# AI ASSISTED CODING ASSIGNMENT-9.5-1

## ERUKULA SAMPATH

## 2403A51270 BATCH-12

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.

## Code with Manual Line-by-Line Comments:

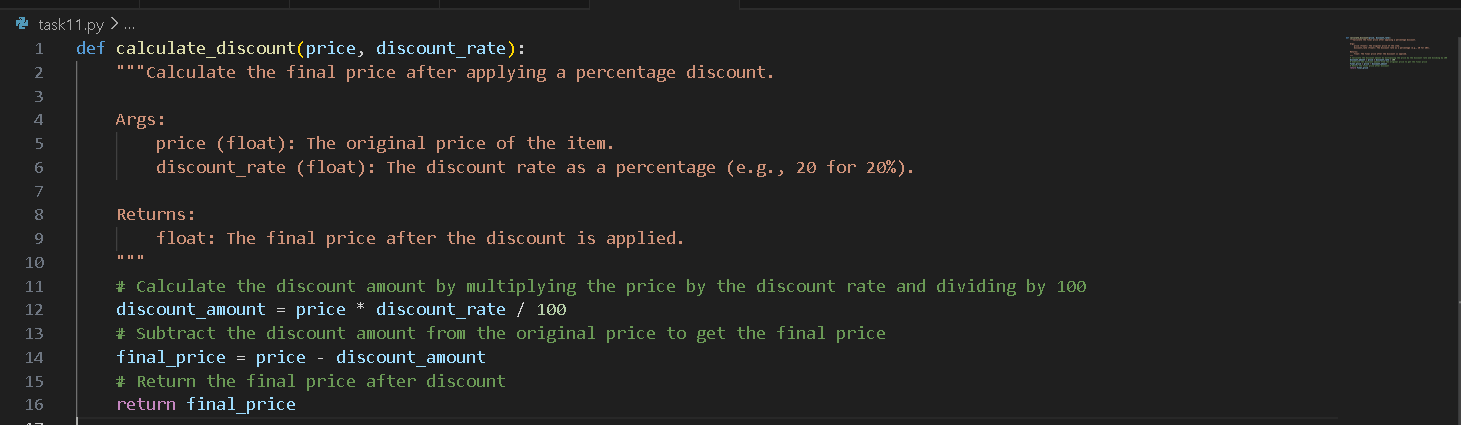
A black background with green text

AI-generated content may be incorrect.

## Prompt:

TASK: Add detailed line-by-line comments and a Google-style docstring to the following function that calculates the final price after discount.

## Updated code:



## Comparing:

| **Aspect** | **AI-Generated** | **Manual** |
| --- | --- | --- |
| **Clarity** | Focuses on describing the math expression literally. | Explains the purpose in a slightly more human-friendly way (“final price”). |
| **Detail** | Mentions subtraction and the formula explicitly. | Mentions “calculate discount” and “final price” more conceptually. |
| **Tone** | More technical/stepwise. | More explanatory/business-friendly. |

**Conclusion:**  
AI comments are good for literal line explanations, but manual comments can emphasize **intent** and **business meaning**.  
Both together help code reviewers understand the “what” and the “why.”

Task 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:

TASK: Add Google-style docstrings to the following functions with input, output, and description:

## Code:

A screenshot of a computer program

AI-generated content may be incorrect.

## 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:

TASK: Generate a 2–3 line summary comment explaining the purpose of the function below.Then generate a step-by-step flow-style comment and a short paragraph describing possible real-world use cases.

## Short Paragraph Documentation (Use Cases)

**Possible Use Cases:**  
This function can be used in IoT applications, industrial monitoring systems, or environmental data logging platforms where sensor readings may contain missing values or spikes. By cleaning the data, computing a central tendency, and flagging outliers, engineers or data scientists can quickly identify faulty sensors, unusual conditions, or critical thresholds in real time.

## Code:

A screenshot of a computer program

AI-generated content may be incorrect.

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?

## Main python script:

A screen shot of a computer program

AI-generated content may be incorrect.

A black rectangle with white text

AI-generated content may be incorrect.

Usage Guide (Generated from Comments):  
This chatbot starts in the console and waits for user input.

* Type any message to receive a reply.
* The bot matches your message to predefined responses (case-insensitive).
* If the message isn’t recognized, the bot responds with a default message.
* Type “bye” to exit the chatbot.

## Reflection: Automated vs Manual Documentation

**Automated Documentation Benefits:**

* **Consistency:** Comments and docstrings automatically generate up-to-date usage guides.
* **Time-saving:** No need to manually rewrite instructions when code changes.
* **Real-time updates:** Tools like Sphinx, pdoc, or mkdocstrings build docs directly from code comments.
* **Easier onboarding:** New developers can read generated docs and quickly understand functions, parameters, and workflows.

**Manual Documentation Benefits:**

* Allows nuanced, contextual, or business explanations beyond what’s in code.
* Good for tutorials, guides, and higher-level architecture overviews.

**Reflection:** Automated documentation is excellent for function-level and API-level references in **real-time projects**, while manual documentation complements it with **context and design rationale**.