

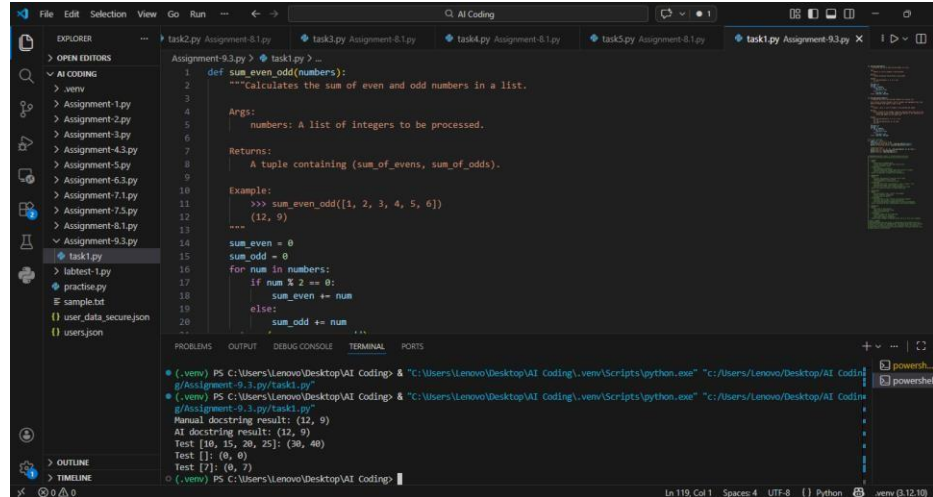
| SCHOOL OF COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE | | DEPARTMENT OF COMPUTER SCIENCE ENGINEERING | |
|---|---|--|--------------------------|
| Program Name: B. Tech | | Assignment Type: Lab | Academic Year: 2025-2026 |
| Course Coordinator Name | | Dr. Rishabh Mittal | |
| | | | |
| Course Code | 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 | Pushpanjali | Applicable to Batches | All batches |
| Assignment Number: 9.3 (Present assignment number) / 24 (Total number of assignments) | | | |
| | | | |
| Q.No. | Question | Expected Time to complete | |
| 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 <hr/> <p>Lab Outcomes (LOs) 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 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 <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> <p>1. First, write the function with a manually written Google Style docstring including:</p> <ul style="list-style-type: none"> - Description - Args - Returns - Example <p>2. Then generate an AI-style Google docstring for the same function separately (without changing logic).</p> | Week 4 - Wednesday | |

3. Provide a structured comparison analyzing:

- Clarity
- Correctness
- Completeness
- Readability

Ensure the code runs without errors.

Code :



```
1 def sum_even_odd(numbers):
2     """Calculates the sum of even and odd numbers in a list.
3
4     Args:
5         numbers: A list of integers to be processed.
6
7     Returns:
8         A tuple containing (sum_of_evens, sum_of_odds).
9
10    Example:
11    >>> sum_even_odd([1, 2, 3, 4, 5, 6])
12    (12, 9)
13    """
14    sum_even = 0
15    sum_odd = 0
16    for num in numbers:
17        if num % 2 == 0:
18            sum_even += num
19        else:
20            sum_odd += num
21    return (sum_even, sum_odd)
```

Terminal Output:

```
(.venv) PS C:\Users\Lenovo\Desktop\AI Coding> & "C:\Users\Lenovo\Desktop\AI Coding\.venv\Scripts\python.exe" "C:\Users\Lenovo\Desktop\AI Coding\Assignment-0.3.py\task1.py"
Manual docstring result: (12, 9)
AI docstring result: (12, 9)
Test [10, 15, 20, 25]: (30, 40)
Test [1]: (0, 0)
Test [7]: (0, 7)
```

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 :

```

1  # =====
2  # PART 1: MANUALLY WRITTEN INLINE COMMENTS VERSION
3  # =====
4
5  class sru_student:
6      # Class definition for SRU student with essential attributes and methods
7
8      def __init__(self, name, roll_no, hostel_status):
9          # Constructor method to initialize a student object
10         # Parameters:
11         # - name: Student's full name (string)
12         # - roll_no: Student's roll number (integer or string)
13         # - hostel_status: Whether student lives in hostel (boolean or string)
14
15         self.name = name # Store the student's name in instance variable
16         self.roll_no = roll_no # Store the student's roll number
17         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)`

- 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 :

The screenshot shows a VS Code editor with a file explorer on the left containing several Python files. The main editor window displays a file named 'task3.py' with the following content:

```
1 """
2 Calculator module providing basic arithmetic operations.
3
4 This module contains four fundamental arithmetic functions: addition, subtraction,
5 multiplication, and division. Each function accepts two numeric arguments and returns
6 the result of the respective operation. The divide function includes error handling
7 for division by zero.
8
9 Functions
10 -----
11 add : Add two numbers
12 subtract : Subtract second number from first
13 multiply : Multiply two numbers
14 divide : Divide first number by second with zero-check
15 """
16
17
```

Below the code editor, there is a 'PROBLEMS' panel showing test results for 'Testing Calculator Module':

```
add(5, 3) = 8
subtract(10, 4) = 6
multiply(6, 2) = 12
divide(15, 3) = 5.0
divide(10, 0) raised: ZeroDivisionError: Cannot divide by zero
```

The terminal window at the bottom shows the command prompt and the output of the tests:

```
(.venv) PS C:\Users\Lenovo\Desktop\AI Coding\ & "C:\Users\Lenovo\Desktop\AI Coding\.venv\Scripts\python.exe" "C:\Users\Lenovo\Desktop\AI Coding\Assignment-9.3.py\task3.py"
Testing Calculator Module
add(5, 3) = 8
subtract(10, 4) = 6
multiply(6, 2) = 12
divide(15, 3) = 5.0
divide(10, 0) raised: ZeroDivisionError: Cannot divide by zero
All functions work correctly!
```

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

| | | |
|--|--|--|
| | <p>structure, accuracy, completeness, readability, and professional quality.</p> | |
| | <div><div>Additional Requirement<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</div><div>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</div></div> | |