**Name:Chaithra Alluri id:2403A51105 batch:06**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **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.   **Prompt:**  Add Google-style docstrings to all functions in the given Python script .Each docstring should include a brief description of what the function does, all parameters with type hints and descriptions, the return value with type hint and description, and an example usage of the function. Ensure that all docstrings are properly formatted according to the Google Python Style Guide. Output the complete Python script with the docstrings added to every function.  **CODE:**    **OUTPUT:**    **OBSERVATION:**  Google-style docstrings were added to all functions, clearly describing their purpose, parameters with type hints, return values, and example usage. The script is now more readable, maintainable, and consistent, with no changes to the original functionality.  **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.   **PROMPT:**  **CODE:**          **OUTPUT:**    **OBSERVATION:**  Meaningful inline comments were added to only the complex or non-intuitive parts of the Python script. These comments clarify tricky logic, such as Fibonacci term calculation, updating the maximum value, set usage for fast membership checks, preserving order in unique elements, and avoiding dictionary mutation. The script’s readability and maintainability are improved without cluttering straightforward code.  **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.   **PROMPT:**  create a module-level docstring for the following Python file. The docstring should be a single multi-line comment at the top of the file that summarizes the purpose of the module, its dependencies, and the main functions or classes included. Do not rewrite the code or change its logic. Ensure the docstring clearly describes the functionality and how the module can be used. Output the complete Python file with the module-level docstring added at the beginning.  **CODE:**      **OUTPUT:**    **OBSERVATION:**  The module-level docstring summarizes the file’s purpose, main functions, and dependencies, including utilities for math (gcd), lists (flatten, chunk\_list), strings (is\_palindrome), and matrices (transpose\_matrix). It improves readability, aids maintainability, and provides developers a clear overview of the module’s functionality without needing to examine each function individually.  **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.   **PROMPT:**  Convert the inline comments in the Python script into Google-style docstrings for each function. Preserve the meaning while structuring the information clearly. Each docstring should include a description, parameters with type hints, return values with type hints, and example usage if relevant. Output the complete Python code with comments replaced by well-formatted, standardized docstrings without changing the original functionality.  **CODE:**    **OUTPUT:**    **OBSERVATION:**  All inline comments have been successfully converted into structured Google-style docstrings. Each function now clearly explains its purpose, logic, parameters, return values, and provides example usage. This enhances readability, maintainability, and consistency while preserving the original functionality of the code.  **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.   **PROMPT:**  Review the provided Python code and correct any outdated or inaccurate docstrings. Rewrite each docstring to accurately reflect the current function behavior, following Google-style formatting. Ensure they clearly describe the function’s purpose, parameters with type hints, return values, and example usage if relevant. Output the full Python file with all docstrings updated and consistent with the code.  **CODE:**      **OUTPUT:**    **OBSERVATION:**  All docstrings accurately describe the current behavior of the functions, following Google-style formatting. Each docstring clearly explains the function’s purpose, input parameters with type hints, return values, and provides example usage. This ensures the code is readable, maintainable, and consistent with its actual functionality, improving clarity for developers and users.  **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.   **PROMPT:**  Compare documentation generated by AI for the same Python function using two prompts: a vague prompt (“Add comments to this function”) and a detailed prompt (“Add Google-style docstrings with parameters, return types, and examples”). Analyze differences in quality, accuracy, and completeness of the documentation. Provide a comparison table showing results from both prompts along with observations  **TABLE:**   |  |  |  |  | | --- | --- | --- | --- | | **Aspect** | **Vague Prompt: *“Add comments to this function”*** | **Detailed Prompt: *“Add Google-style docstrings with parameters, return types, and examples”*** | **Observations** | | **Format** | Single-line or inline comments (e.g., # Adds two numbers) | Structured Google-style docstring with sections: Description, Args, Returns, Example | Detailed prompt enforces a standard, professional format. | | **Clarity** | Very brief, assumes prior knowledge | Clear explanation of function purpose, parameter meaning, return value, and example usage | Detailed prompt is more beginner-friendly. | | **Accuracy** | Correct but superficial (just says “adds two numbers”) | Correct, with precise details on parameter types and return type | Both are accurate, but detailed prompt adds type specificity. | | **Completeness** | Only describes purpose | Includes description, parameters, return type, and usage example | Detailed prompt ensures comprehensive documentation. | | **Usefulness** | Limited for quick reference | Useful for developers, API users, and tools like Sphinx/ReadTheDocs | Detailed prompt output is professional and ready for integration. |   **OBSERVATION:**  The vague prompt produces minimal, surface-level comments that do not explain parameters, return values, or usage. The detailed prompt yields complete Google-style docstrings, improving clarity, usability, and maintainability. Detailed prompts ensure documentation is accurate, structured, and developer-friendly. | | | | | | Week5 - Monday |  |