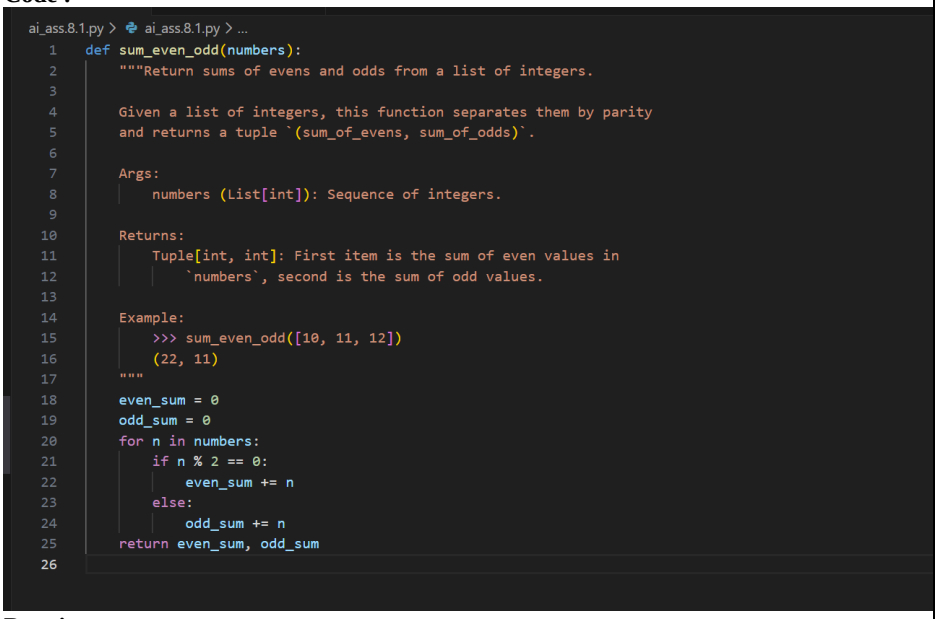
SCHOOLOFCOMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE		DEPARTMENTOFCOMPUTER SCIENCE ENGINEERING	
Program Name: B. Tech		**Assignment Type: Lab**	**Academic Year:**2025-2026
Course Coordinator Name		Dr. Rishabh Mittal	
CourseCode	23CS002PC304	**Course Title**	AI Assisted Coding
Year/Sem	III/II	**Regulation**	R23
Date and Day of Assignment	**Week3 – Wednesday**	**Time(s)**	23CSBTB01 To 23CSBTB52
Name	D.udayan	**Batche**	48
AssignmentNumber:9.3(Presentassignmentnumber)/**24**(Totalnumberofassignments)			
H.NO:2303A54017			

Q.No.	Question	*ExpectedTime to complete*
	Lab 9: Documentation Generation – Automatic Documentation and Code Comments **Lab Objectives** • To understand the importance of documentation and code comments in software development • To explore how AI-assisted coding tools generate documentation and inline comments • To practice generating function-level and module-level docstrings automatically • To evaluate the quality and accuracy of AI-generated documentation • To develop a small automated documentation generator in Python	
1	**Lab Outcomes (LOs)** After completing this lab, students will be able to: • Apply AI-assisted coding tools to generate docstrings and inline comments • Analyze AI-generated documentation for correctness and readability • Create structured documentation using standard formats (Google, NumPy) • Design a mini documentation generation tool **Task 1: Basic Docstring Generation** **Scenario** You are developing a utility function that processes numerical lists and must be properly documented for future maintenance. **Prompt : Create a Python function named sum_even_odd(numbers) that takes a list of integers and returns a tuple containing the sum of even numbers and sum of odd numbers.** **1. First, write the function with a manually written Google Style docstring including:** **- Description** **- Args** **- Returns** **- Example** **2. Then generate an AI-style Google docstring for the same function separately (without changing logic).**	Week4 - Wednesday

3. Provide a structured comparison analyzing:
- **Clarity**
- **Correctness**
- **Completeness**
- **Readability**

Ensure the code runs without errors.

Code :



Requirements

• Write a Python function to return the **sum of even numbers** and **sum of odd numbers** in a given list

• Manually add a **Google Style docstring** to the function

• Use an AI-assisted tool (Copilot / Cursor AI) to generate a function-level docstring

• Compare the **AI-generated docstring** with the **manually written docstring**

• Analyze clarity, correctness, and completeness

Expected Output

• Python function with manual Google-style docstring

• AI-generated docstring for the same function

• Comparison explaining differences between manual and AI-generated documentation

• Improved understanding of AI-generated function-level documentation

Explanation : In this task, we create a function `sum_even_odd(numbers)` that finds the sum of even and odd numbers from a list and returns them as a tuple. The main focus is writing Google Style docstrings manually and then generating an AI-style docstring for the same function. Finally, both docstrings are compared based on clarity, correctness, completeness, and readability.

Task 2: Automatic Inline Comments

Scenario

You are developing a student management module that must be easy to understand for new developers.

Prompt : Create a Python class named sru_student with:

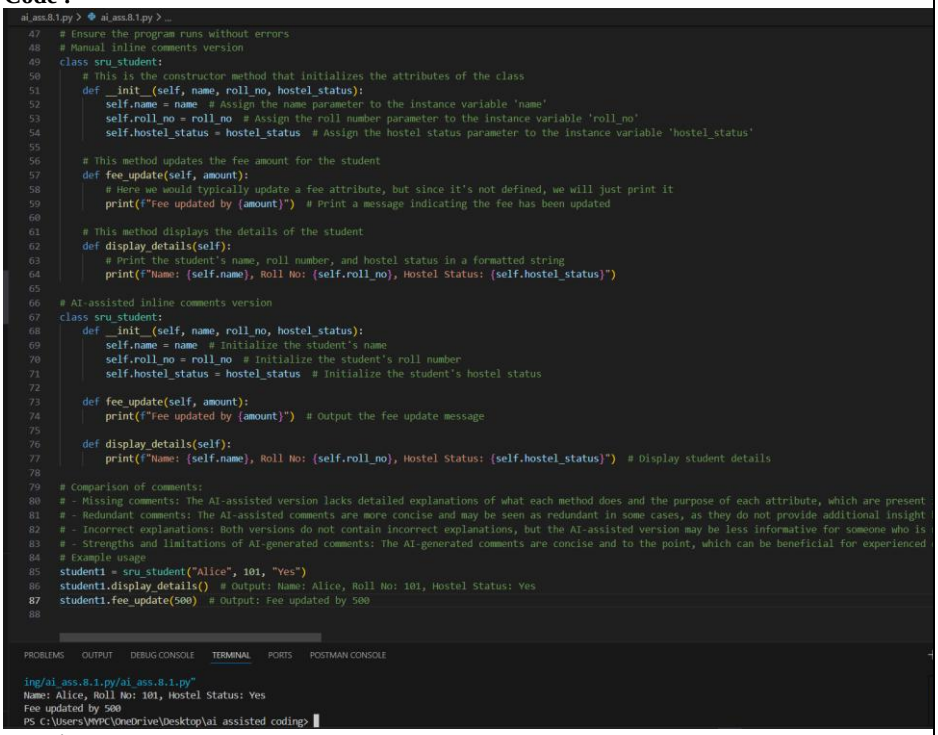
Attributes:
- **name**
- **roll_no**
- **hostel_status**

Methods:
- fee_update(amount)
- display_details()

1. First, write the class with detailed manual inline comments explaining each line or logical block.
2. Then generate an AI-assisted version of inline comments for the same code (without changing logic).
3. Provide a comparison discussing:
- Missing comments
- Redundant comments
- Incorrect explanations
- Strengths and limitations of AI-generated comments

Ensure the program runs without errors.

Code :



```
ai_ass.8.1.py > 🔷 ai_ass.8.1.py > ...
47  # Ensure the program runs without errors
48  # Manual inline comments version
49  class sru_student:
50      # This is the constructor method that initializes the attributes of the class
51      def __init__(self, name, roll_no, hostel_status):
52          self.name = name  # Assign the name parameter to the instance variable 'name'
53          self.roll_no = roll_no  # Assign the roll number parameter to the instance variable 'roll_no'
54          self.hostel_status = hostel_status  # Assign the hostel status parameter to the instance variable 'hostel_status'
55
56      # This method updates the fee amount for the student
57      def fee_update(self, amount):
58          # Here we would typically update a fee attribute, but since it's not defined, we will just print it
59          print(f"Fee updated by {amount}")  # Print a message indicating the fee has been updated
60
61      # This method displays the details of the student
62      def display_details(self):
63          # Print the student's name, roll number, and hostel status in a formatted string
64          print(f"Name: {self.name}, Roll No: {self.roll_no}, Hostel Status: {self.hostel_status}")
65
66  # AI-assisted inline comments version
67  class sru_student:
68      def __init__(self, name, roll_no, hostel_status):
69          self.name = name  # Initialize the student's name
70          self.roll_no = roll_no  # Initialize the student's roll number
71          self.hostel_status = hostel_status  # Initialize the student's hostel status
72
73      def fee_update(self, amount):
74          print(f"Fee updated by {amount}")  # Output the fee update message
75
76      def display_details(self):
77          print(f"Name: {self.name}, Roll No: {self.roll_no}, Hostel Status: {self.hostel_status}")  # Display student details
78
79  # Comparison of comments:
80  # - Missing comments: The AI-assisted version lacks detailed explanations of what each method does and the purpose of each attribute, which are present
81  # - Redundant comments: The AI-assisted comments are more concise and may be seen as redundant in some cases, as they do not provide additional insight
82  # - Incorrect explanations: Both versions do not contain incorrect explanations, but the AI-assisted version may be less informative for someone who is
83  # - Strengths and limitations of AI-generated comments: The AI-generated comments are concise and to the point, which can be beneficial for experienced
84  # Example usage
85  student1 = sru_student("Alice", 101, "Yes")
86  student1.display_details()  # Output: Name: Alice, Roll No: 101, Hostel Status: Yes
87  student1.fee_update(500)  # Output: Fee updated by 500
88

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  POSTMAN CONSOLE

ing/ai_ass.8.1.py/ai_ass.8.1.py"
Name: Alice, Roll No: 101, Hostel Status: Yes
Fee updated by 500
PS C:\Users\MYPC\OneDrive\Desktop\ai assisted coding>
```

Requirements
• Write a Python program for an sru_student class with the following:
– Attributes: name, roll_no, hostel_status
– Methods: fee_update() and display_details()
• Manually write **inline comments** for each line or logical block
• Use an AI-assisted tool to automatically add inline comments
• Compare **manual comments** with **AI-generated comments**
• Identify missing, redundant, or incorrect AI comments
Expected Output
• Python class with manually written inline comments
• AI-generated inline comments added to the same code
• Comparative analysis of manual vs AI comments
• Critical discussion on strengths and limitations of AI-generated comments

Explanation : In this task, we create a class `sru_student` with attributes like name, roll number, and hostel status, and methods to update fees and display details. First, the code is written with detailed manual inline comments. Then AI-generated comments are written for the same code. Finally, we compare

both comment styles and analyze missing, redundant, and incorrect comments.

Task 3: Module-Level and Function-Level Documentation
Scenario
You are building a small calculator module that will be shared across multiple projects and requires structured documentation.
Prompt: Create a Python calculator module containing four functions:
- add(a, b)
- subtract(a, b)
- multiply(a, b)
- divide(a, b)

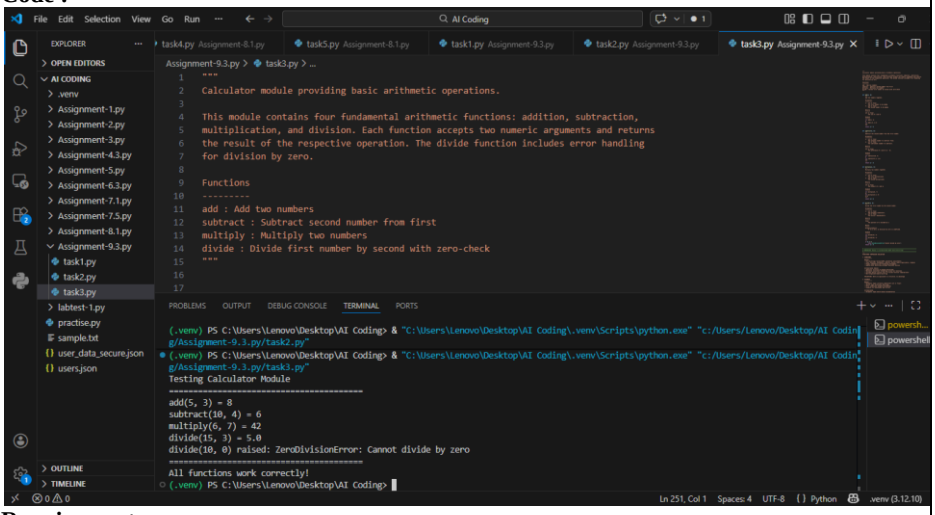
1. First, manually write NumPy Style docstrings for each function including:
- Parameters
- Returns
- Raises (for divide by zero)
- Example

2. Then generate:
- A module-level docstring
- AI-generated NumPy Style function-level docstrings (without modifying function logic)

3. Provide a structured comparison evaluating:
- Structure
- Accuracy
- Completeness
- Readability
- Professional quality

Ensure the code runs without errors.

Code :



Requirements
• Write a Python script containing **3–4 functions** (e.g., add, subtract, multiply, divide)
• Manually write **NumPy Style docstrings** for each function
• Use AI assistance to generate:
– A **module-level docstring**
– Individual **function-level docstrings**
• Compare AI-generated docstrings with manually written ones
• Evaluate documentation structure, accuracy, and readability
Expected Output
• Python script with manual NumPy-style docstrings

	• AI-generated module-level and function-level documentation • Comparison between AI-generated and manual documentation • Clear understanding of structured documentation for multi-function scripts **Explanation :** In this task, we create a calculator module with functions add, subtract, multiply, and divide. Manual NumPy Style docstrings are written for each function, including parameters, returns, raises, and examples. Then AI-generated docstrings and a module-level docstring are created. Finally, we compare manual vs AI docstrings based on structure, accuracy, completeness, readability, and professional quality.	
	Additional Requirement • Push the complete project documentation as a **.md file** to a GitHub repository • Ensure documentation covers module overview and function descriptions **Note: Report should be submitted a word document for all tasks in a single document with prompts, comments & code explanation, and output and if required, screenshots**	