|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **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** | | | | Venkataramana Veeramsetty | | | | | |
| **Instructor(s) Name** | | | | |  | | --- | | Dr. V. Venkataramana (Co-ordinator) | | Dr. T. Sampath Kumar | | Dr. Pramoda Patro | | Dr. Brij Kishor Tiwari | | Dr.J.Ravichander | | Dr. Mohammand Ali Shaik | | Dr. Anirodh Kumar | | Mr. S.Naresh Kumar | | Dr. RAJESH VELPULA | | Mr. Kundhan Kumar | | Ms. Ch.Rajitha | | Mr. M Prakash | | Mr. B.Raju | | Intern 1 (Dharma teja) | | Intern 2 (Sai Prasad) | | Intern 3 (Sowmya) | | NS\_2 ( Mounika) | | | | | | |
| **Course Code** | | | 24CS002PC215 | **Course Title** | | AI Assisted Coding | | | |
| **Year/Sem** | | | II/I | **Regulation** | | R24 | | | |
| **Date and Day**  **of Assignment** | | | Week5 - Monday | **Time(s)** | |  | | | |
| **Duration** | | | 2 Hours | **Applicable to**  **Batches** | |  | | | |
| **AssignmentNumber: 9.1**(Present assignment number)/**24**(Total number of assignments) | | | | | | | | | |
|  | **Q.No.** | **Question** | | | | | | ***Expected Time***  ***to complete*** |  |
|  | 1 | **Lab 9 – Documentation Generation: Automatic Documentation and Code Comments**  **Lab Objectives**   * To use AI-assisted coding tools for generating Python documentation and code comments. * To apply zero-shot, few-shot, and context-based prompt engineering for documentation creation. * To practice generating and refining docstrings, inline comments, and module-level documentation. * To compare outputs from different prompting styles for quality analysis.   **Task Description #1** (Documentation – Google-Style Docstrings for Python Functions)   * Task: Use AI to add Google-style docstrings to all functions in a given Python script. * Instructions:   + Prompt AI to generate docstrings without providing any input-output examples.   + Ensure each docstring includes:     - Function description     - Parameters with type hints     - Return values with type hints     - Example usage   + Review the generated docstrings for accuracy and formatting. * Expected Output #1:   + A Python script with all functions documented using correctly formatted Google-style docstrings.   **Task Description #2** (Documentation – Inline Comments for Complex Logic)   * Task: Use AI to add meaningful inline comments to a Python program explaining only complex logic parts. * Instructions:   + Provide a Python script without comments to the AI.   + Instruct AI to skip obvious syntax explanations and focus only on tricky or non-intuitive code sections.   + Verify that comments improve code readability and maintainability. * Expected Output #2:   + Python code with concise, context-aware inline comments for complex logic blocks.   **Task Description #3** (Documentation – Module-Level Documentation)   * Task: Use AI to create a module-level docstring summarizing the purpose, dependencies, and main functions/classes of a Python file. * Instructions:   + Supply the entire Python file to AI.   + Instruct AI to write a single multi-line docstring at the top of the file.   + Ensure the docstring clearly describes functionality and usage without rewriting the entire code. * Expected Output #3:   + A complete, clear, and concise module-level docstring at the beginning of the file.   **Task Description #4** (Documentation – Convert Comments to Structured Docstrings)   * Task: Use AI to transform existing inline comments into structured function docstrings following Google style. * Instructions:   + Provide AI with Python code containing inline comments.   + Ask AI to move relevant details from comments into function docstrings.   + Verify that the new docstrings keep the meaning intact while improving structure. * Expected Output #4:   + Python code with comments replaced by clear, standardized docstrings.   **Task Description #5** (Documentation – Review and Correct Docstrings)   * Task: Use AI to identify and correct inaccuracies in existing docstrings. * Instructions:   + Provide Python code with outdated or incorrect docstrings.   + Instruct AI to rewrite each docstring to match the current code behavior.   + Ensure corrections follow Google-style formatting. * Expected Output #5:   + Python file with updated, accurate, and standardized docstrings.   **Task Description #6** (Documentation – Prompt Comparison Experiment)   * Task: Compare documentation output from a vague prompt and a detailed prompt for the same Python function. * Instructions:   + Create two prompts: one simple (“Add comments to this function”) and one detailed (“Add Google-style docstrings with parameters, return types, and examples”).   + Use AI to process the same Python function with both prompts.   + Analyze and record differences in quality, accuracy, and completeness. * Expected Output #6:   + A comparison table showing the results from both prompts with observations. | | | | | | Week5 - Monday |  |

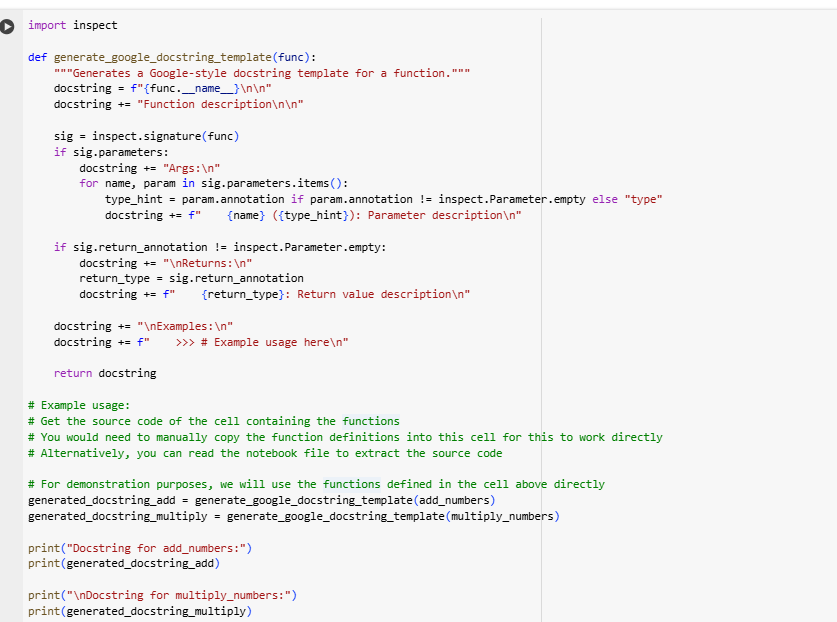
Name: J.SAI GANSEH

ROLL.NO:2403A51321 DATE:25/08/25

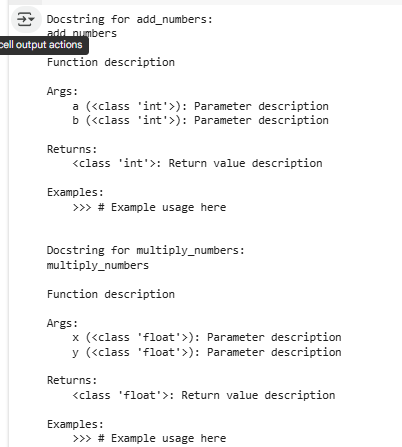
TASK 1:

* PROMPT: add Google-style docstrings to all functions in a given Python script.
  + Ensure each docstring includes:
    - Function description
    - Parameters with type hints
    - Return values with type hints
    - Example usage

CODE:



OUTPUT::



**Observation:**

* The code generates Google-style docstrings using the **inspect** module.
* It collects function name, parameters, and return type to build the template.
* Example usage is shown with add\_numbers and multiply\_numbers.
* This helps in writing clean and consistent documentation easily.

**#Task2**

**PROMPT:** Python script that currently has no comments. Please add inline comments only for the complex or non-obvious logic parts. Do not explain basic syntax like loops, variable assignments, or print statements. Focus on tricky, non-intuitive, or important logic sections. Make sure the comments are meaningful, improve readability, and help with maintainability of the code

**CODE:**

A screenshot of a computer program

AI-generated content may be incorrect.

A white rectangular object with a black border

AI-generated content may be incorrect.

**Observation:**

* The code solves three problems: longest substring, two-sum, and missing number.
* It uses efficient methods like sliding window, hash map, and sum formula.
* The logic improves speed compared to simple brute-force approaches.

**#TASK3**

**PROMPT:**

Create a single multi-line module-level docstring to place at the very top of the file. The docstring should summarize the overall purpose of the script, mention any dependencies used, and briefly describe the main functions or classes it contains. Do not rewrite or change the code itself, just provide the docstring that improves readability and explains usage."

**Code:**



**Output:**

A screenshot of a computer program

AI-generated content may be incorrect.

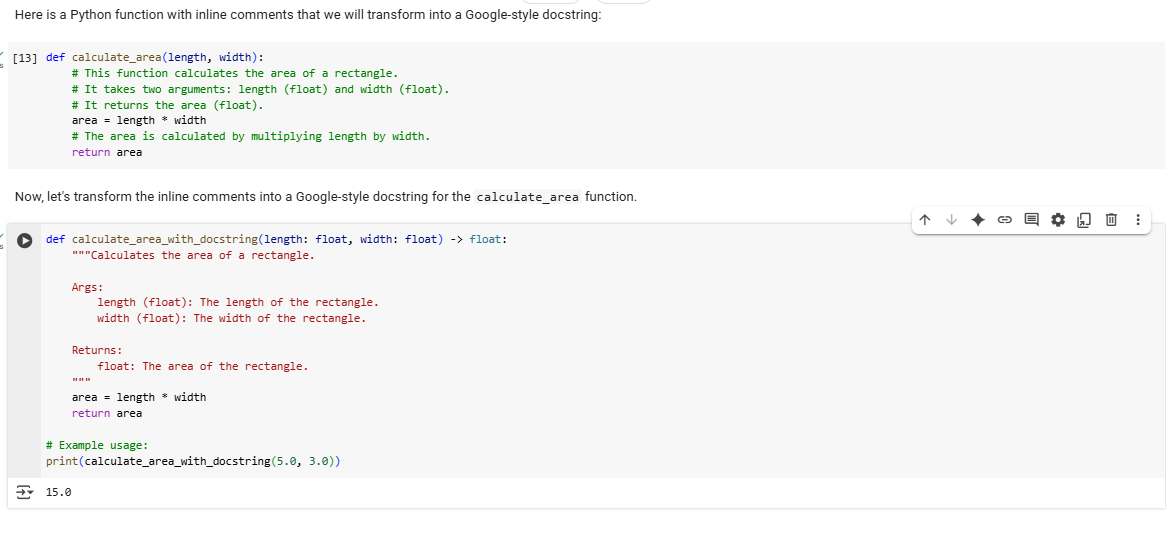
**Observation:**

1. The module provides three efficient algorithmic solutions: longest substring, two-sum, and missing number.
2. It uses optimized techniques like sliding window, hash maps, and arithmetic formulas.
3. The code avoids brute-force methods, improving performance and readability.
4. The module includes example test cases under the main block for quick verification.

**#TASK4**

**PROMPT**: a Python file that contains inline comments inside functions. Please transform those inline comments into structured function-level docstrings using the format. Remove the inline comments once they are included in the docstrings. Ensure the new docstrings keep the same meaning, clearly describe the purpose, arguments, and return values, and improve the overall readability and structure of the code."

**Code:**

****

**Observation:**

1. The task focuses on converting inline comments into structured Google-style docstrings.
2. This improves readability by keeping explanations at the function level instead of scattered inside code.
3. The expected output is a Python file with standardized docstrings replacing inline comments.

**#Task5**

**PROMPT:**

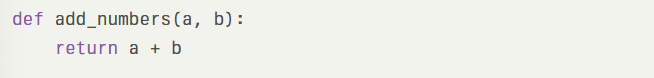
Python code that contains outdated or incorrect docstrings. Please review each function and update its docstring so it accurately matches the current code behavior. Use the Google-style format for all docstrings, ensuring they clearly describe the function’s purpose, arguments, return values, and any important details. Remove or correct any inaccuracies so that the documentation is consistent with the implementation**.**

**Code:A screenshot of a computer

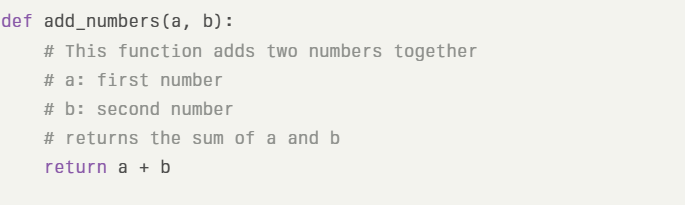
AI-generated content may be incorrect.  
 OUT PUT:  
  
OBSERVATION:**Keeping docstrings consistent with code avoids confusion and ensures accuracy for developers.

**#TASK6**

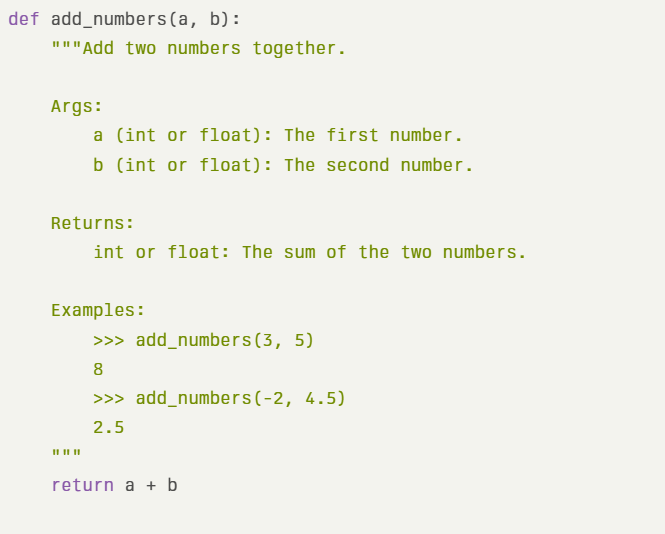
**PROMPT:** Python function. First, generate documentation using a vague prompt: 'Add comments to this function'. Then, generate documentation again using a detailed prompt: 'Add a Google-style docstring with purpose, Args, Returns, and an Examples section'. Finally, compare the two outputs in terms of quality, accuracy, and completeness.  
**CODE: simple function**



**Outputs based on  
. Vague Prompt:**Add comments to this function



**Detailed Prompt:**Add a Google-style docstring with purpose, Args, Returns, and an Examples section



**Observation (Comparision analysis):** A vague prompt produces minimal explanatory comments meant only for human readers, while a detailed Google-style docstring adds *structure, clarity, and reusability*. The second version is far superior for collaborative, maintainable software.