|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **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** | | | Week4 - Monday | **Time(s)** | |  | | | |
| **Duration** | | | 2 Hours | **Applicable to**  **Batches** | |  | | | |
| **AssignmentNumber:8.1**(Present assignment number)/**24**(Total number of assignments) | | | | | | | | | |
|  | | | | | | | | | |
|  | | | | | | | | | |
|  | **Q.No.** | **Question** | | | | | | ***Expected Time***  ***to complete*** |  |
|  | 1 | Lab 8: Test-Driven Development with AI – Generating and Working with Test Cases  **Lab Objectives:**   * To introduce students to test-driven development (TDD) using AI code generation tools. * To enable the generation of test cases before writing code implementations. * To reinforce the importance of testing, validation, and error handling. * To encourage writing clean and reliable code based on AI-generated test expectations.   **Lab Outcomes (LOs):**  After completing this lab, students will be able to:   * Use AI tools to write test cases for Python functions and classes. * Implement functions based on test cases in a test-first development style. * Use unittest or pytest to validate code correctness. * Analyze the completeness and coverage of AI-generated tests. * Compare AI-generated and manually written test cases for quality and logic   Task Description #1 (Password Strength Validator – Apply AI in Security Context)   * Task: Apply AI to generate at least 3 assert test cases for is\_strong\_password(password) and implement the validator function. * Requirements:   + Password must have at least 8 characters.   + Must include uppercase, lowercase, digit, and special character.   + Must not contain spaces.   Example Assert Test Cases:  assert is\_strong\_password("Abcd@123") == True  assert is\_strong\_password("abcd123") == False  assert is\_strong\_password("ABCD@1234") == True  **PROMPT:**    **CODE:**        Expected Output #1:   * Password validation logic passing all AI-generated test cases.   **OUTPUT:**    **CONCLUSION:**  **This Python script defines a function, is\_strong\_password, to validate password strength based on a set of rules. It checks if a password has at least 8 characters, no spaces, and includes an uppercase letter, a lowercase letter, a digit, and a special character. The function returns True if all conditions are met, and False otherwise. The script also contains a suite of assert statements to test the validator with various valid and invalid passwords.**  Task Description #2 (Number Classification with Loops – Apply AI for Edge Case Handling)   * Task: Use AI to generate at least 3 assert test cases for a classify\_number(n) function. Implement using loops. * Requirements:   + Classify numbers as Positive, Negative, or Zero.   + Handle invalid inputs like strings and None.   + Include boundary conditions (-1, 0, 1).   Example Assert Test Cases:  assert classify\_number(10) == "Positive"  assert classify\_number(-5) == "Negative"  assert classify\_number(0) == "Zero"  **PROMPT:**    **CODE:**      Expected Output #2:   * Classification logic passing all assert tests.   **OUTPUT:**    **CONCLUSION:**  **This Python script provides a robust function, is\_strong\_password, to validate password strength against a clear set of rules. It efficiently checks for minimum length, absence of spaces, and the required mix of uppercase, lowercase, digit, and special characters. The code is clean, well-documented, and includes a comprehensive suite of assert tests to ensure its reliability and correctness. It serves as a practical, production-ready utility for enforcing strong password policies and enhancing application security.**  Task Description #3 (Anagram Checker – Apply AI for String Analysis)   * Task: Use AI to generate at least 3 assert test cases for is\_anagram(str1, str2) and implement the function. * Requirements:   + Ignore case, spaces, and punctuation.   + Handle edge cases (empty strings, identical words).   Example Assert Test Cases:  assert is\_anagram("listen", "silent") == True  assert is\_anagram("hello", "world") == False  assert is\_anagram("Dormitory", "Dirty Room") == True  **PROMPT:**    **CODE:**      Expected Output #3:   * Function correctly identifying anagrams and passing all AI-generated tests.   **OUTPUT**:    **CONCLUSION**:  This Python script provides an elegant and efficient is\_anagram function to determine if two strings are anagrams. It robustly handles variations by normalizing the inputs—ignoring case, spaces, and punctuation—before making a comparison. The core logic, which compares the sorted versions of the cleaned strings, is both simple and powerful. Finally, the script includes a comprehensive suite of assert tests, ensuring its reliability by validating against numerous edge cases and requirements.  Task Description #4 (Inventory Class – Apply AI to Simulate Real-World Inventory System)   * Task: Ask AI to generate at least 3 assert-based tests for an Inventory class with stock management. * Methods:   + add\_item(name, quantity)   + remove\_item(name, quantity)   + get\_stock(name)   Example Assert Test Cases:  inv = Inventory()  inv.add\_item("Pen", 10)  assert inv.get\_stock("Pen") == 10  inv.remove\_item("Pen", 5)  assert inv.get\_stock("Pen") == 5  inv.add\_item("Book", 3)  assert inv.get\_stock("Book") == 3  **PROMPT:**    **CODE:**          Expected Output #4:   * Fully functional class passing all assertions.   **OUTPUT:**    **CONCLUSION:**  **This Python script provides a well-designed Inventory class for managing item stock with clear, encapsulated logic. It features robust methods to add, remove, and retrieve item quantities, using a dictionary for efficient data handling. The class includes essential error checking, such as preventing the removal of more stock than available and rejecting invalid quantities. A comprehensive set of assert statements effectively tests the functionality, ensuring the class is reliable and works as expected under various conditions.**  Task Description #5 (Date Validation & Formatting – Apply AI for Data Validation)   * Task: Use AI to generate at least 3 assert test cases for validate\_and\_format\_date(date\_str) to check and convert dates. * Requirements:   + Validate "MM/DD/YYYY" format.   + Handle invalid dates.   + Convert valid dates to "YYYY-MM-DD".   Example Assert Test Cases:  assert validate\_and\_format\_date("10/15/2023") == "2023-10-15"  assert validate\_and\_format\_date("02/30/2023") == "Invalid Date"  assert validate\_and\_format\_date("01/01/2024") == "2024-01-01"  **PROMPT:**    **CODE:**        Expected Output #5:   * Function passes all AI-generated assertions and handles edge cases.   **OUTPUT:**    **CONCLUSION:**  **This Python script provides a validate\_and\_format\_date function to reliably check and convert date strings from "MM/DD/YYYY" to "YYYY-MM-DD". It expertly uses the datetime module within a try...except block to handle both format errors and logically invalid dates (like "02/30/2023"). The function returns the newly formatted string on success or None on failure, ensuring predictable behavior. A comprehensive suite of assert tests confirms its correctness and robustness across various valid, invalid, and edge-case inputs.**  ✅ Deliverables (For All Tasks)   1. AI-generated prompts for code and test case generation. 2. At least 3 assert test cases for each task. 3. AI-generated initial code and execution screenshots. 4. Analysis of whether code passes all tests. 5. Improved final version with inline comments and explanation. 6. Compiled report (Word/PDF) with prompts, test cases, assertions, code, and output.   Top of Form  **Evaluation Criteria:**   | **Criteria** | **Max Marks** | | --- | --- | | Task #1 | 0.5 | | Task #2 | 0.5 | | Task #3 | 0.5 | | Task #4 | 0.5 | | Task #5 | 0.5 | | **Total** | **2.5 Marks** | | | | | | | Week4 - Monday |  |