**ASSIGNMENT-2**

**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** | | |  | **Course Title** | | AI Assisted Problem Solving Using Python | | | |
| **Year/Sem** | | | I/I | **Regulation** | | R24 | | | |
| **Date and Day**  **of Assignment** | | | Week1 - TUESDAY | **Time(s)** | |  | | | |
| **Duration** | | | 2 Hours | **Applicable to**  **Batches** | | M. Tech/MCA | | | |
| **AssignmentNumber:2.3**(Present assignment number)/**24**(Total number of assignments) | | | | | | | | | |
|  | | | | | | | | | |
|  | **Q.No.** | **Question** | | | | | | ***Expected Time***  ***to complete*** |  |
|  | 1 | Lab 2: Exploring Additional AI Coding Tools – Gemini (Colab) and Cursor AI  **Lab Objectives:**   * To explore and evaluate the functionality of Google Gemini for AI-assisted coding within Google Colab. * To understand and use Cursor AI for code generation, explanation, and refactoring. * To compare outputs and usability between Gemini, GitHub Copilot, and Cursor AI. * To perform code optimization and documentation using AI tools.   **Lab Outcomes (LOs):**  After completing this lab, students will be able to:   * Generate Python code using Google Gemini in Google Colab. * Analyze the effectiveness of code explanations and suggestions by Gemini. * Set up and use Cursor AI for AI-powered coding assistance. * Evaluate and refactor code using Cursor AI features. * Compare AI tool behavior and code quality across different platforms.   **Task Description#1**   * Use Google Gemini in Colab to write a function that reads a CSV file and calculates mean, min, max.   **Expected Output#1**   * Functional code with output and screenshot     Fig 1.1(a)    Fig 1.1(b) Python function created using Google Gemini in Google Colab to read a CSV file and calculate the mean, minimum, and maximum values for each numeric column.  As shown in Fig 1.1 (a) and Fig 1.1 (b), the function successfully reads the CSV file and calculates summary statistics for numeric data.  **Code Explanation:**   1. Importing Libraries:    * pandas is imported as pd to read the CSV file and perform calculations.    * os is imported to handle file paths (optional but good practice). 2. Function Definition (analyze\_csv):    * The function reads a CSV file and calculates the mean, minimum, and maximum for numeric columns.    * A docstring is added to describe the function’s purpose, input, and output clearly. 3. Reading the CSV File:    * The try-except block uses pd.read\_csv() to open the file.    * If the file is missing, it prints an error message and safely exits. 4. Selecting Numeric Columns:    * select\_dtypes(include=['number']) filters only numeric columns, ignoring text or non-numeric data. 5. Handling Empty Numeric Data:    * If no numeric columns are found, it prints a message and stops execution. 6. Calculating Mean, Min, and Max:    * .agg(['mean', 'min', 'max']) computes the required statistics for each numeric column.    * The results are returned as a DataFrame for easy display. 7. Creating a Sample CSV File:    * A small dataset is created with numeric (col1, col2) and text (col3) columns.    * The dataset is saved as sample\_data.csv for testing. 8. Function Execution:    * The created CSV file path is passed to analyze\_csv().    * The function processes the file and stores results in analysis\_output. 9. Displaying the Results:    * If valid output exists, it prints “Analysis Results” and displays the table with mean, min, and max values. 10. Final Output:     * The table shows the calculated mean, minimum, and maximum for all numeric columns.   **OUTPUT:**    Fig 1.2 Output generated from the analyze\_csv() function showing the calculated mean, minimum, and maximum values for each numeric column.  **Task Description#2**   * Compare Gemini and Copilot outputs for a palindrome check function.   **Expected Output#2**   * Side-by-side comparison and observations     Fig 2.1 (a): Python function generated using Google Gemini in Google Colab to check whether a string is a palindrome, ignoring case, spaces, and punctuation.    Fig 2.1 (b): Output of the palindrome-checking function generated using Google Gemini in Colab.  **Code Explanation for Google Gemini (Colab) – Palindrome Function**   1. The code starts by importing the **re module** to use regular expressions. 2. The function is\_palindrome(text) is defined to check whether a string is a palindrome. 3. The text is first converted to **lowercase** using .lower(). 4. Then, **all non-alphanumeric characters** (like spaces, commas, and punctuation) are removed using re.sub(r'[^a-zA-Z0-9]', '', text). 5. The cleaned string is then compared with its **reverse** using slicing ([::-1]). 6. If both strings are the same, it returns **True**; otherwise, **False**. 7. Finally, the function is tested with three examples — "A man, a plan, a canal: Panama", "racecar", and "hello world". 8. The first two return **True**, and the last one returns **False**, proving the function works correctly.     Fig 2.2 (a): Python function generated using GitHub Copilot in Visual Studio Code to check if a string is a palindrome.    Fig 2.2 (b): Output of the palindrome function generated by GitHub Copilot in VS Code.  **Code Explanation for GitHub Copilot (VS Code) – Palindrome Function**   1. The function is\_palindrome(s) is defined to check whether a string is a palindrome. 2. It converts all characters in the input string to **lowercase** using .lower(). 3. Using a **generator expression**, it keeps only **alphanumeric characters** (char for char in s if char.isalnum()). 4. These characters are joined together using "".join() to form a cleaned string. 5. The cleaned string is then compared with its **reverse** ([::-1]). 6. If both are the same, the function returns **True**; otherwise, **False**. 7. The function is tested with the same inputs — "A man, a plan, a canal: Panama", "racecar", and "hello". 8. The first two return **True**, and the last one returns **False**, showing the function is accurate and efficient.   **SIDE BY SIDE COMPARISON :**  **Copilot (Output):**  **Google colab (Output):**     * Both tools gave the correct results for all test cases. * Gemini’s code used the re module and regular expressions (re.sub()) to remove unwanted characters and punctuation. * Copilot’s code used a simpler method with isalnum() and a generator expression to filter characters. * The logic and output were the same, but Copilot’s version was shorter and more readable, while Gemini’s version was more descriptive and used advanced text-cleaning logic.   **Task Description#3**   * Ask Gemini to explain a Python function (to calculate area of various shapes) line by line..   **Expected Output#3**   * Detailed explanation with code snippet       Fig 3.1 Python function implemented in Google Colab using Gemini to calculate areas of circle, rectangle, and triangle.  **CODE EXPLANATION:**   1. **import math** – Imports the math module to use mathematical constants like π (pi) for circle area. 2. **def area(shape, \*\*kwargs):** – Defines a function named area that accepts a shape name and additional parameters depending on the shape. 3. **Docstring ("""...""")** – Describes what the function does and lists which arguments are needed for each shape. 4. **shape = shape.lower()** – Converts the input to lowercase so the function works even if the user types “Circle” or “CIRCLE.” 5. **if shape == 'circle':** – Checks if the shape is a circle. 6. **r = kwargs.get('radius', 0)** – Gets the radius value; defaults to 0 if not provided. 7. **return math.pi \* r \* r** – Calculates the area of the circle using the formula πr². 8. **elif shape == 'rectangle':** – If the shape is a rectangle, this block runs. 9. **l = kwargs.get('length', 0) and w = kwargs.get('width', 0)** – Gets length and width values. 10. **return l \* w** – Returns the rectangle’s area (length × width). 11. **elif shape == 'triangle':** – Checks if the shape is a triangle. 12. **b = kwargs.get('base', 0) and h = kwargs.get('height', 0)** – Gets base and height values. 13. **return 0.5 \* b \* h** – Calculates the triangle’s area using ½ × base × height. 14. **else:** – Runs if the user enters an unsupported shape name. 15. **raise ValueError(...)** – Displays an error message guiding the user to choose a valid shape. 16. **Example calls** –  * area('circle', radius=3) → 28.27 * area('rectangle', length=4, width=5) → 20 * area('triangle', base=6, height=4) → 12   **OUTPUT:**    Fig 3.2 Program output verifying area calculations for the test cases (circle r=3, rectangle 4x5, triangle b=6 h=4)  **Task Description#4**   * Install and configure Cursor AI. Use it to generate a Python function (e.g., sum of squares).   **Expected Output#4**   * Screenshots of working environments with few prompts to generate python code     Fig 2.4(a) Cursor AI installation in VS Code    Fig 2.4(b) Python function and example test cases generated using Cursor AI in VS Code.  **CODE EXPLANATION:**   1. Function Definition   *def sum\_of\_squares(numbers):*   * Defines a function named sum\_of\_squares * It takes one parameter: numbers (a list of integers or floats)  1. Docstring (Explanation Inside Triple Quotes)   *"""*  *Calculate the sum of squares of numbers in a list.*  *...*  *"""*   * Describes what the function does * Mentions:   + Args: input list   + Returns: sum of squares   + Examples: sample inputs and their outputs * Helps users understand how to use the function  1. Return Statement   *return sum(x \*\* 2 for x in numbers)*   * Uses a generator expression * x \*\* 2 → squares each number * sum() → adds all squared values * Example: [1,2,3] → 1² + 2² + 3² = 14  1. Main Block   *if \_\_name\_\_ == "\_\_main\_\_":*   * Ensures this part runs only when the file is executed, not imported * Used for testing code  1. Test Lists   *test\_list1 = [1, 2, 3]*  *test\_list2 = [2, 4, 6]*  *test\_list3 = [5]*  *test\_list4 = []*   * Four sample lists to check if the function works properly * Covers multiple scenarios:   + normal lists   + single element   + empty list  1. Print Statements   *print(f"Sum of squares of {test\_list1}: {sum\_of\_squares(test\_list1)}")*   * Uses f-strings to print message + result * Calls the function for each test list * Example output: Sum of squares of [1, 2, 3]: 14   **OUTPUT:**    Fig 2.4(c) Output generated using Cursor AI showing the sum of squares for different test lists  **Task Description#5**   * Student need to write code to calculate sum of add number and even numbers in the list   **Expected Output#5**   * Refactored code written by student with improved logic   **Prompt:**  Write a Python function to calculate the sum of odd numbers and even numbers in a list.    Fig 2.5(a) Write a Python function to calculate the sum of odd numbers and even numbers in a list.  **CODE EXPLANATION:**  1. Function Definition   * The function calculate\_odd\_even\_sums(numbers) is created to calculate two things:   + sum of odd numbers   + sum of even numbers * It takes one argument: a list of integers.   2. Docstring   * A short description is written inside triple quotes. * It explains what the function does:   + “Returns the sum of odd numbers and even numbers in the given list.”   3. Initialize Variables   * Two variables are declared and set to zero:   + odd\_sum = 0 → stores total of odd numbers   + even\_sum = 0 → stores total of even numbers   4. Loop Through the List   * for num in numbers: * This loop goes through each number in the provided list.   5. Check if Number is Even   * if num % 2 == 0: * % operator checks remainder after division by 2. * If remainder is 0 → number is even → add it to even\_sum using:   + even\_sum += num   6. Otherwise, It Is Odd   * The else: part runs when number is not even. * Odd numbers are added to odd\_sum using:   + odd\_sum += num   7. Return Final Results   * return odd\_sum, even\_sum * The function returns both totals as a pair (tuple):   + first → odd sum   + second → even sum   8. Example Usage Block   * if \_\_name\_\_ == "\_\_main\_\_": * Ensures the test code runs only when the file is executed directly.   9. Sample List   * data = [10, 21, 32, 43, 54, 65] * A test list is created to verify the function.   10. Function Call and Result Storage   * odd\_total, even\_total = calculate\_odd\_even\_sums(data) * Calls the function and stores:   + total odd numbers in odd\_total   + total even numbers in even\_total   11. Print the Results   * The output is displayed using three print statements:   + Original List   + Sum of Odd Numbers   + Sum of Even Numbers   **OUTPUT:**    Fig 2.5(c) Program output showing the calculated sum of odd and even numbers from the given list  **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 Use of Gemini in Colab (Task#1 & #2) | 2.5 | | Code Explanation Accuracy (Gemini) (Task#3) | 2.5 | | Cursor AI Setup and Usage (Task#4) | 2.5 | | Refactoring and Improvement Analysis (Task#5) | 2.5 | | **Total** | **10 Marks** | | | | | | | Week1 - TuesDay |  |