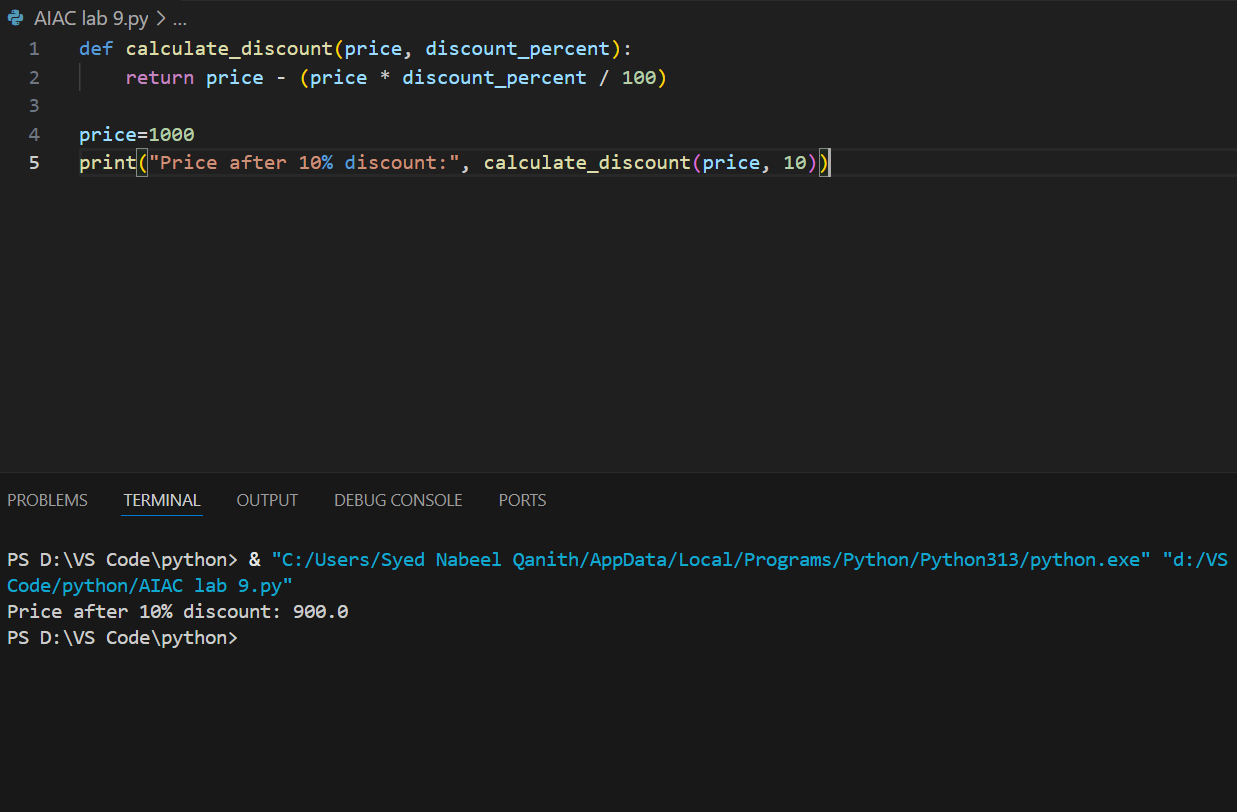
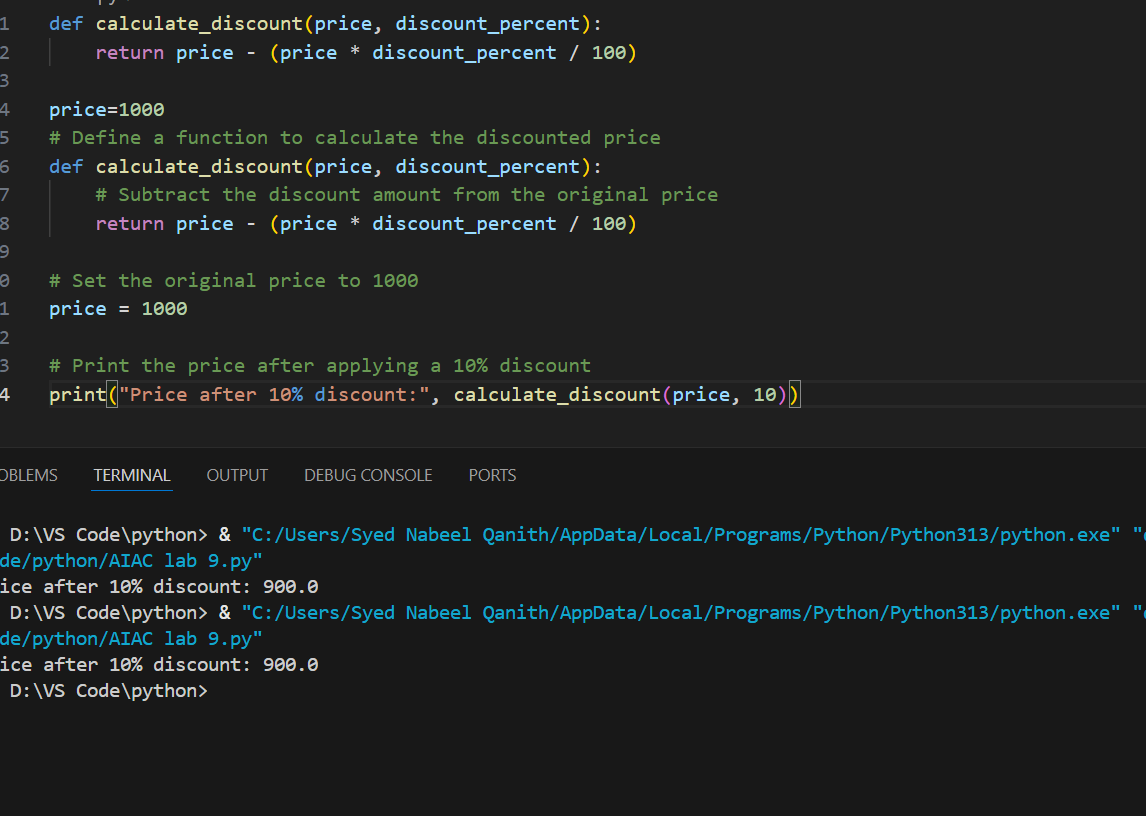
LAB ASSIGNMENT 9

Task Description #1 (Automatic Code Commenting)  
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

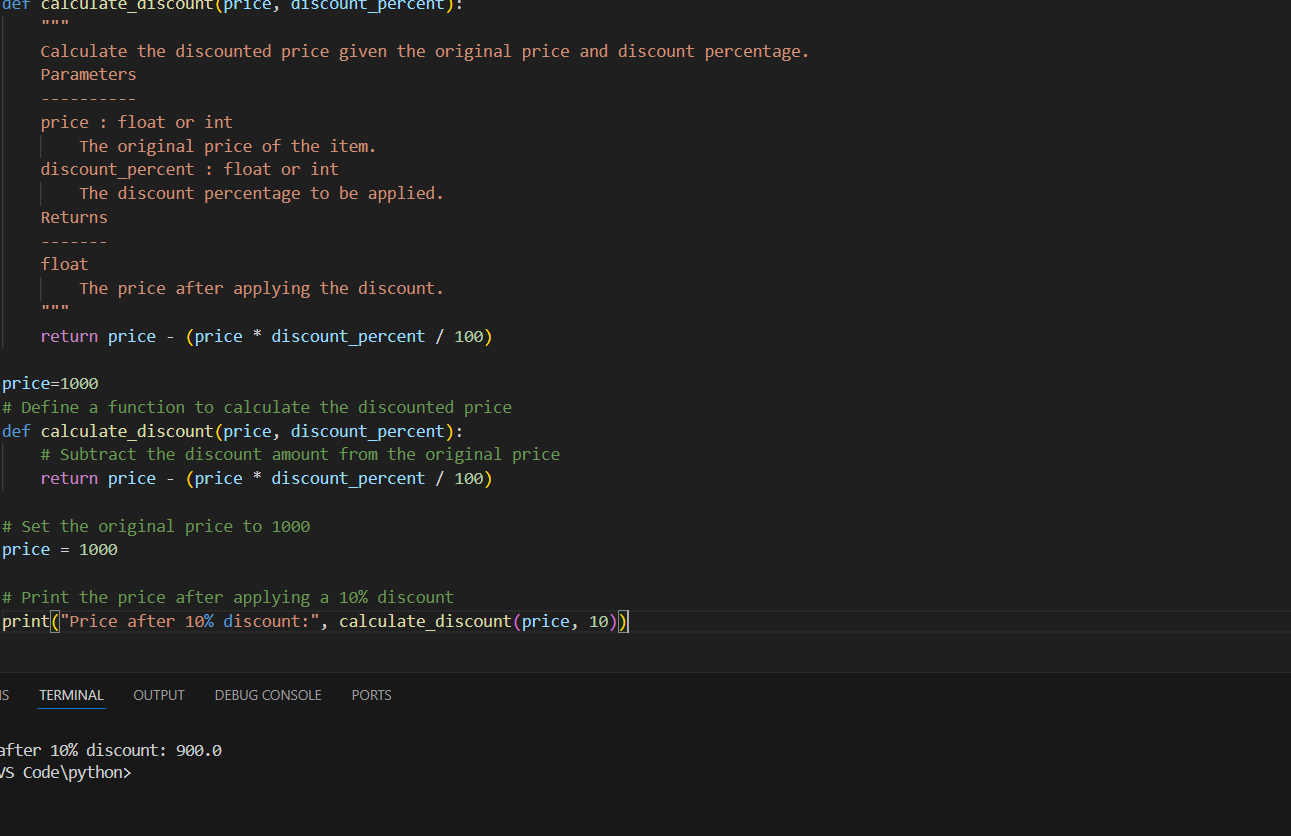
PROMPT:give a python function to calculate discount without comments



Prompt:Give line by line comments for the abouve program



Prompt: Modify the function so that it includes a docstring in Google-style  
or NumPy-style format



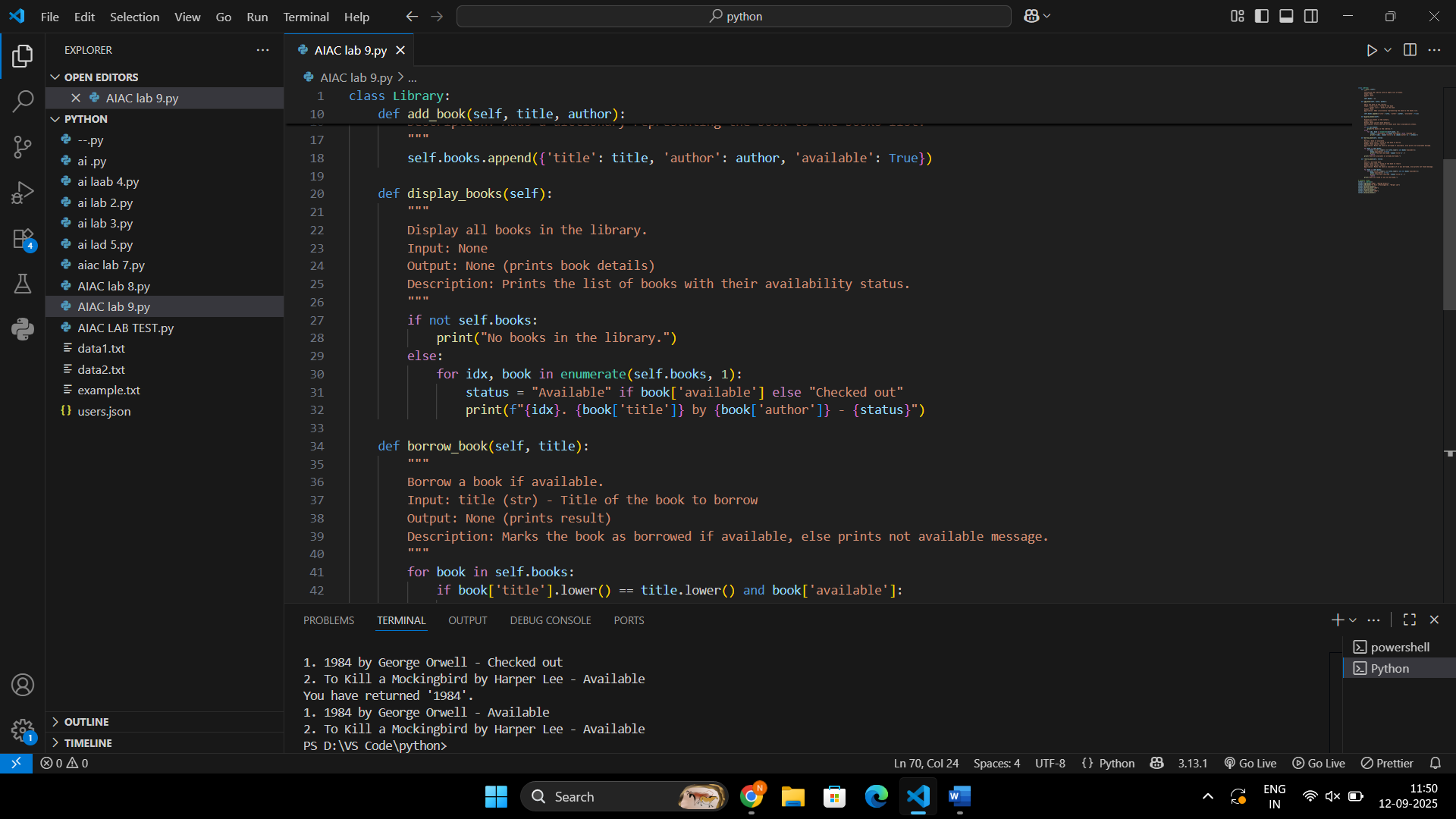
To compare auto-generated comments with a manually written version, please provide:

* The code with auto-generated comments.
* Your manually written comments (or the code with them).

Once you share both, I can analyze and compare clarity, accuracy, and usefulness for you. Please paste both versions or specify the file(s) to review.

ask Description #2 (API Documentation Generator)  
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:build a library management system with functions using docstrip with input,output and description.



**Library.init**

* Description: Initialize the library with an empty list of books.
* Input: None
* Output: None

**Library.add\_book**

* Description: Add a new book to the library.
* Input: title (str) - Title of the book; author (str) - Author of the book
* Output: None

**Library.display\_books**

* Description: Display all books in the library.
* Input: None
* Output: None (prints book details)

**Library.borrow\_book**

* Description: Borrow a book if available. Marks the book as borrowed if available, else prints not available message.
* Input: title (str) - Title of the book to borrow
* Output: None (prints result)

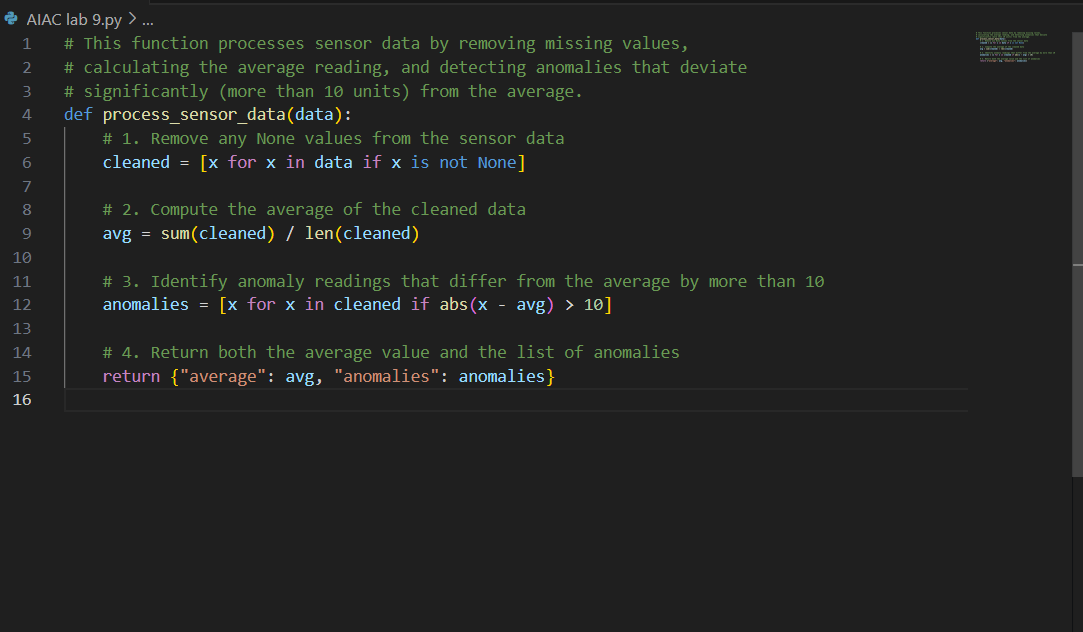
**Library.return\_book**

* Description: Return a borrowed book. Marks the book as available if it was borrowed, else prints not found message.
* Input: title (str) - Title of the book to return
* Output: None (prints result

Task Description #3 (AI-Assisted Code Summarization)  
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 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



**Documentation Paragraph**

This function can be applied in real-world IoT or monitoring systems where sensor reliability is critical. For example, it could be used in industrial equipment monitoring to detect sudden spikes or drops in temperature or pressure readings, in smart agriculture to identify faulty soil moisture readings, or in healthcare wearables to highlight abnormal heart rate values. By providing both a cleaned average and flagged anomalies, it enables engineers and analysts to ensure data quality and quickly respond to unusual sensor behavior.

Task Description #4 (Real-Time Project Documentation)  
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?

# Chatbot Application

A simple Python-based chatbot that interacts with users through text input.

This project demonstrates the use of basic Natural Language Processing (NLP) techniques and rule-based responses.

---

## Features

- Text-based chatbot with simple rule-based responses

- Easily extendable logic

- Example usage included

---

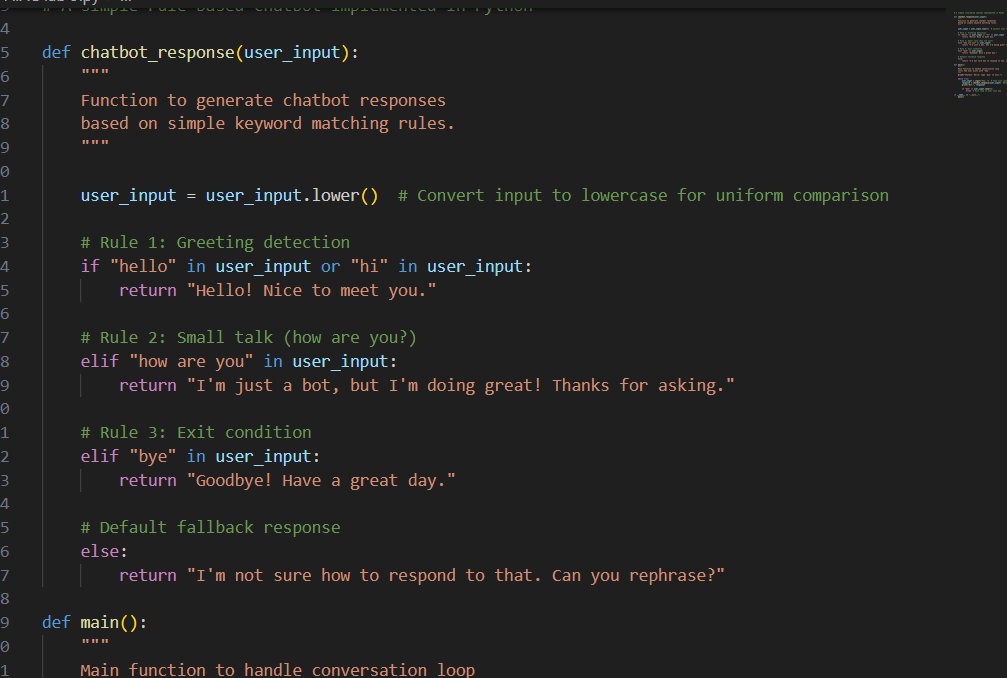
## Installation

1. Clone this repository:

```bash

git clone https://github.com/your-username/chatbot-app.git

cd chatbot-app



**. AI-Simulated Usage Guide (Generated from Code Comments)**

Plain English guide (as if an AI read the comments and explained it):

* The chatbot listens for user input and compares it against predefined rules.
* If the input contains greetings like **"hello" or "hi"**, the bot replies with a friendly greeting.
* If the user asks **"how are you"**, the bot replies with a casual response.
* If the user types **"bye"**, the bot says goodbye and ends the conversation.
* For any other input, the bot asks the user to rephrase.
* The conversation continues in a loop until the user exits with "bye"

**. Reflection**

**Automated documentation** helps in real-time projects by:

* **Consistency**: Code comments and docstrings can automatically generate usage guides, ensuring docs match the code.
* **Efficiency**: Saves developer time compared to writing long manuals manually.
* **Maintainability**: When code changes, regenerating docs ensures they remain updated.
* **Accessibility**: Non-technical stakeholders can understand the project through plain-English usage guides.

**Manual documentation**, while flexible, often becomes outdated quickly if not maintained. Automation ensures real-time synchronization between code and documentation.