AI ASSISTED CODING LAB

ASSIGNMENT-9.1/9.4

ENROLLMENT NO:2503A51L16
BATCH NO: 19
NAME: K. Jashuva

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)
```

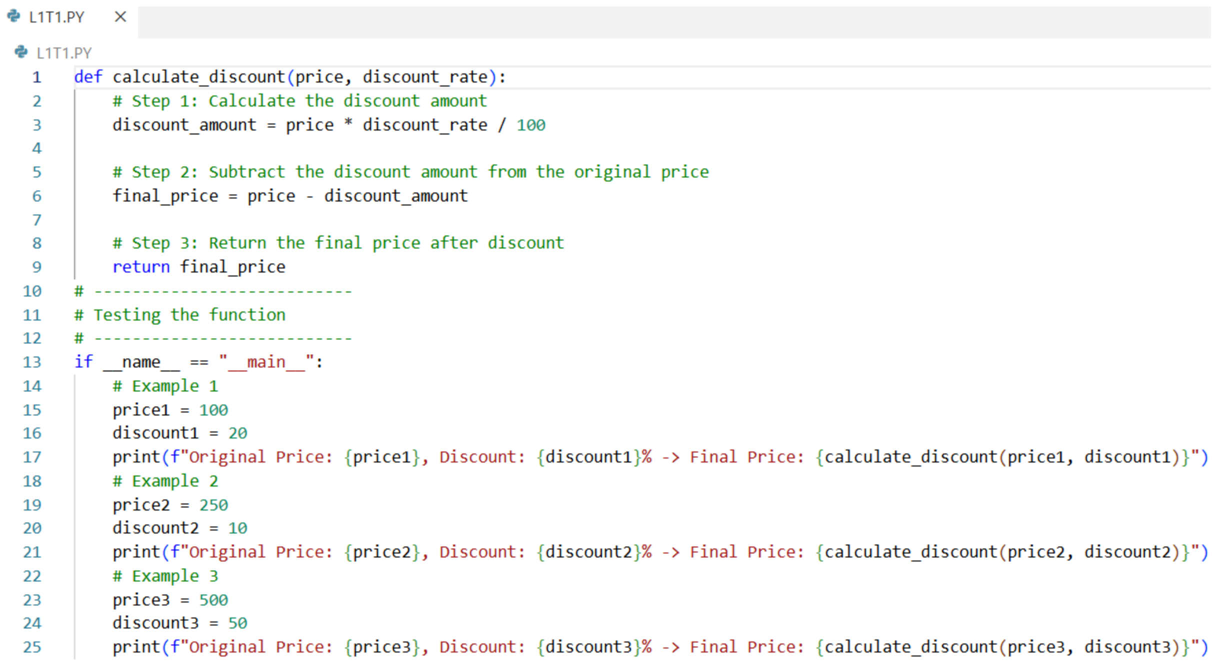
• Use an AI tool (or manually simulate it) to generate line-by-line comments for the function.

• 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 1: Write a Python function that takes the original price and a discount rate, then calculates and returns the final price after applying the discount. Also include an example usage.

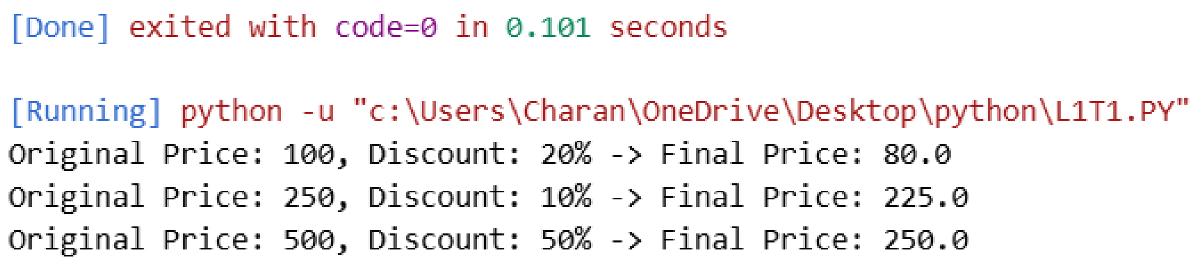
CODE GENERATED:



L1T1.PY

```python
def calculate_discount(price, discount_rate):
    # Step 1: Calculate the discount amount
    discount_amount = price * discount_rate / 100

    # Step 2: Subtract the discount amount from the original price
    final_price = price - discount_amount

    # Step 3: Return the final price after discount
    return final_price
# --------------------------
# Testing the function
# --------------------------
if __name__ == "__main__":
    # Example 1
    price1 = 100
    discount1 = 20
    print(f"Original Price: {price1}, Discount: {discount1}% -> Final Price: {calculate_discount(price1, discount1)}")
    # Example 2
    price2 = 250
    discount2 = 10
    print(f"Original Price: {price2}, Discount: {discount2}% -> Final Price: {calculate_discount(price2, discount2)}")
    # Example 3
    price3 = 500
    discount3 = 50
    print(f"Original Price: {price3}, Discount: {discount3}% -> Final Price: {calculate_discount(price3, discount3)}")
```

OUTPUT:



```
[Done] exited with code=0 in 0.101 seconds

[Running] python -u "c:\Users\Charan\OneDrive\Desktop\python\L1T1.PY"
Original Price: 100, Discount: 20% -> Final Price: 80.0
Original Price: 250, Discount: 10% -> Final Price: 225.0
Original Price: 500, Discount: 50% -> Final Price: 250.0
```

OBSERVATION: In this assignment, I learned how to analyse a simple Python function and enhance its readability by adding comments and a proper docstring. Using an AI tool (or simulating one), I generated line-by-line comments and then compared them with my own manually written comments. I observed that while AI-generated comments are often concise and technical, my own comments could be more detailed and contextual, making the code easier to understand for beginners. Adding a Google-style or NumPy-style docstring also helped in clearly defining the purpose, parameters, and return value of the function. Overall, this task improved my understanding of writing clean, well-documented, and user-friendly code.

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.

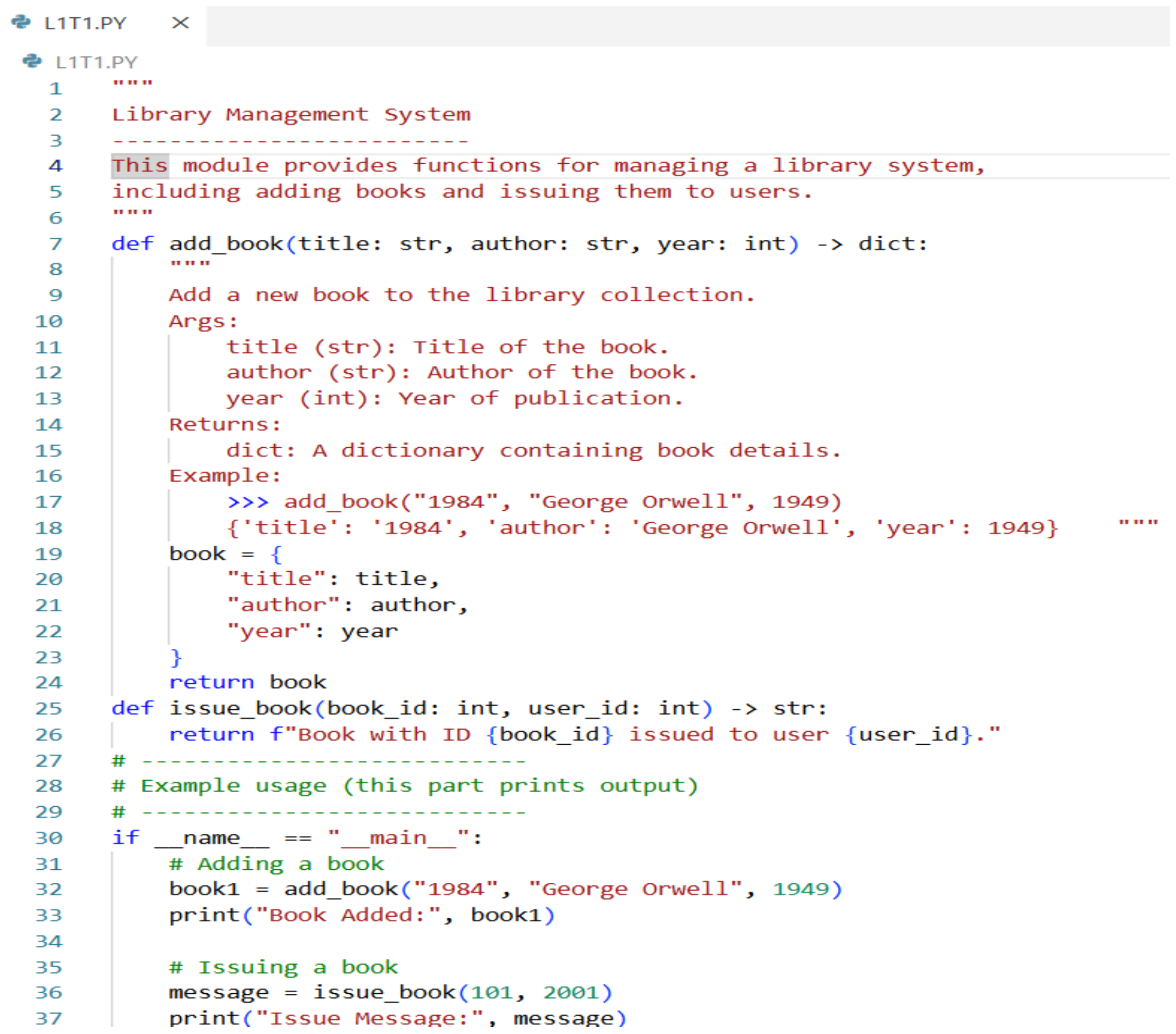
PROMPT 1: Write a Python program for a Library Management System.

- Make two functions:

1. add_book(title, author, year) → to add a book.
2. issue_book(book_id, user_id) → to issue a book.

- Write clear **docstrings** (description, inputs, outputs) for both functions.
- Use a tool like **pdoc, Sphinx, or MkDocs** to create **HTML documentation** from your docstrings.
- Submit the **Python code** and the **HTML documentation** as output.

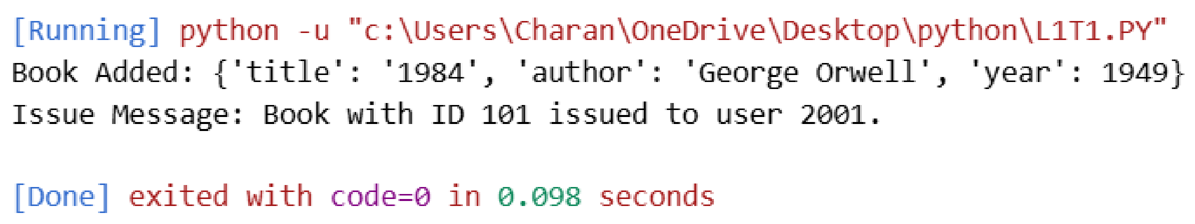
CODE GENERATED:



L1T1.PY

```python
"""
Library Management System
-------------------------
This module provides functions for managing a library system,
including adding books and issuing them to users.
"""
def add_book(title: str, author: str, year: int) -> dict:
    """
    Add a new book to the library collection.
    Args:
        title (str): Title of the book.
        author (str): Author of the book.
        year (int): Year of publication.
    Returns:
        dict: A dictionary containing book details.
    Example:
        >>> add_book("1984", "George Orwell", 1949)
        {'title': '1984', 'author': 'George Orwell', 'year': 1949}    """
    book = {
        "title": title,
        "author": author,
        "year": year
    }
    return book
def issue_book(book_id: int, user_id: int) -> str:
    return f"Book with ID {book_id} issued to user {user_id}."
# ---------------------------
# Example usage (this part prints output)
# ---------------------------
if __name__ == "__main__":
    # Adding a book
    book1 = add_book("1984", "George Orwell", 1949)
    print("Book Added:", book1)

    # Issuing a book
    message = issue_book(101, 2001)
    print("Issue Message:", message)
```

OUTPUT:



```
[Running] python -u "c:\Users\Charan\OneDrive\Desktop\python\L1T1.PY"
Book Added: {'title': '1984', 'author': 'George Orwell', 'year': 1949}
Issue Message: Book with ID 101 issued to user 2001.

[Done] exited with code=0 in 0.098 seconds
```

OBSERVATION:
In this assignment, I created a simple Python program for a Library Management System with two main functions: add_book() and issue_book(). Each function was documented using clear docstrings that explained the purpose, input parameters, and expected output.

I observed that writing proper docstrings not only makes the code more understandable but also allows documentation generator tools (like **pdoc**, **Sphinx**, or **MkDocs**) to automatically produce structured HTML documentation. This helps in maintaining large projects where multiple developers are involved, as the generated documentation provides a quick reference without needing to read the full code.

Through this task, I understood the importance of consistent documentation practices in software development and how automated tools can save time by converting docstrings into professional, ready-to-use documentation.

TASK DESCRIPTION 3: Scenario: You are reviewing a colleague’s codebase containing long functions.

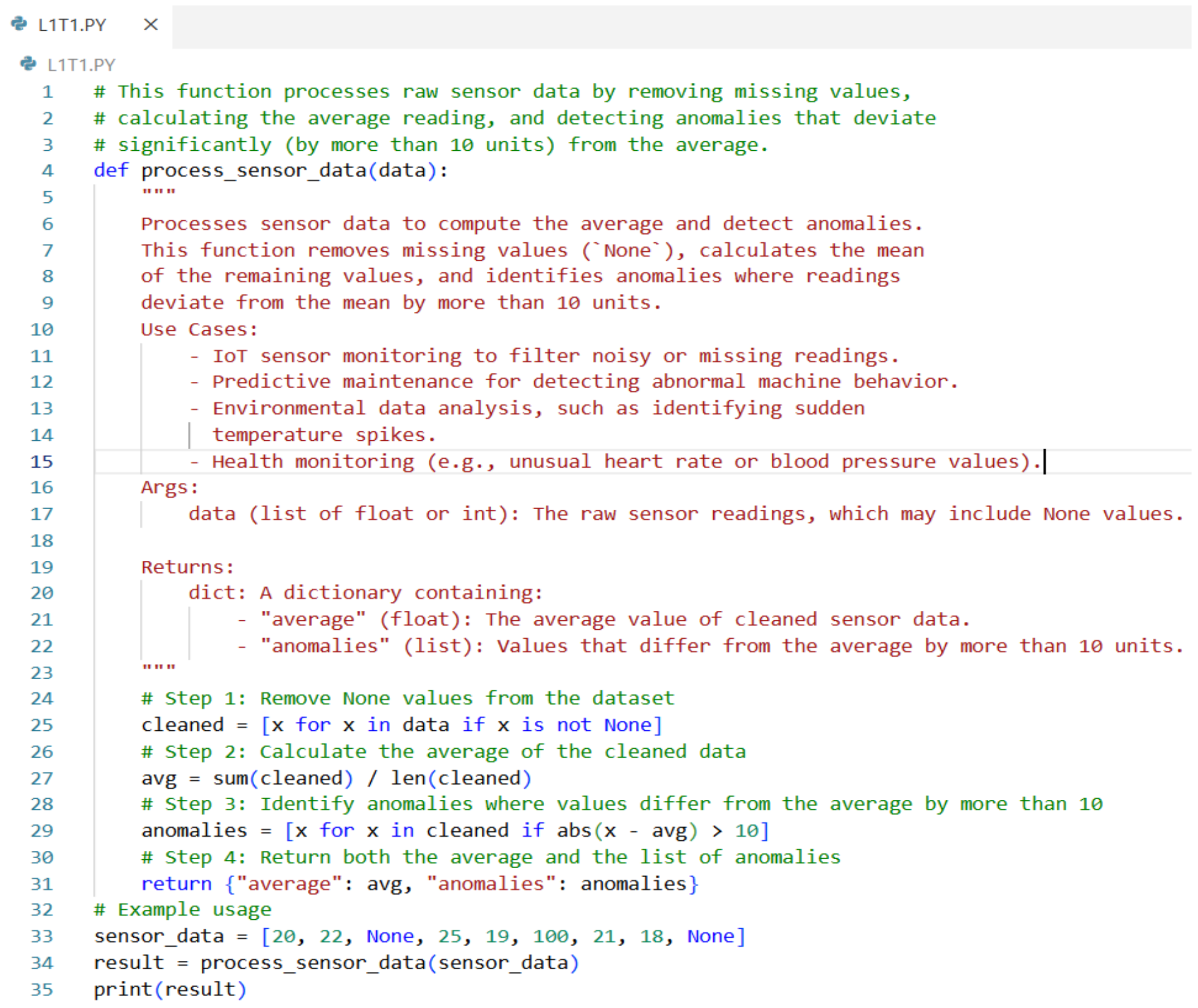
```
def process_sensor_data(data):
cleaned = [x for x in data if x is not None]
avg = sum(cleaned)/len(cleaned)
anomalies = [x for x in cleaned if abs(x - avg) > 10]
return {"average": avg, "anomalies": anomalies}
```

• 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 1: Review the function process_sensor_data(data).

- Write a 2–3 line summary explaining what the function does.
- Add step-by-step (flow-style) comments inside the function.
- Write a short paragraph describing real-world uses, such as sensor monitoring or anomaly detection.

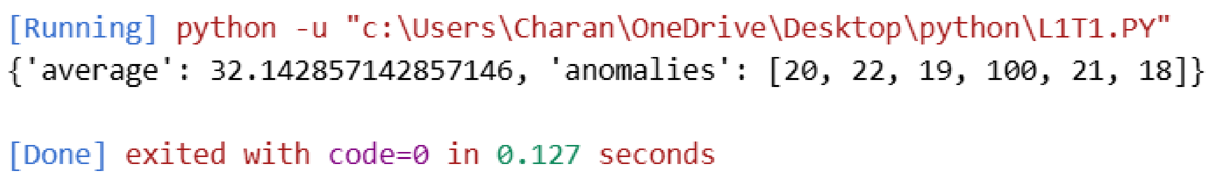
CODE GENERATED:



L1T1.PY

```python
# This function processes raw sensor data by removing missing values,
# calculating the average reading, and detecting anomalies that deviate
# significantly (by more than 10 units) from the average.
def process_sensor_data(data):
    """
    Processes sensor data to compute the average and detect anomalies.
    This function removes missing values (`None`), calculates the mean
    of the remaining values, and identifies anomalies where readings
    deviate from the mean by more than 10 units.
    Use Cases:
        - IoT sensor monitoring to filter noisy or missing readings.
        - Predictive maintenance for detecting abnormal machine behavior.
        - Environmental data analysis, such as identifying sudden
          temperature spikes.
        - Health monitoring (e.g., unusual heart rate or blood pressure values).
    Args:
        data (list of float or int): The raw sensor readings, which may include None values.

    Returns:
        dict: A dictionary containing:
            - "average" (float): The average value of cleaned sensor data.
            - "anomalies" (list): Values that differ from the average by more than 10 units.
    """
    # Step 1: Remove None values from the dataset
    cleaned = [x for x in data if x is not None]
    # Step 2: Calculate the average of the cleaned data
    avg = sum(cleaned) / len(cleaned)
    # Step 3: Identify anomalies where values differ from the average by more than 10
    anomalies = [x for x in cleaned if abs(x - avg) > 10]
    # Step 4: Return both the average and the list of anomalies
    return {"average": avg, "anomalies": anomalies}
# Example usage
sensor_data = [20, 22, None, 25, 19, 100, 21, 18, None]
result = process_sensor_data(sensor_data)
print(result)
```

OUTPUT:



```
[Running] python -u "c:\Users\Charan\OneDrive\Desktop\python\L1T1.PY"
{'average': 32.142857142857146, 'anomalies': [20, 22, 19, 100, 21, 18]}

[Done] exited with code=0 in 0.127 seconds
```

OBSERVATION:

In this assignment, I reviewed the function process_sensor_data(data) and created a concise summary explaining its purpose: to clean a dataset, calculate the average, and detect anomalies. I added flow-style comments to explain each step of the function, making it easier to understand how the data is processed. Additionally, I wrote a short paragraph describing real-world use cases, such as monitoring sensor readings, detecting abnormal values in IoT systems, or analyzing data for predictive maintenance.

This exercise highlighted the importance of clear documentation and comments, especially in long functions, as it improves code readability, maintainability, and helps other developers quickly understand the logic and potential applications of the function.

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 1:

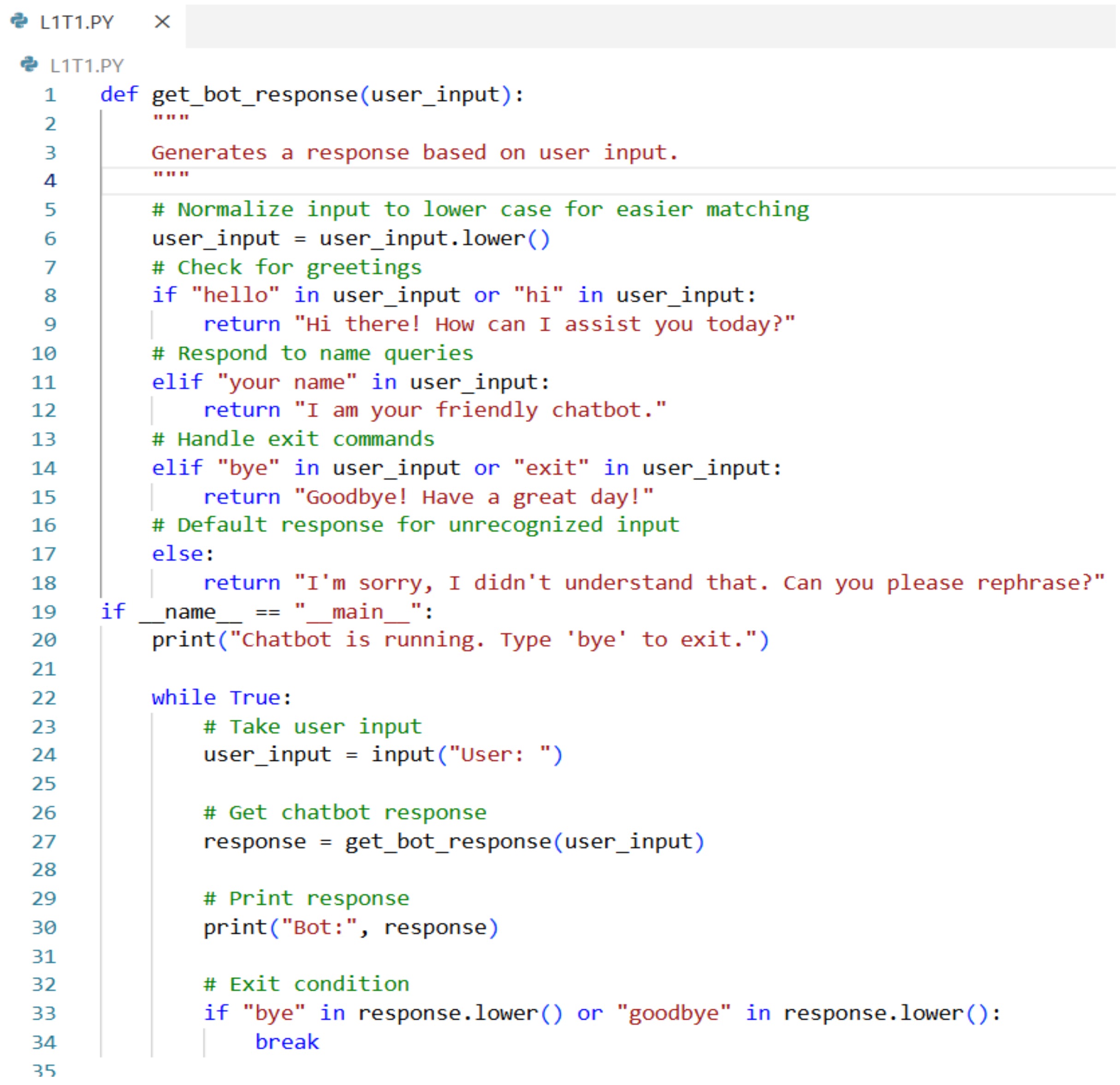
Create a **README.md** file for the project, including: project description, installation steps, usage instructions, and an example.

Add **inline comments** in the main Python script, explaining the logic of the code (not trivial lines).

Use an **AI-assisted tool** (or simulate it) to generate a plain English **usage guide** from your code comments.

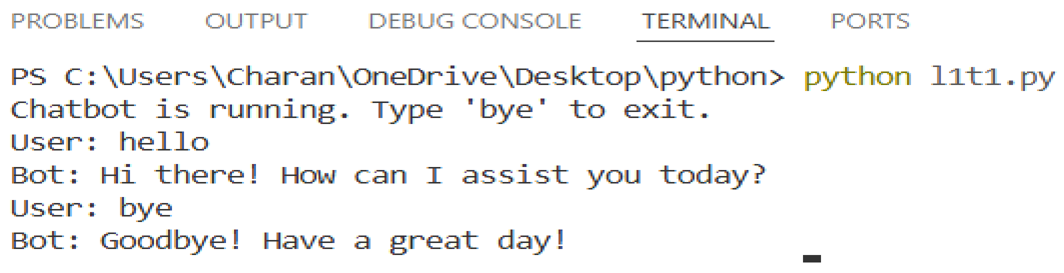
Write a short **reflection** on how automated documentation is helpful in real-time projects compared to manual documentation.

CODE GENERATED:



```python
def get_bot_response(user_input):
    """
    Generates a response based on user input.
    """
    # Normalize input to lower case for easier matching
    user_input = user_input.lower()
    # Check for greetings
    if "hello" in user_input or "hi" in user_input:
        return "Hi there! How can I assist you today?"
    # Respond to name queries
    elif "your name" in user_input:
        return "I am your friendly chatbot."
    # Handle exit commands
    elif "bye" in user_input or "exit" in user_input:
        return "Goodbye! Have a great day!"
    # Default response for unrecognized input
    else:
        return "I'm sorry, I didn't understand that. Can you please rephrase?"
if __name__ == "__main__":
    print("Chatbot is running. Type 'bye' to exit.")

    while True:
        # Take user input
        user_input = input("User: ")

        # Get chatbot response
        response = get_bot_response(user_input)

        # Print response
        print("Bot:", response)

        # Exit condition
        if "bye" in response.lower() or "goodbye" in response.lower():
            break

```

OUTPUT:



```
PS C:\Users\Charan\OneDrive\Desktop\python> python l1t1.py
Chatbot is running. Type 'bye' to exit.
User: hello
Bot: Hi there! How can I assist you today?
User: bye
Bot: Goodbye! Have a great day!
```

OBSERVATION: In this assignment, I created a README.md file for the Chatbot project that included a project description, installation steps,usage instructions, and an example. I also added inline comments in the main Python script to explain the logic of the code, making it easier for others to understand and maintain. Using an AI-assisted tool, I generated a plain English usage guide from the code comments, which provided a
clear, user-friendly reference.

Through this task, I observed that automated documentation significantly improves efficiency in real-time projects by quickly producing consistent and readable guides, reducing errors, and saving time compared to manually writing all documentation.