

SCHOOL OF COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE		DEPARTMENT OF COMPUTER SCIENCE ENGINEERING			
Program Name: B. Tech		Assignment Type: Lab			
Course Coordinator Name		Dr. Rishabh Mittal			
CourseCode	23CS002PC304	Course Title	AI Assisted Coding		
Year/Sem	III/II	Regulation	R23		
Date and Day of Assignment	Week 3 – Wednesday	Time(s)	23CSBTB01 To 23CSBTB52		
Name	sushma	Applicable to Batches	All batches		
Assignment Number: 9.3(Present assignment number)/24(Total number of assignments)					
Q.No.	Question				
1	<p>Lab 9: Documentation Generation – Automatic Documentation and Code Comments</p> <p>Lab Objectives</p> <ul style="list-style-type: none"> 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 <p>Lab Outcomes (LOs)</p> <p>After completing this lab, students will be able to:</p> <ul style="list-style-type: none"> 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 <p>Task 1: Basic Docstring Generation</p> <p>Scenario</p> <p>You are developing a utility function that processes numerical lists and must be properly documented for future maintenance.</p> <p>Prompt : Create a Python function named <code>sum_even_odd(numbers)</code> that takes a list of integers and returns a tuple containing the sum of even numbers and sum of odd numbers.</p> <ol style="list-style-type: none"> First, write the function with a manually written Google Style docstring including: <ul style="list-style-type: none"> Description Args Returns Example Then generate an AI-style Google docstring for the same function separately (without changing logic). 				

- 3. Provide a structured comparison analyzing:**

 - Clarity
 - Correctness
 - Completeness
 - Readability

Ensure the code runs without errors.

Code :

The screenshot shows a VS Code interface with the following details:

- File Explorer:** Shows files like `task2.py`, `task3.py`, `task4.py`, `task5.py`, and `task1.py`. `task1.py` is currently selected.
- Code Editor:** Displays the content of `task1.py`. The code defines a function `sum_even_odd(numbers)` that calculates the sum of even and odd numbers in a list. It includes docstrings and examples.
- Terminal:** Shows command-line output for running `task1.py` in a virtual environment. The output includes:
 - PS C:\Users\Lenovo\Desktop\AI Coding & "C:\Users\Lenovo\Desktop\AI Coding\venv\Scripts\python.exe" "c:/Users/Lenovo/Desktop/AI Coding/g/Assignment-9.1.py\task1.py"
 - (venv) PS C:\Users\Lenovo\Desktop\AI Coding & "C:\Users\Lenovo\Desktop\AI Coding\venv\Scripts\python.exe" "c:/Users/Lenovo/Desktop/AI Coding/g/Assignment-9.1.py\task1.py"
- Bottom Status Bar:** Shows file paths, line numbers (Ln 119 Col 1), and other status information.

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 :

```

sru_student:
    # Class definition for SRU student with essential attributes and methods
    def __init__(self, name, roll_no, hostel_status):
        # Constructor method to initialize a student object
        # Parameters:
        # - name: Student's full name (string)
        # - roll_no: Student's roll number (Integer or string)
        # - hostel_status: Whether student lives in hostel (boolean or string)

        self.name = name # Store the student's name in instance variable
        self.roll_no = roll_no # Store the student's roll number
        self.hostel_status = hostel_status # Store hostel residency status

```

TESTING MANUALLY COMMENTED VERSION:

```

Fee updated: Rs. 50000 added. Total fees paid: Rs. 50000
Fee updated: Rs. 25000 added. Total fees paid: Rs. 75000

```

Student Name: Rajesh Kumar
Roll Number: 101
Hostel Status: Residing in Hostel A
Total Fees Paid: Rs. 75000

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)`

	<p>structure, accuracy, completeness, readability, and professional quality.</p> <hr/> <p>Additional Requirement</p> <ul style="list-style-type: none">• Push the complete project documentation as a .md file to a GitHub repository• Ensure documentation covers module overview and function descriptions <p>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</p>	
--	--	--