|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **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 | | | | | |
| **Instructor(s)Name** | | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | |  | | --- | | Mr. S Naresh Kumar | | Ms. B. Swathi | | Dr. Sasanko Shekhar Gantayat | | Mr. Md Sallauddin | | Dr. Mathivanan | | Mr. Y Srikanth | | Ms. N Shilpa | | Dr. Rishabh Mittal (Coordinator) | | Dr. R. Prashant Kumar | | Mr. Ankushavali MD | | Mr. B Viswanath | | Ms. Sujitha Reddy | | Ms. A. Anitha | | Ms. M.Madhuri | | Ms. Katherashala Swetha | | Ms. Velpula sumalatha | | Mr. Bingi Raju | | Mr. G. Kranthi | | | | | | | |
| **Course Code** | | | 23CS002PC304 | **Course Title** | | AI Assisted Coding | | | |
| **Year/Sem** | | | III/I | **Regulation** | | R23 | | | |
| **Date and Day**  **of Assignment** | | | Week 5 - Thursday | **Time(s)** | | 23CSBTB01 To 23CSBTB52 | | | |
| **Duration** | | | 2 Hours | **Applicable to**  **Batches** | | All Batches | | | |
| **AssignmentNumber:9.4** (Present assignment number)/**24**(Total number of assignments) | | | | | | | | | |
|  | | | | | | | | | |
|  | **Q.No.** | **Question** | | | | | | ***ExpectedTime***  ***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.   **Lab Outcomes**   * Generate structured code documentation using AI tools * Apply appropriate documentation styles to different code contexts * Improve code readability through selective commenting * Convert informal developer comments into professional documentation * Analyze and refine AI-generated documentation | | | | | | Week 5 |  |
|  |  | **Task 1: Auto-Generating Function Documentation in a Shared Codebase**  **Scenario**  You have joined a development team where several utility functions are already implemented, but the code lacks proper documentation. New team members are struggling to understand how these functions should be used.  **Task Description**  You are given a Python script containing multiple functions without any docstrings.  Using an AI-assisted coding tool:   * Ask the AI to automatically generate **Google-style function docstrings** for each function * Each docstring should include:   + A brief description of the function   + Parameters with data types   + Return values   + At least one example usage (if applicable)   Experiment with different prompting styles (zero-shot or context-based) to observe quality differences.  **Expected Outcome**   * A Python script with well-structured Google-style docstrings * Docstrings that clearly explain function behavior and usage * Improved readability and usability of the codebase   **PROMPT**: Generate Google-style docstrings for the following Python functions. Include a brief description, parameters with types, return values, and one example usage.  **EXPLAINATION**: This code adds professional Google-style docstrings to functions. It explains what each function does, its parameters, return values, and gives an example usage. This makes the code easier for new developers to understand and use correctly.    **Task 2: Enhancing Readability Through AI-Generated Inline Comments**  **Scenario**  A Python program contains complex logic that works correctly but is difficult to understand at first glance. Future maintainers may find it hard to debug or extend this code.  **Task Description**  You are provided with a Python script containing:   * Loops * Conditional logic * Algorithms (such as Fibonacci sequence, sorting, or searching)   Use AI assistance to:   * Automatically insert **inline comments only for complex or non-obvious logic** * Avoid commenting on trivial or self-explanatory syntax   The goal is to improve clarity without cluttering the code.  **Expected Outcome**   * A Python script with concise, meaningful inline comments * Comments that explain *why* the logic exists, not *what* Python syntax does * Noticeable improvement in code readability   **PROMPT:** Improve the readability of the following Python script by adding concise inline comments only where the logic is complex or non-obvious. Do not comment trivial Python syntax.  **EXPLAINATION**: This code improves readability by adding comments only where the logic is complex. Instead of explaining basic syntax, it explains why certain decisions are made (like narrowing search space in binary search). This helps future developers debug or modify the code easily.    **Task 3: Generating Module-Level Documentation for a Python Package**  **Scenario:**  Your team is preparing a Python module to be shared internally (or uploaded to a repository). Anyone opening the file should immediately understand its purpose and structure.  **Task Description:**  Provide a complete Python module to an AI tool and instruct it to automatically generate a **module-level docstring** at the top of the file that includes:   * The purpose of the module * Required libraries or dependencies * A brief description of key functions and classes * A short example of how the module can be used   Focus on clarity and professional tone.  **Expected Outcome**   * A well-written multi-line module-level docstring * Clear overview of what the module does and how to use it * Documentation suitable for real-world projects or repositories   **PROMPT:** Generate a professional module-level docstring for the following Python file. Include the purpose of the module, dependencies, key functions/classes, and a short example of usage. Maintain a professional tone suitable for GitHub."  **EXPLAINATION:** The module-level docstring at the top explains the purpose of the entire file. It lists key functions, dependencies, and gives an example of how to use the module. This helps anyone opening the file immediately understand what the project does.    **Task 4: Converting Developer Comments into Structured Docstrings**  **Scenario**  In a legacy project, developers have written long explanatory comments inside functions instead of proper docstrings. The team now wants to standardize documentation.  **Task Description**  You are given a Python script where functions contain detailed inline comments explaining their logic.  Use AI to:   * Automatically convert these comments into structured **Google-style or NumPy-style docstrings** * Preserve the original meaning and intent of the comments * Remove redundant inline comments after conversion   **Expected Outcome**   * Functions with clean, standardized docstrings * Reduced clutter inside function bodies * Improved consistency across the codebase   **PROMPT:** Convert the inline developer comments in the following functions into structured Google-style docstrings. Preserve the meaning and remove redundant inline comments.  **EXPLAINATION:**  This task converts long inline comments into structured Google-style docstrings. It keeps the original meaning but organizes it in a clean, professional format. This reduces clutter inside functions and standardizes documentation.    **Task 5: Building a Mini Automatic Documentation Generator**  **Scenario**  Your team wants a simple internal tool that helps developers start documenting new Python files quickly, without writing documentation from scratch.  **Task Description**  Design a small Python utility that:   * Reads a given .py file * Automatically detects:   + Functions   + Classes * Inserts **placeholder Google-style docstrings** for each detected function or class   AI tools may be used to assist in generating or refining this utility.  Note: The goal is **documentation scaffolding**, not perfect documentation.  **Expected Outcome**   * A working Python script that processes another .py file * Automatically inserted placeholder docstrings * Clear demonstration of how AI can assist in documentation automation   **PROMPT:** Improve the following Python script that automatically inserts placeholder Google-style docstrings for detected functions and classes. Make it cleaner.  **EXPLAINATION:** This script automatically detects functions and classes in a Python file. It inserts placeholder docstrings so developers don’t need to write documentation from scratch. This demonstrates automation and improves productivity in large codebases. | | | | | |  |  |