**ASSIGNMENT 9.5**

**B.VAISHNAVI**

**2403A52408**

# Lab 9 – Documentation Generation: Automatic Documentation and Code Comments

Lab Objectives

* Inline comments
* Docstrings
* Auto-documentation tools
* AI-assisted summarization

# Task Description #1 (Automatic Code Commenting)

PROMPT : generate at least 3 assert test cases for is\_strong\_password(password) and implement the validator function.

• Requirements:

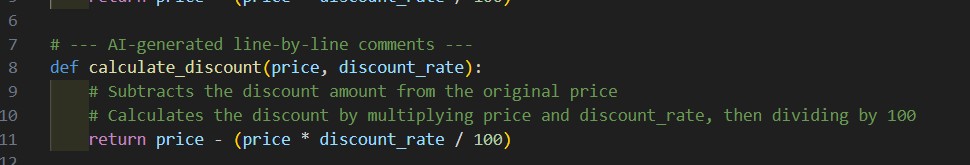
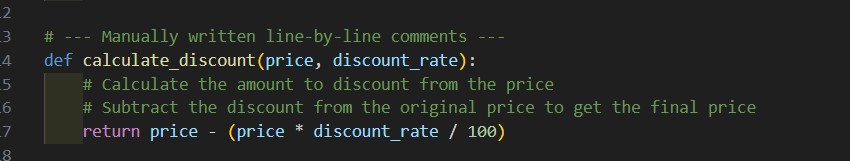
o Password must have at least 8 characters.

o Must include uppercase, lowercase, digit, and special character.

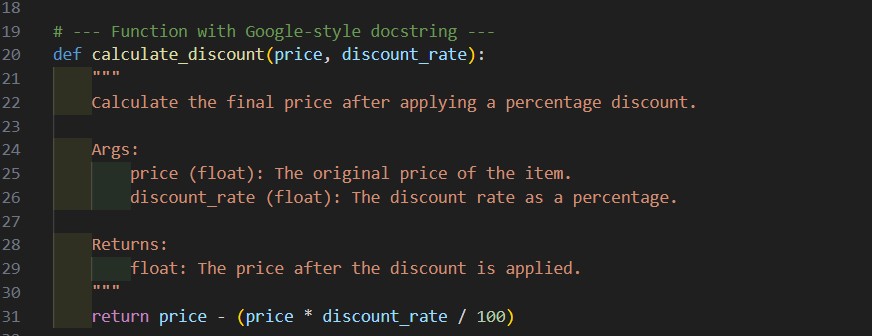
o Must not contain spaces.

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.

# --- Comparison ---

1. AI-generated comments focus on the calculation steps.

2.Manual comments clarify the purpose and the logic.

3.The docstring provides structured documentation for users and tools.

**Conclusion**: AI is helpful for quick annotation, but **manual comments provide better clarity** and domain-specific insight when needed.

# Task Description #2 (API Documentation Generator)

PROMPT: Write detailed docstrings for each function, explaining what they do, their inputs, and outputs. Then use tools like Sphinx or MkDocs to automatically generate clean, well-structured API documentation from these docstrings. Focus on making the docs clear and easy for others to use.

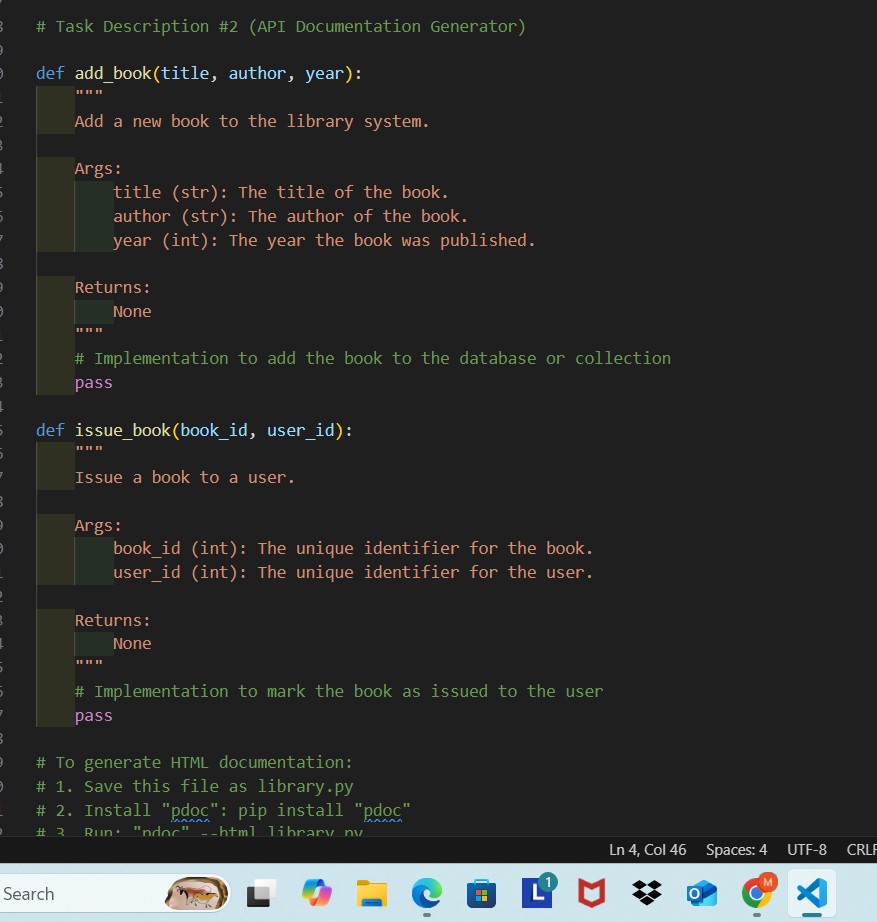
Scenario: A team is building a Library Management System with multiple functions. def add\_book(title, author, year):

# code to add book pass def issue\_book(book\_id, user\_id):

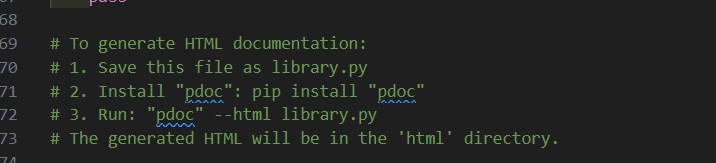
# code to issue book

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.

# Task Description #3 (AI-Assisted Code Summarization)

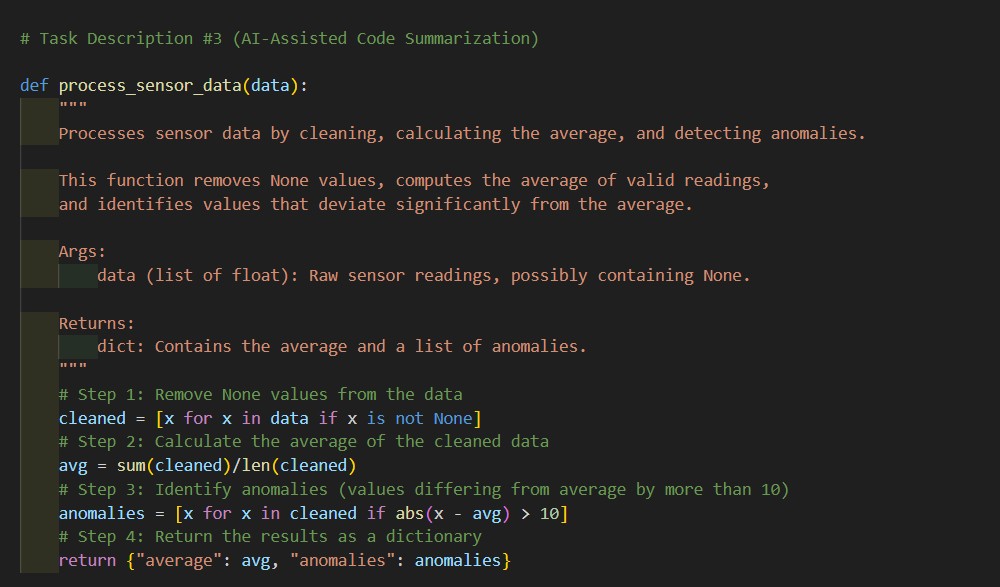
PROMPT: Write brief comments summarizing each function’s goal. Clearly explain each step, like reading data, cleaning it, or outputting results. Include real-world use cases—for example, how the function processes user data or generates reports. Keep it practical and easy for developers to follow

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}

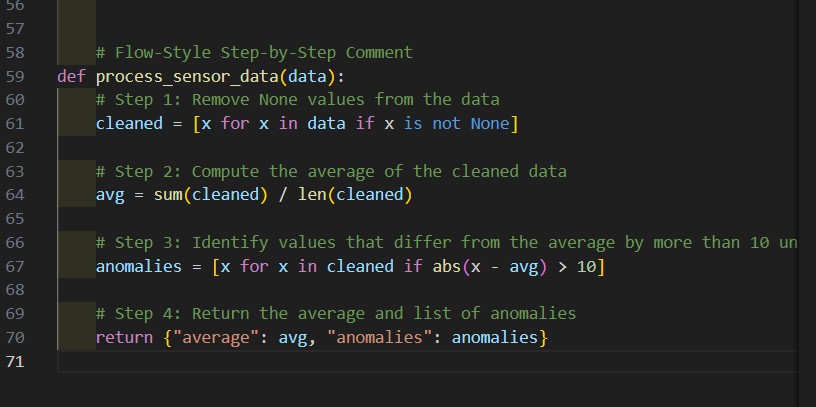


* Generate a summary comment explaining the purpose of the function in 2–3 lines.

1.Processes sensor data by removing invalid entries, computing average,

2.and identifying anomalies that deviate significantly from the average.

* Create a flow-style comment (step-by-step explanation).



# Flow-style comment:

1. Remove None values from input data.
2. Calculate the average of the cleaned data.
3. Find values that differ from the average by more than 10.
4. Return the average and anomalies as a dictionary.

• Write a short paragraph of documentation describing possible use cases of this function in real-world scenarios

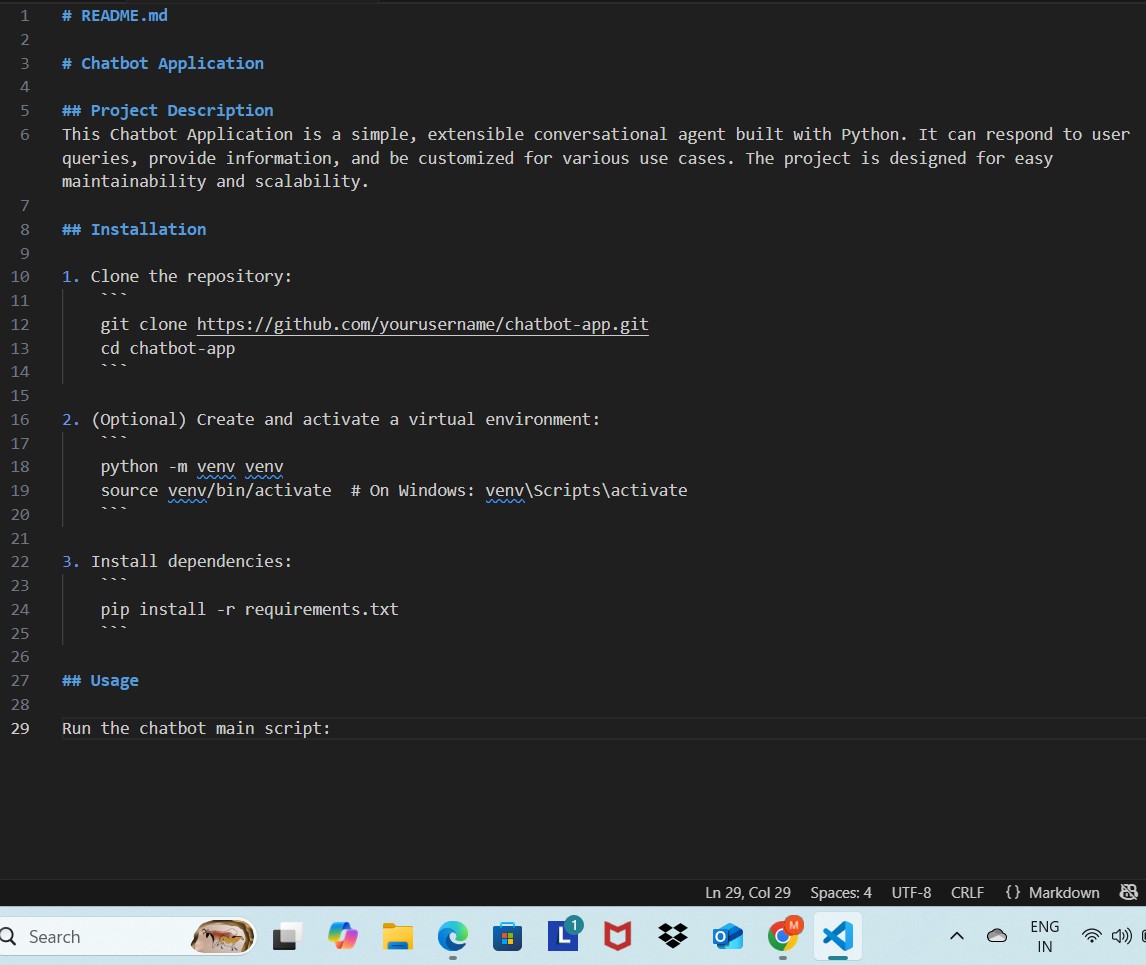
**=>**This function is useful in IoT and industrial systems where sensors may occasionally report invalid or extreme values. It helps in filtering out noise and identifying abnormal behavior such as temperature spikes or equipment malfunctions.

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

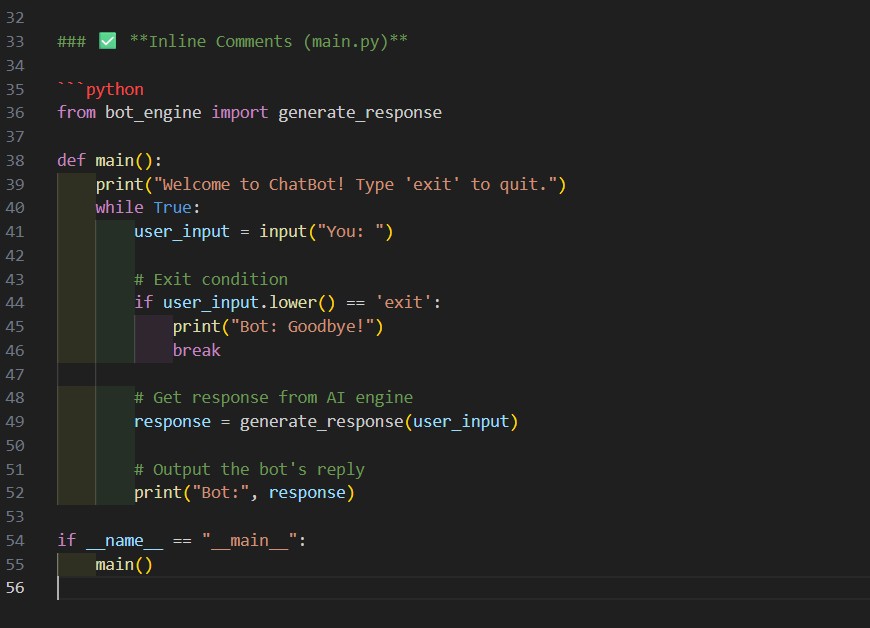
PROMPT: Produce real-time project documentation by creating a detailed README covering setup and usage, writing precise inline comments explaining complex code sections, and generating AI-assisted user guides. Evaluate the accuracy and completeness of automated documentation tools compared to manual documentation, emphasizing maintainability and developer onboarding

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.

**=>**This chatbot starts a conversation with the user. When the user types something, the chatbot generates an AI-based reply. Typing "exit" ends the conversation. All processing happens in the generate\_response() function which contains the core logic for response generation.

* Reflect: How does automated documentation help in real-time projects compared to manual documentation?

**Feature Automated Documentation Manual Documentation**

|  |  |
| --- | --- |
| **Speed** Fast | Time-consuming |
| **Accuracy** May miss context | Context-aware |
| **Consistency** High | May vary by developer |
| **Best For** APIs, simple functions | Complex logic, design decisions |
| **Drawback** Generic language | Requires constant updates |

**Conclusion**: Automated tools accelerate documentation but **must be supplemented by manual context-driven comments**, especially in real-time team projects.