|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **SCHOOL OF COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE** | | | | | **DEPARTMENT OF COMPUTER SCIENCE ENGINEERING** | | | | |
| **ProgramName:**B. Tech | | | | **Assignment Type: Lab** | | | **AcademicYear:**2025-2026 | | |
| **CourseCoordinatorName** | | | | 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) | | | | | | |
| **CourseCode** | | | 24CS002PC215 | **CourseTitle** | | AI Assisted Coding | | | |
| **Year/Sem** | | | II/I | **Regulation** | | R24 | | | |
| **Date and Day**  **of Assignment** | | | Week4 - Thursday | **Time(s)** | |  | | | |
| **Duration** | | | 2 Hours | **Applicableto**  **Batches** | |  | | | |
| **AssignmentNumber:7.4**(Present assignment number)/**24**(Total number of assignments) | | | | | | | | | |
|  | | | | | | | | | |
|  | | | | | | | | | |
|  | **Q.No.** | **Question** | | | | | | ***ExpectedTime***  ***to complete*** |  |
|  | 1 | Lab 7: Error Debugging with AI – Systematic Approaches to Finding and Fixing Bugs  Lab Objectives:   * To identify and correct syntax, logic, and runtime errors in Python programs using AI tools. * To understand common programming bugs and AI-assisted debugging suggestions. * To evaluate how AI explains, detects, and fixes different types of coding errors. * To build confidence in using AI to perform structured debugging practices.   Lab Outcomes (LOs):  After completing this lab, students will be able to:   * Use AI tools to detect and correct syntax, logic, and runtime errors. * Interpret AI-suggested bug fixes and explanations. * Apply systematic debugging strategies supported by AI-generated insights. * Refactor buggy code using responsible and reliable programming patterns.   **Task Description #1:**  • Introduce a buggy Python function that calculates the factorial of a number using recursion. Use Copilot or Cursor AI to detect and fix the logical or syntax errors.  **PROMPT:**    **CODE:**        **Expected Outcome #1:**  **•** Copilot or Cursor AI correctly identifies missing base condition or incorrect recursive call and suggests a functional factorial implementation.  **OUTPUT:**    **CONCLUSION:**  **This script provides excellent, production-quality implementations of both iterative and recursive factorial functions. It adheres to modern Python standards with clear docstrings, type hints, and robust error handling for invalid inputs like negative numbers. The recursive function clearly demonstrates the core concepts of a base case and a recursive step. The main execution block effectively tests both functions to ensure they produce the correct results and handle edge cases properly.**  **Task Description #2:**  **•** Provide a list sorting function that fails due to a type error (e.g., sorting list with mixed integers and strings). Prompt AI to detect the issue and fix the code for consistent sorting**.**  **PROMPT:**    **CODE:**      **Expected Outcome #2:**  **•** AI detects the type inconsistency and either filters or converts list elements, ensuring successful sorting without a crash.  **OUTPUT:**    **CONCLUSION:**  **This script effectively demonstrates two robust strategies for sorting lists with mixed data types, skillfully avoiding Python's Type Error. The first method intelligently converts all elements to strings for a unified lexicographical sort using the key argument. The second method filters the list by data type, creating and sorting separate, homogeneous lists of integers and strings. Overall, it's an excellent showcase of defensive programming and practical data manipulation techniques for handling heterogeneous collections.**  **Task Description #3:**  **•** Write a Python snippet for file handling that opens a file but forgets to close it. Ask Copilot or Cursor AI to improve it using the best practice (e.g., with open() block).  **PROMPT:**    **CODE:**    **Expected Outcome #3:**  **•** AI refactors the code to use a context manager, preventing resource leakage and runtime warnings.  **OUTPUT:**    **CONCLUSION:**  **This script provides a robust, production-quality function for safely writing to files in Python. It correctly utilizes a with statement, ensuring the file is automatically closed to prevent resource leaks and potential data corruption. The function is made more resilient with try...except error handling for I/O operations. Overall, it's a clear and concise demonstration of modern, idiomatic Python for reliable file management.**  **Task Description #4:**  **•** Provide a piece of code with a ZeroDivisionError inside a loop. Ask AI to add error handling using try-except and continue execution safely.  **PROMPT:**    **CODE:**    **Expected Outcome #4:**  **•** Copilot adds a try-except block around the risky operation, preventing crashes and printing a meaningful error message.  **OUTPUT:**    **CONCLUSION:**  **This script provides an excellent demonstration of handling ZeroDivisionError within a loop using a try-except block. By catching the specific exception, it prevents a fatal crash and ensures the program's continuity and stability. The use of the continue statement is key, allowing the loop to safely skip the problematic iteration and proceed. Overall, it's a clear, practical example of defensive programming, essential for creating reliable and resilient applications.**  **Task Description #5:**  **•** Include a buggy class definition with incorrect \_\_init\_\_ parameters or attribute references. Ask AI to analyze and correct the constructor and attribute usage.  **PROMPT:**    **CODE:**      **Expected Outcome #5:**  **•** Copilot identifies mismatched parameters or missing self references and rewrites the class with accurate initialization and usage.  **OUTPUT:**    **CONCLUSION:**  **This script provides an excellent, clear demonstration of a common AttributeError in object-oriented Python. It effectively illustrates how a simple typo in an attribute name within the \_\_init\_\_ method can cause runtime failures in other parts of the class. The code wisely uses a try...except block not just to prevent a crash, but also to explain the exact nature of the bug. Overall, it's a perfect, self-contained lesson on the importance of consistent attribute naming and proper error handling**  **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**  **Evaluation Criteria:**   | **Criteria** | **Max Marks** | | --- | --- | | Logic | 0.5 | | Type mismatch in list elements during sorting | 0.5 | | Resource | 0.5 | | Runtime | 0.5 | | Syntax | 0.5 | | **Total** | **2.5 Marks** | | | | | | | Week4 - Thursday |  |