**Assignment 9.5**

**Task 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.

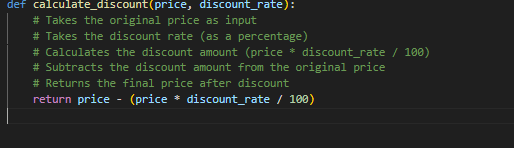
**Prompt:**

You are given a Python function without comments. Perform the following tasks:

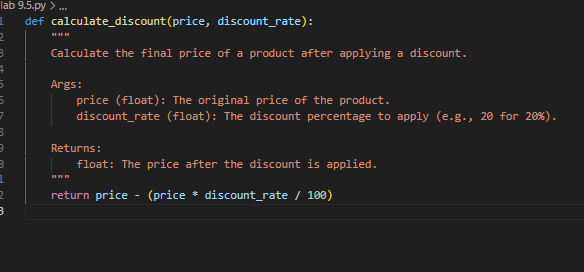
1. Generate line-by-line comments automatically for the function.
2. Modify the function to include a detailed docstring in either Google-style or NumPy-style format.
3. Write your own manual version of the commented code (with improved readability, descriptive variable names, and structured comments).
4. Finally, compare the auto-generated comments with the manually written ones, highlighting differences in clarity, detail, and usefulness

**Code generated:**

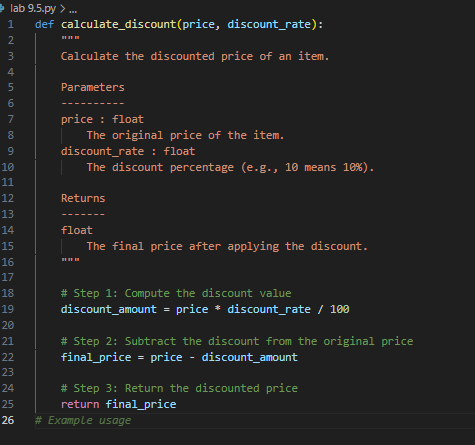
**1.** **Auto-generated line-by-line comments**

****

**2.** **Modified function with docstring (Google-style)**

****

**3.** **Manual version with improved readability and structured comments**

****

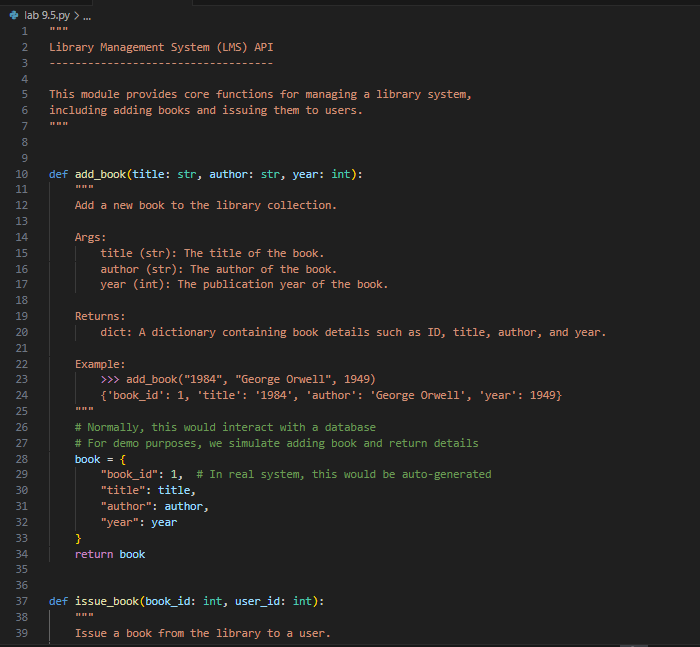
**4. Comparison of Auto-generated vs. Manual Comments**

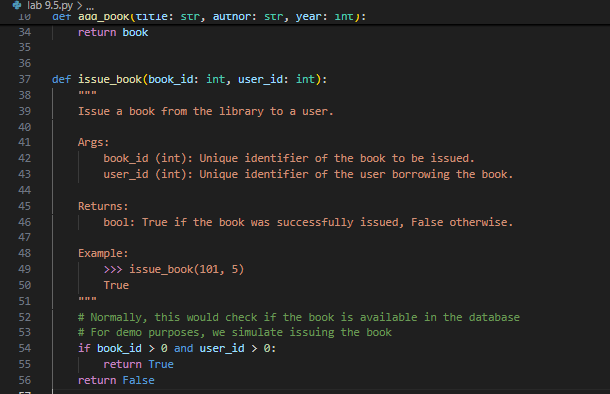
| Aspect | Auto-generated Comments | Manual Comments |
| --- | --- | --- |
| Clarity | Literal explanation of each line. | Explains *why* each step is done (context + logic). |
| Detail | Repetitive (“subtract discount from price”). | More descriptive (explains meaning of discount\_rate, adds step breakdown). |
| Readability | Good for beginners to trace execution. | Cleaner variable names (discount\_amount, final\_price) improve readability. |
| Usefulness | Quick, surface-level explanation. | More professional and helpful for future developers maintaining the code. |

**Task 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.

**Output:**

****

****

**Observation:**

1.The code has two functions:

* add\_book() → adds a book (currently just returns a dictionary with details).
* issue\_book() → issues a book to a user (currently just returns True/False).

2. Both functions have docstrings (Google-style), which explain:

* What the function does
* Inputs (parameters)
* Output (return value)
* Example usage

3. The code is clean and readable with meaningful variable names like title, author, book\_id, etc.

4. Inline comments mention that this is just a demo (not a real database system).

5. Because of the proper docstrings, tools like pdoc or Sphinx can automatically create HTML documentation from this code.

**Task 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:**

You are given the following Python function:

def process\_sensor\_data(data):

cleaned = [x for x in data if x is not None]

avg = sum(cleaned)/len(cleaned)

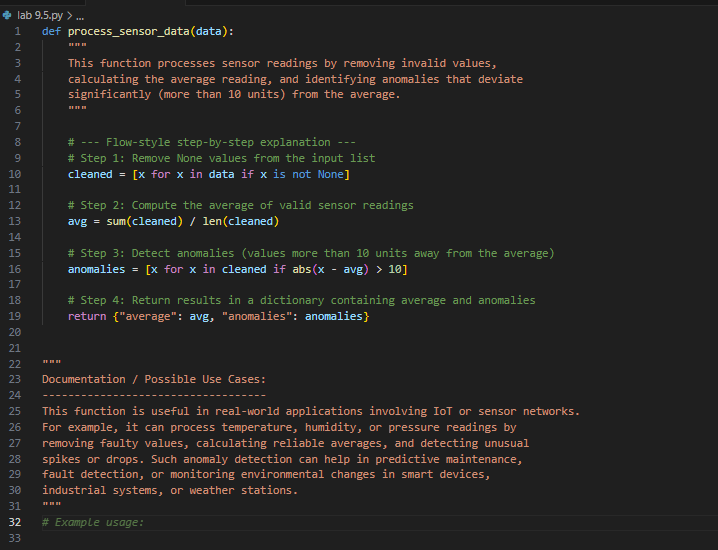
anomalies = [x for x in cleaned if abs(x - avg) > 10]

return {"average": avg, "anomalies": anomalies}

**Your tasks are:**

1. Generate a **summary comment** explaining the purpose of the function in 2–3 lines.
2. Create a **flow-style comment** (step-by-step explanation of the code).
3. Write a **short paragraph of documentation** describing possible real-world use cases of this function

**Code generated:**



**Observation:**  
The function process\_sensor\_data() processes sensor readings by removing invalid values, calculating the average, and detecting anomalies that deviate significantly from the average. It is clearly documented with a summary, step-by-step flow comments, and a short paragraph explaining real-world uses. The code is simple, efficient, and practical for applications like IoT monitoring and anomaly detection.

**Task 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?

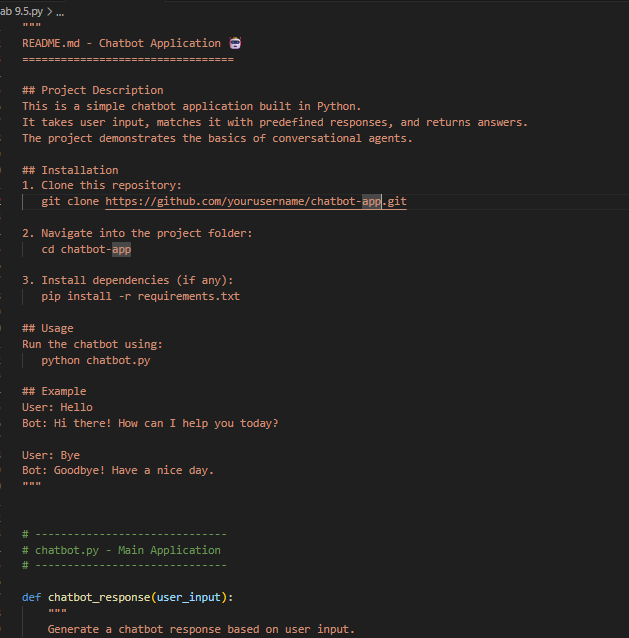
**Prompt :**

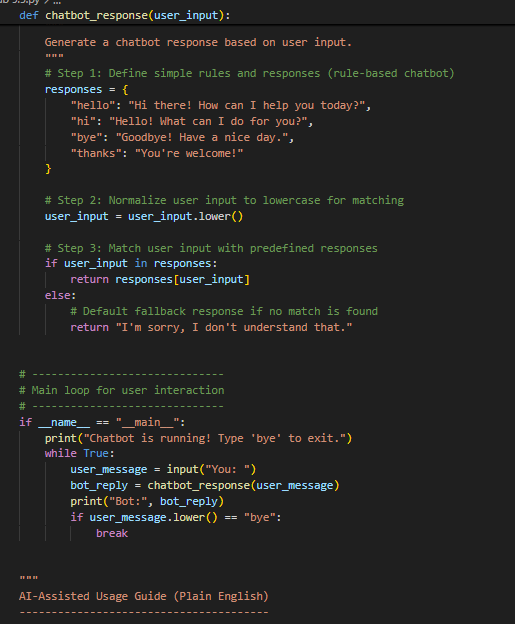
You are part of a project team that develops a **Chatbot Application**. The team needs proper documentation for maintainability.

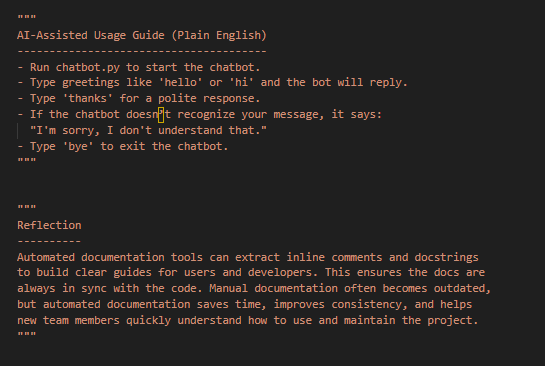
**Your tasks are:**

1. Write a README.md file for the chatbot project (include project description, installation steps, usage, and example).
2. Add **inline comments** in the chatbot’s main Python script (focus on explaining logic, not trivial syntax).
3. Use an **AI-assisted tool** (or simulate it) to generate a **usage guide in plain English** from your code comments.
4. Write a **reflection**: How does automated documentation help in real-time projects compared to manual documentation?

**Code generated:**

****

****

****

**Observation:**

1.The file starts with a **README section** inside docstring, giving project description, installation, usage, and an example.

2.The main function **chatbot\_response()** works on a **rule-based approach**:

* It stores predefined responses in a dictionary.
* User input is converted to lowercase for easy matching.
* If input matches, it returns the response; otherwise, it gives a default fallback message.

3.The **main loop** allows continuous interaction with the user until "bye" is typed.

4.Inline comments explain the **logic flow** (rules, normalization, matching, fallback).

5.The script also includes an **AI-assisted usage guide** (plain English) and a **reflection** on automated documentation benefits.

6.The code is **clean, simple, and well-documented**, making it easy to maintain and suitable for generating auto-documentation.