**ASSIGNMENT-4**

**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. and 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** | | | Week3 - Monday | **Time(s)** | |  | | | |
| **Duration** | | | 2 Hours | **Applicable to**  **Batches** | | M.Tech. and MCA | | | |
| **AssignmentNumber:4.3**(Present assignment number)/**24**(Total number of assignments) | | | | | | | | | |
|  | | | | | | | | | |
|  | **Q.No.** | **Question** | | | | | | ***Expected Time***  ***to complete*** |  |
|  | 1 | Lab 4: Advanced Prompt Engineering – Zero-shot, One-shot, and Few-shot Techniques  **Lab Objectives:**   * To explore and apply different levels of prompt examples in AI-assisted code generation. * To understand how zero-shot, one-shot, and few-shot prompting affect AI output quality. * To evaluate the impact of context richness and example quantity on AI performance. * To build awareness of prompt strategy effectiveness for different problem types.   **Lab Outcomes (LOs):**  After completing this lab, students will be able to:   * Use zero-shot prompting to instruct AI with minimal context. * Use one-shot prompting with a single example to guide AI code generation. * Apply few-shot prompting using multiple examples to improve AI responses. * Compare AI outputs across the three prompting strategies.   **Task Description#1**   * Zero-shot: Prompt AI to write a function that checks whether a given year is a leap year.   **Expected Output#1**   * AI-generated function with no examples provided     Fig 4.1(a) Zero-shot AI-generated leap year function in VS Code.    Fig 4.1(b) Testing the zero-shot AI-generated leap year function in VS Code using sample inputs.  **CODE EXPLANATION:**  The function is\_leap\_year(year) checks whether a given year is a leap year based on the rules of the Gregorian calendar.  Explanation of the Logic   1. A year is a leap year if it is divisible by 4. Example: 2020 ÷ 4 = 505 → possible leap year. 2. However, years divisible by 100 are *not* leap years. Example: 1900 ÷ 100 = 19 → NOT a leap year. 3. Exception: If the year is divisible by 400, it *is* a leap year. Example: 2000 ÷ 400 = 5 → leap year.   How the Code Works   * year % 4 == 0 → checks if the year is divisible by 4. * year % 100 != 0 → ensures century years are normally not leap years. * year % 400 == 0 → overrides the previous rule for years divisible by 400.   **OUTPUT:**    Fig 4.1(c) Output  **Task Description#2**   * One-shot: Give one input-output example to guide AI in writing a function that converts centimeters to inches.   **Expected Output#2**   * Function with correct conversion logic   **Prompt (One-shot):**  “Example: 10 cm = 3.937 inches. Using this example, write a Python function to convert centimeters to inches.”    Fig 4.2(a) One-shot AI-generated function for converting centimeters to inches implemented in VS Code.    Fig 4.2(b) Testing the centimeter-to-inch conversion function with user input in VS Code.  **CODE EXPLANATION:**   1. The function cm\_to\_inches(cm) takes a value in centimeters. 2. It uses the standard conversion formula: 1 inch = 2.54 centimeters 3. To convert centimeters to inches, the value is divided by 2.54. 4. The function returns the converted value in inches.   **OUTPUT:**    Fig 4.2(c) Output showing the conversion of 20 centimeters to inches using the one-shot generated function.  **Task Description#3**   * Few-shot: Provide 2–3 examples to generate a function that formats full names as “Last, First”.   **Expected Output#3**   * Well-structured function respecting the examples   **Prompt (Few-shot)**  **Examples provided to the AI:**   * “John Doe → Doe, John” * “Mary Ann Smith → Smith, Mary” * “Ada Lovelace → Lovelace, Ada”   **Full Prompt:**  “Using the above examples, write a Python function that takes a full name as input and returns it in the format ‘Last, First’.”    Fig 4.3(a) Few-shot AI-generated Python function that formats full names into ‘Last, First’ structure.    Fig 4.3(b) Implementation and testing of the name-formatting function using multiple sample inputs in VS Code.  **CODE EXPLANATION:**   1. Splitting the Name: The function uses .split() to break the input name into separate words. 2. Selecting First and Last Names:    * The first word is treated as the *first name*.    * The last word is treated as the *last name*, matching the examples. 3. Middle Names: Middle names are ignored, just like in the provided examples (e.g., “Mary Ann Smith” → “Smith, Mary”). 4. Formatting: The result is returned in the required “Last, First” structure.   **OUTPUT:**    Fig 4.3(c) Terminal output showing correctly formatted names generated by the few-shot name-formatting function.  **Task Description#4**   * Compare zero-shot and few-shot prompts for writing a function that counts the number of vowels in a string.   **Expected Output#4**   * Functional output and comparative reflection   **Prompt (Zero-shot):**  “Write a Python function that counts the number of vowels in a string.”    Fig 4.4(a) Zero-shot Python function for counting vowels using examples.    Fig 4.4(b) Few-shot Python function for counting vowels, created using example-guided prompting.  **CODE EXPLANATION:**  **Zero-shot**   * The model assumed vowels include both uppercase and lowercase. * Directly checks characters in "aeiouAEIOU". * No extra consistency checks since no examples were provided.   **Few-shot**   * Converts the string to lowercase first (more consistent). * Follows the examples strictly (vowels = a, e, i, o, u). * Does not consider “y” as a vowel because the examples did not.   **OUTPUT:**    Fig 4.4 (c) Terminal output displaying vowel count results from the zero-shot generated function.    Fig 4.5(d) Terminal output showing the vowel count results using the few-shot generated function.  **Comparative Reflection**   * Zero-shot produces a correct but generic function. It assumes its own rules (case-insensitive, includes uppercase vowels automatically). * Few-shot produces a more consistent and example-driven function. It converts the string to lowercase and follows the exact vowel definition given in examples. * Few-shot is more reliable in tasks where the desired behavior depends on the user’s examples or formatting rules. * Zero-shot is faster, but may interpret the task differently without guidance.   **Task Description#5**   * Use few-shot prompting to generate a function that reads a .txt file and returns the number of lines.   **Expected Output#5**   * Working file-processing function with AI-guided logic     **Few-shot Prompt**  **Examples given to the AI:**   * Example 1: File: *notes.txt* (contains 10 lines) → Output: **10** * Example 2: File: *empty.txt* (contains 0 lines) → Output: **0**   **Full Prompt:**  “Using the above examples, write a Python function that reads a .txt file and returns how many lines it contains. Include basic error handling.”    Fig 4.5(a) Few-shot AI-generated Python function for counting non-empty lines in a text file.    Fig 4.5(b) The sample text file containing four lines used for testing the line-count function.  **CODE EXPLANATION:**  1. Function Definition   * The function is defined as:   *def count\_lines\_in\_file(path):*   * path is the filename or full file path.   2. Docstring   * Explains:   + What the function does.   + That it returns None if an error occurs.   3. Try Block – Opening the File   * The code tries to open the file safely: * with open(path, "r", encoding="utf-8") as f: * with ensures the file auto-closes after reading.   4. Counting Non-empty Lines   * The main logic:   *return sum(1 for line in f if line.strip() != "")*   * line.strip() removes spaces and newline characters. * If the line is not empty:   + It counts as a valid line. * The sum of valid lines is returned.   5. Handling File Not Found   * If the file is missing:   *except FileNotFoundError:*  *print("File not found:", path)*  *return None*   * This prevents the program from crashing.   6. Handling Other Errors   * Any other unexpected error is caught:   *except Exception as e:*  *print("Error:", e)*  *return None*  7. Example Usage   * To test the function:   *print(count\_lines\_in\_file("sample.txt"))*  **OUTPUT:**    Fig 4.5(c) Terminal output showing the correct line count returned by the file-processing function.  **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** | | --- | --- | | Zero Shot (Task #1) | 2.5 | | One Shot (Task#2) | 2.5 | | Few Shot (Task#3 & Task #5) | 2.5 | | Comparison (Task#4) | 2.5 | | **Total** | **10 Marks** | | | | | | | Week3 - Monday |  |