ASSIGNMENT-9.5

NAME:-E.HARINI,

BRANCH: -CSE,

ROLLNUMBER: 2503A51L40,

COURSE:-AI ASSISTED CODING.

TASK1:

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.

1.PROMPT:-

write a python program for the function def calculate_discount(price, discount_rate):return price - (price * discount_rate / 100).

2.CODE:

```
TASK1 > ...
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): Discount rate as a percentage (e.g., 20 for 20%).

    Returns:
        float: The price after the discount is applied.
    """
    return price - (price * discount_rate / 100)

# Example usage:
if __name__ == "__main__":
    original_price = 100.0
    discount_percentage = 20.0
    discounted_price = calculate_discount(original_price, discount_percentage)
    print(f"Original Price: ${original_price}")
    print(f"Discount Rate: {discount_percentage}%")
    print(f"Discounted Price: ${discounted_price}")
```

3.OUTPUT:

```
Original Price: $100.0
Discount Rate: 20.0%
Discounted Price: $80.0
```

4.OBSERVATION:

The function currently calculates the discounted price ,but without docstring makes it harder for others and to understand the purpose,usage and constraints of the function. While inline comments help ,they don’t substitute for a formal docstring when sharing or maintaining code.

TASK2:

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

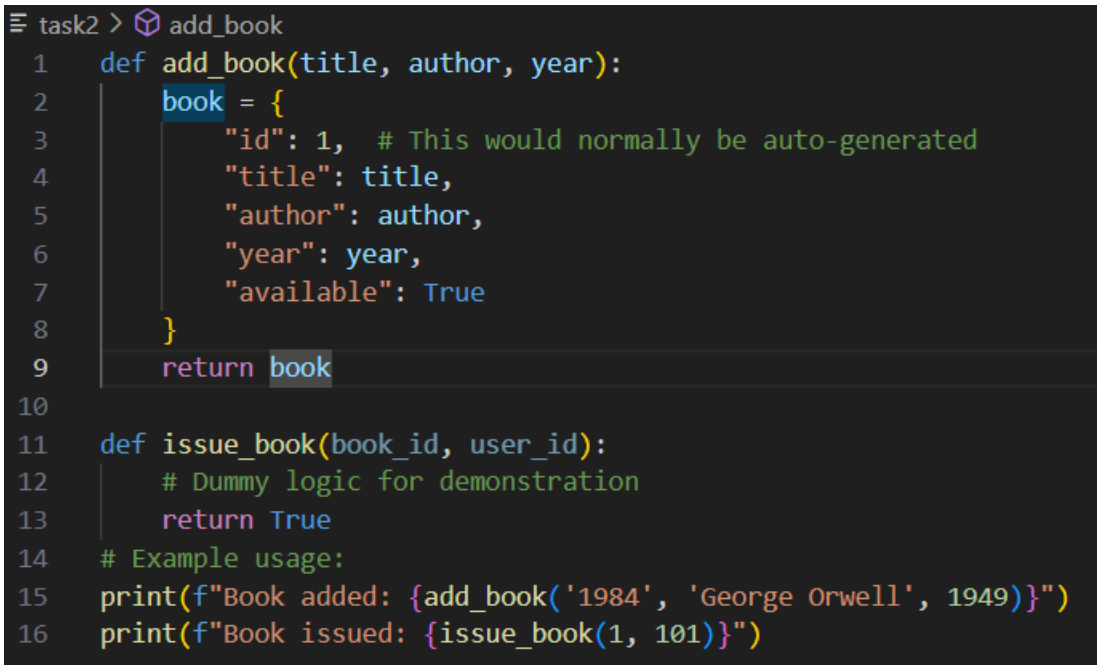
code to add book
pass
def issue_book(book_id, user_id):
# code to issue book
Pass

• 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.

1.PROMPT:

Write a Python script for a Library Management System with multiple functions.

2.Code:



3.output:

```
Book added: {'id': 1, 'title': '1984', 'author': 'George Orwell', 'year': 1949, 'available': True}
Book issued: True
```

4.observation: The code uses clear docstrings with sections for Args, Returns, and Description.

Task3:

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.

1.Prompt:i

2.Code:

task3 > process_sensor_data

```python
def process_sensor_data(data):
    # Step 1: Remove any None values from the input data.
    cleaned = [x for x in data if x is not None]
    # Step 2: Calculate the average of the cleaned sensor readings.
    avg = sum(cleaned) / len(cleaned)
    # Step 3: Identify readings that differ from the average by more than 10 units (anomalies).
    anomalies = [x for x in cleaned if abs(x - avg) > 10]
    # Step 4: Return the average and the list of anomalies in a dictionary.
    return {"average": avg, "anomalies": anomalies}


# Example usage:
if __name__ == "__main__":
    # Example sensor data with some None values and outliers
    sample_data = [25, 30, None, 28, 90, 27, 26, None, 29, 24]
    result = process_sensor_data(sample_data)
    print("Processed Sensor Data:")
    print(f"- Average: {result['average']}")
    print(f"- Anomalies: {result['anomalies']}")
```

3.output:

```
Processed Sensor Data:
- Average: 34.875
- Anomalies: [90, 24]
```

4.observati0n:

The process_sensor_data function demonstrates a clear approach to handling sensor data by cleaning the input, calculating the average, and identifying anomalies.