AI ASSISTED CODING

ASSIGNMENT-9.5

Akula Vivek

2403a51258

BATCH-11

Task Description #1 (Automatic Code Commenting)

Scenario: You have been given a Python function without comments.

def calculate_discount(price, discount_rate):

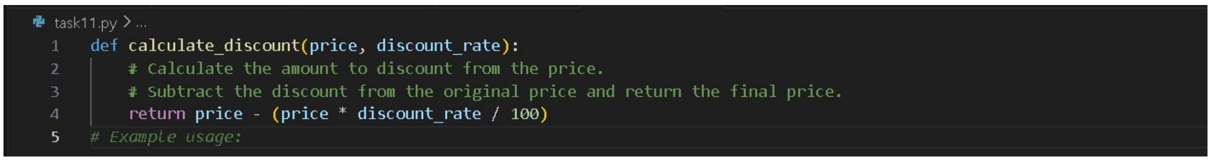
return price - (price * discount_rate / 100)

• 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:



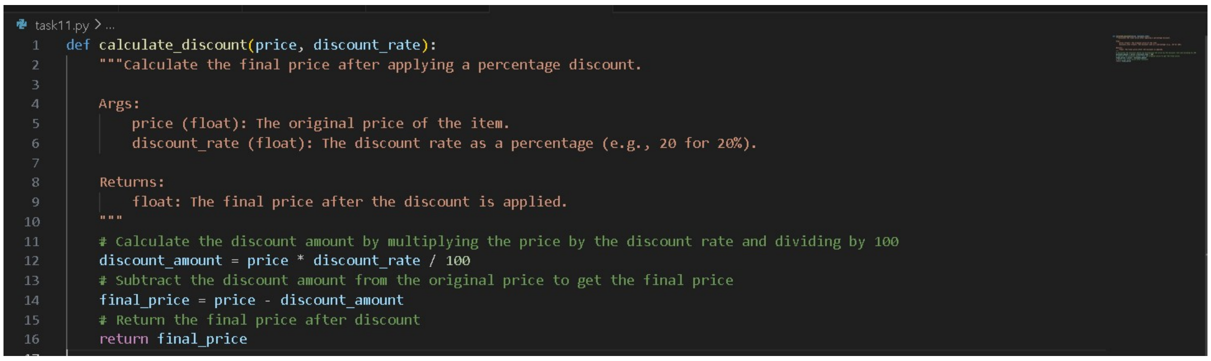
task11.py > ...

```python
def calculate_discount(price, discount_rate):
    # Calculate the amount to discount from the price.
    # Subtract the discount from the original price and return the final price.
    return price - (price * discount_rate / 100)
# Example usage:
```

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:



task11.py > ...

```python
def calculate_discount(price, discount_rate):
    """Calculate the final price after applying a percentage discount.

    Args:
        price (float): The original price of the item.
        discount_rate (float): The discount rate as a percentage (e.g., 20 for 20%).

    Returns:
        float: The final price after the discount is applied.
    """
    # Calculate the discount amount by multiplying the price by the discount rate and dividing by 100
    discount_amount = price * discount_rate / 100
    # Subtract the discount amount from the original price to get the final price
    final_price = price - discount_amount
    # Return the final price after discount
    return final_price
```

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).

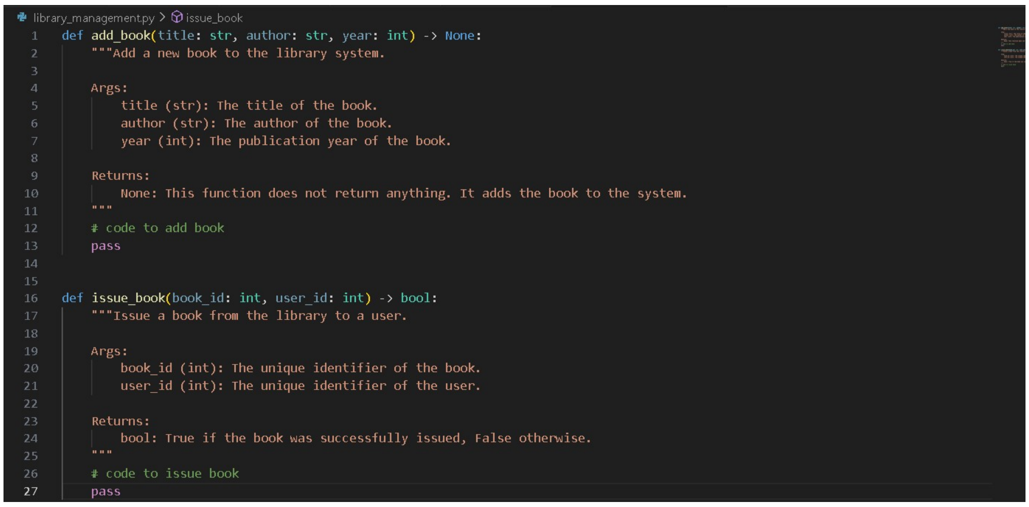
• Use a documentation generator tool (like pdoc, Sphinx, or MkDocs) to automatically create HTML documentation.

• 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:



```python
def add_book(title: str, author: str, year: int) -> None:
    """Add a new book to the library system.

    Args:
        title (str): The title of the book.
        author (str): The author of the book.
        year (int): The publication year of the book.

    Returns:
        None: This function does not return anything. It adds the book to the system.
    """
    # code to add book
    pass


def issue_book(book_id: int, user_id: int) -> bool:
    """Issue a book from the library to a user.

    Args:
        book_id (int): The unique identifier of the book.
        user_id (int): The unique identifier of the user.

    Returns:
        bool: True if the book was successfully issued, False otherwise.
    """
    # code to issue book
    pass
```

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:

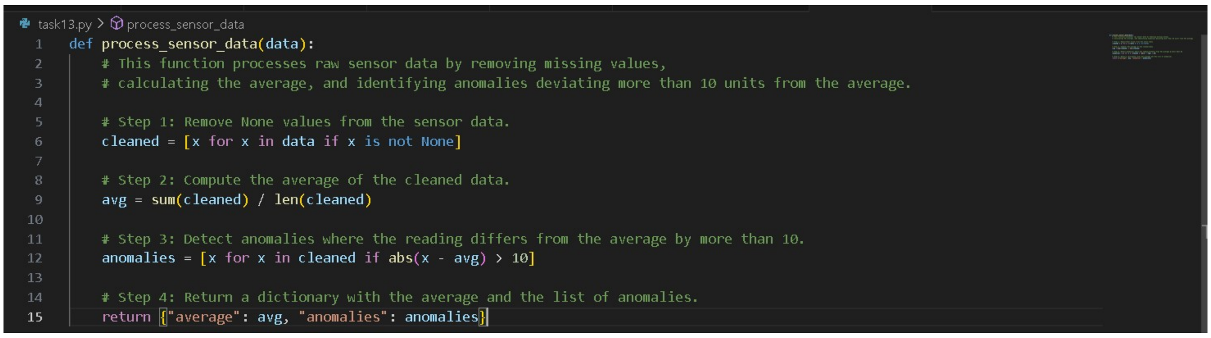
TASK: Generate a 2–3 line summary comment explaining the purpose of the function below.Then generate a step-by-step flow-style comment and a short paragraph describing possible real-world use cases.

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:



task13.py > process_sensor_data

```python
def process_sensor_data(data):
    # This function processes raw sensor data by removing missing values,
    # calculating the average, and identifying anomalies deviating more than 10 units from the average.

    # Step 1: Remove None values from the sensor data.
    cleaned = [x for x in data if x is not None]

    # Step 2: Compute the average of the cleaned data.
    avg = sum(cleaned) / len(cleaned)

    # Step 3: Detect anomalies where the reading differs from the average by more than 10.
    anomalies = [x for x in cleaned if abs(x - avg) > 10]

    # Step 4: Return a dictionary with the average and the list of anomalies.
    return {"average": avg, "anomalies": anomalies}
```

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.

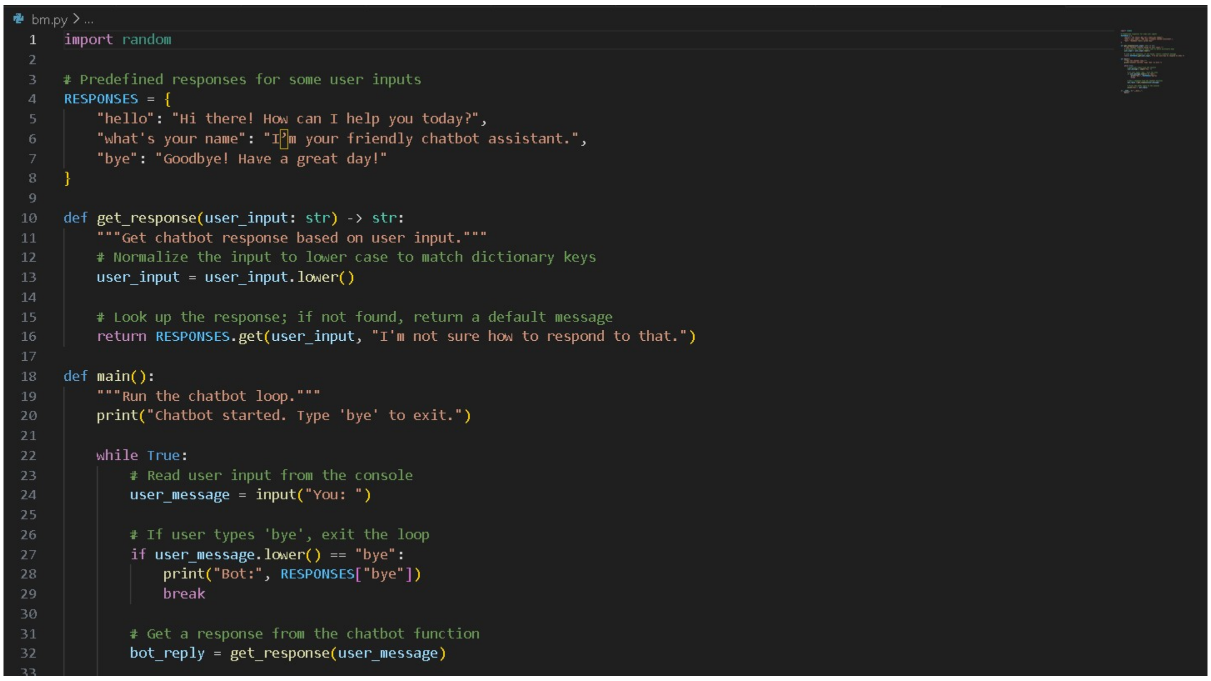
• Write a README.md file for the chatbot project (include project description, installation steps, usage, and example).

• Add inline comments in the chatbot's main Python script (focus on explaining logic, not trivial code).

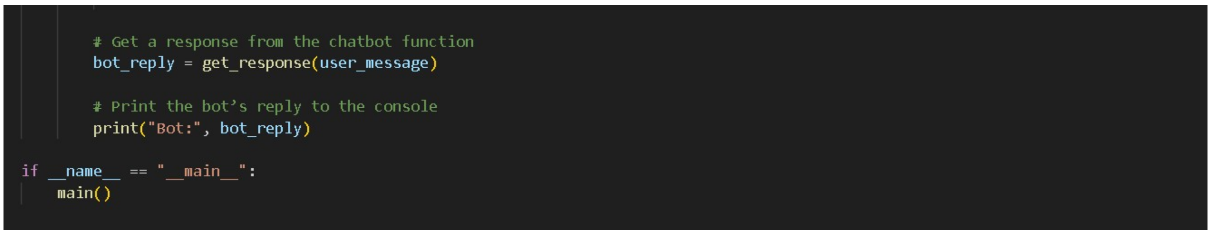
• Use an AI-assisted tool (or simulate it) to generate a usage guide in plain English from your code comments.

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

Main python script:



```
import random

# Predefined responses for some user inputs
RESPONSES = {
    "hello": "Hi there! How can I help you today?",
    "what's your name": "I’m your friendly chatbot assistant.",
    "bye": "Goodbye! Have a great day!"
}

def get_response(user_input: str) -> str:
    """Get chatbot response based on user input."""
    # Normalize the input to lower case to match dictionary keys
    user_input = user_input.lower()

    # Look up the response; if not found, return a default message
    return RESPONSES.get(user_input, "I'm not sure how to respond to that.")

def main():
    """Run the chatbot loop."""
    print("Chatbot started. Type 'bye' to exit.")

    while True:
        # Read user input from the console
        user_message = input("You: ")

        # If user types 'bye', exit the loop
        if user_message.lower() == "bye":
            print("Bot:", RESPONSES["bye"])
            break

        # Get a response from the chatbot function
        bot_reply = get_response(user_message)
```



```
        # Get a response from the chatbot function
        bot_reply = get_response(user_message)

        # Print the bot’s reply to the console
        print("Bot:", bot_reply)

if __name__ == "__main__":
    main()
```

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**.