| **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** | | | Week7 - Monday | **Time(s)** | |  | | | |
| **Duration** | | | 2 Hours | **Applicable to**  **Batches** | |  | | | |
| **AssignmentNumber:13.5**(Present assignment number)/**24**(Total number of assignments) | | | | | | | | | |
|  | | | | | | | | | |
|  | **Q.No.** | **Question** | | | | | | ***Expected Time***  ***to complete*** |  |
|  | 1 | **Lab 13: Code Refactoring – Improving Legacy Code with AI Suggestions**  **Lab Objectives:**   * Identify code smells and inefficiencies in legacy Python scripts. * Use AI-assisted coding tools to **refactor** for readability, maintainability, and performance. * Apply **modern Python best practices** while ensuring output correctness.     **Task 1: Refactoring a Legacy Calculator Script**  **Scenario:** A university has a legacy Python script for a basic calculator that uses long, repetitive if-else statements for each operation. The code is difficult to maintain.   * Upload the calculator script to a GitHub repository. * Use **GitHub Copilot** to suggest a more modular and cleaner version (e.g., functions, dictionary-based mapping). * Compare the AI-suggested refactoring with the original code and document improvements.   Before:    **Prompt: Refactor this calculator script to make it modular using functions and dictionary-based operation mapping. Include proper error handling for division by zero and invalid inputs.**  **Code:**      **Output:**  **Comparison:**   | **Aspect** | **Legacy Code** | **Refactored Code (AI-Suggested)** | | --- | --- | --- | | **Code Structure** | Single large block with if-else | Modular functions (add, subtract, etc.) | | **Maintainability** | Hard to modify or extend | Easy to add new ops via dictionary | | **Error Handling** | Manual and scattered | Centralized and exception-based | | **Readability** | Long repetitive logic | Clean and short mapping-based logic | | **Reusability** | Not reusable | Functions can be imported elsewhere | | **Extensibility** | Adding new ops requires new if-else | Just add new key-function in dictionary | | **AI Suggestion Source** | None | GitHub Copilot suggested dictionary-based approach |     **Task 2: Modernizing a Student Database Program**  **Scenario:** An old student management program uses procedural code with global variables and no error handling. The program frequently crashes when handling incorrect inputs.   * Push the legacy code into your GitHub repo. * Ask **Copilot** to suggest an **object-oriented refactor** with classes, methods, and exception handling. * Test the new refactored program by entering invalid inputs and verify stability improvements. * Original:   Prompt:  Refactored code:    Output:   | **Feature** | **Legacy** | **Refactored** | | --- | --- | --- | | Structure | Procedural | Object-Oriented | | Error Handling | None | Robust with try/except | | Data Management | Global list | Encapsulated in class | | User Experience | Crashes on wrong input | Graceful error messages |   Redme file    **Task 3: Optimizing Performance in File Processing**  **Scenario:** A company’s file-processing script reads large log files line by line using inefficient loops, causing delays.   * Commit the original file-processing script to GitHub. * Use **Copilot** suggestions to replace inefficient loops with more optimized approaches (e.g., list comprehension, built-in functions, generators). * Compare the execution time of legacy vs. refactored versions and document the performance gains. * Original:   Prompt:  Optimize this file processing script using generators, list comprehension, and built-in functions    **Refactored:**    **Readme file**    **Task 4: Enhancing Readability and Documentation**  **Scenario:**  A research group has shared a scientific computation script with minimal comments, inconsistent naming, and poor readability.   * Upload the legacy code to GitHub. * Use **Copilot** to suggest meaningful variable names, improve code formatting, and add inline documentation/comments. * Generate an AI-assisted **README.md** file for the project explaining usage, inputs, and outputs. * Original code:     Prompt:  Refactor this scientific computation code with better variable names, docstrings, comments, and readability  Refactored code:    **Output:**  **Improvements Made**  Renamed variables → a → base\_value, b → number\_of\_terms, s → total\_sum.  Added **function docstrings** and **inline comments**.  Included **error handling**.  User-friendly input and formatted output.  Enhanced readability and structure.  Readme file: | | | | | | Week 7 - Friday |  |