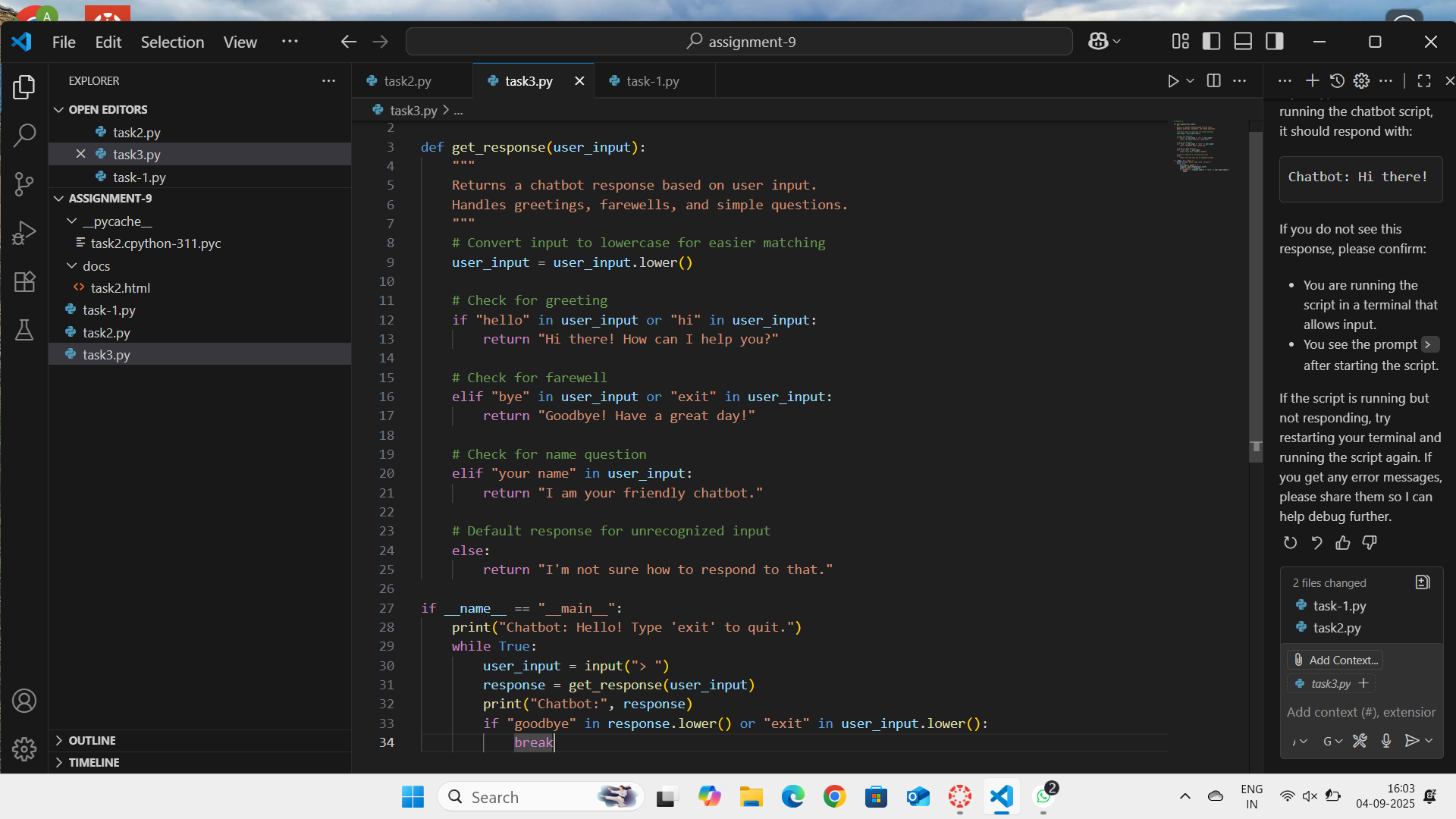
The summary comment concisely explained the function’s goal of cleaning sensor data, calculating averages, and detecting anomalies. The flow-style comment made the logic easier to follow step by step.

**Task Description 4:**

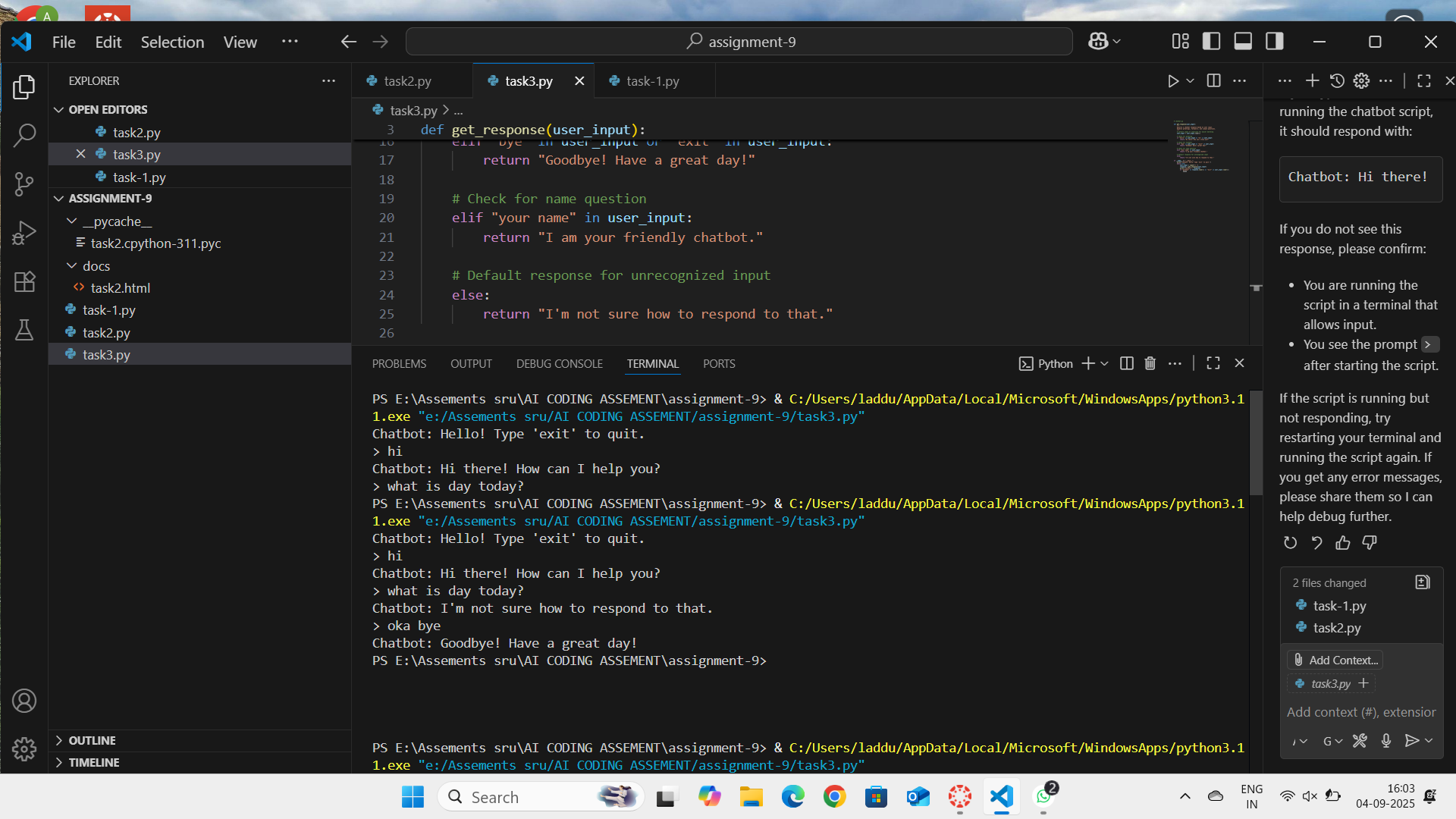
Scenario: You are part of a project team that develops a Chatbot  
Application. The team needs documentation for maintainability.  
• Write a README.md file for the chatbot project (include project  
description, installation steps, usage, and example).  
• Add inline comments in the chatbot’s main Python script (focus  
on explaining logic, not trivial code).  
• Use an AI-assisted tool (or simulate it) to generate a usage guide  
in plain English from your code comments.  
• Reflect: How does automated documentation help in real-time  
projects compared to manual documentation?

**Prompt:** Generate a README.md for a Chatbot project (description, installation, usage, example). Add inline comments in the main Python script (logic-focused). Then create a plain-English usage guide from comments and reflect on benefits of automated vs manual documentation.

**Code:**

****

**Output:**

****

**Observation:** The README provided clear project details and setup instructions, making it easy for new developers to get started. Inline comments explained the logic flow, improving maintainability. The AI-generated usage guide translated code comments into user-friendly language, reducing onboarding time.