**ASSIGNMENT – 1**

**Name:** Azra Farheen  
**Enrollment Number:** 2505B04211

**Course:** M. Tech (Embedded Systems)

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **SCHOOL OF COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE** | | | | | **DEPARTMENT OF COMPUTER SCIENCE ENGINEERING** | | | | |
| **Program Name:** M. Tech/MCA | | | | **Assignment Type: Lab** | | | **AcademicYear:**2025-2026 | | |
| **Course Coordinator Name** | | | | Venkataramana Veeramsetty | | | | | |
| **Course Code** | | | AIPP | **Course Title** | | AI Assisted Problem Solving Using Python | | | |
| **Year/Sem** | | | I/I | **Regulation** | | R24 | | | |
| **Date and Day**  **of Assignment** | | | Week1 - Monday | **Time(s)** | |  | | | |
| **Duration** | | | 2 Hours | **Applicable to**  **Batches** | | M. Tech/MCA | | | |
| **AssignmentNumber:1.3**(Present assignment number)/**24**(Total number of assignments) | | | | | | | | | |
|  | | | | | | | | | |
|  | **Q.No.** | **Question** | | | | | | ***Expected Time***  ***to complete*** |  |
|  | 1 | Lab 1: Environment Setup – GitHub Copilot and VS Code Integration  **Lab Objectives:**   * To install and configure GitHub Copilot in Visual Studio Code. * To explore AI-assisted code generation using GitHub Copilot. * To analyze the accuracy and effectiveness of Copilot's code suggestions. * To understand prompt-based programming using comments and code context   **Lab Outcomes (LOs):**  After completing this lab, students will be able to:   * Set up GitHub Copilot in VS Code successfully. * Use inline comments and context to generate code with Copilot. * Evaluate AI-generated code for correctness and readability. * Compare code suggestions based on different prompts and programming styles.   **Task Description#1**   * Install and configure GitHub Copilot in VS Code. Take screenshots of each step.   **Expected Output#1**   * Install and configure GitHub Copilot in VS Code. Take screenshots of each step.     Fig 1.1 Searched for the “GitHub Copilot” extension in Visual Studio Code and installing it from the Extensions marketplace.    Fig 1.2 GitHub Copilot extension successfully installed and activated in VS Code.    Fig 1.3 Authorizing Visual Studio Code to connect with my GitHub account.  **Task Description#2**   * Use Copilot to generate a is\_prime() Python function**.**   **Expected Output#2**   * Function to check primality with correct logic.     Fig 2.1 GitHub Copilot automatically suggested the is\_prime() function in response to the comment “Function to check if a number is prime”  I accepted Copilot’s suggestion and added test cases to verify the function with different input values (7, 10, 2, and 1). Each test case checks whether the output matches the expected prime result.    Fig 2.2 The final is\_prime() function and test cases written in Visual Studio Code.  **CODE EXPLANATION:**   * Function Definition: The function is\_prime(n) is defined to check whether the number n is a prime number or not. * Docstring: The line """Return True if n is a prime number, otherwise False.""" describes what the function does. * Condition for Numbers ≤ 1: If n is less than or equal to 1, it returns False because numbers ≤ 1 are not prime. * Condition for Numbers ≤ 3: If n is 2 or 3, it returns True since both are prime numbers. * Divisibility Check by 2 and 3: If the number is divisible by 2 or 3, the function returns False because such numbers cannot be prime. * Initialize Variable: A variable i is set to 5 to begin checking from 5 onwards. * While Loop Condition: The loop while i \* i <= n: continues as long as the square of i is less than or equal to n. This helps reduce unnecessary checks beyond the square root of the number. * Check Divisibility by i and i + 2: Inside the loop, the function checks if n is divisible by i or (i + 2). If it is, the number is not prime, and the function returns False. * Increment Step: The variable i is increased by 6 (i += 6) to skip unnecessary even and multiple-of-3 numbers. * Return True: If no divisors are found in the entire loop, the function returns True, meaning the number is prime. * Testing the Function: The function is tested with four inputs (7, 10, 2, 1).   **OUTPUT:**    Fig 2.3 Output of the Copilot-generated is\_prime() function showing correct results.  **Task Description#3**   * Write a comment like # Function to reverse a string and use Copilot to generate the function.   **Expected Output#3**   * Auto-completed reverse function     Fig 3.1 GitHub Copilot automatically suggested the reverse\_string() function after typing the comment “Function to reverse a string”    **CODE:**    Fig 3.2The final reverse\_string() function and test cases written in Visual Studio Code.  **Code Explanation:**   * The function reverse\_string(s) takes a string s as input. * It uses slicing s[::-1] to reverse the order of characters. * The slicing operator starts from the end and moves backward. * The reversed string is then returned by the function. * Tested with “hello”, “Python”, and “Github”. * Output → olleh, nohtyP, buhtiG, confirming correct reversal.   **OUTPUT:**    Fig 3.3 Output of the Copilot-generated reverse\_string() function showing correct results.  **Task Description#4**   * Generate both recursive and iterative versions of a factorial function using comments..   **Expected Output#4**   * Two working factorial implementations     Fig 4.1 GitHub Copilot automatically generated both recursive and iterative factorial functions from the given comment prompt.    Fig 4.2 Final Python code showing both recursive and iterative factorial functions with test cases.  **Code Explanation:**   * This code includes two functions: recursive\_factorial(n) and iterative\_factorial(n). * The recursive version calculates the factorial by calling itself until it reaches the base condition (n == 0 or n == 1). * The iterative version uses a loop to multiply numbers from 1 to n and stores the result in a variable. * Both functions are tested with the input 5, and the expected output is 120. * This demonstrates that both implementations give the same correct result for factorial calculation.   **OUTPUT:**    Fig 4.3 Output of the recursive and iterative factorial functions showing correct results.  **Task Description#5**   * Use Copilot to find the largest number in a list. Assess code quality and efficiency.   **Expected Output#5**   * A valid function with your review     Fig 5.1 GitHub Copilot automatically generated a function to find the largest number in a list.    Fig 5.2 Final Python code generated by GitHub Copilot to find the largest number in a list with multiple test cases.  **Code** **Explanation:**   * The function find\_largest\_number(numbers) finds the largest element in a list. * It checks whether the list is empty and raises an error if so. * Uses the built-in max() function to return the largest value. * Tested with positive, negative, and single-element lists. * Example outputs: 8, 5, −5, and 42. * The code works correctly for all test cases and handles empty lists safely.   **OUTPUT**:    Fig 5.3 Output of the Copilot-generated find\_largest\_number() function showing correct results for all test cases.  **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** | | --- | --- | | Successful Setup of Copilot (Task #1) | 2 | | is\_prime() Python function (Task #2) | 2 | | Reverse a string function (Task #3) | 2 | | Factorial Function (Task #4) | 2 | | Find the largest number (Task #5) | 2 | | **Total** | **10 Marks** | | | | | | | Week1 - Wednesday |  |