|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **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** | | | | 1. Dr. Mohammed Ali Shaik  2. Dr. T Sampath Kumar  3. Mr. S Naresh Kumar  4. Dr. V. Rajesh  5. Dr. Brij Kishore  6. Dr Pramoda Patro  7. Dr. Venkataramana  8. Dr. Ravi Chander  9. Dr. Jagjeeth Singh | | | | | |
| **CourseCode** | | | 24CS002PC215 | **CourseTitle** | | AI Assisted Coding | | | |
| **Year/Sem** | | | II/I | **Regulation** | | R24 | | | |
| **Date and Day**  **of Assignment** | | | Week2-Tuesday | **Time(s)** | |  | | | |
| **Duration** | | | 2 Hours | **Applicableto**  **Batches** | | 24CSBTB01 To 24CSBTB39 | | | |
| **AssignmentNumber:3.2**(Present assignment number)/**24**(Total number of assignments) | | | | | | | | | |
|  | | | | | | | | | |
|  | **Q.No.** | **Question** | | | | | | ***ExpectedTime***  ***to complete*** |  |
|  | 1 | Lab 3: Prompt Engineering – Improving Prompts and Context Management  **Lab Objectives:**   * To understand how prompt structure and wording influence AI-generated code. * To explore how context (like comments and function names) helps AI generate relevant output. * To evaluate the quality and accuracy of code based on prompt clarity. * To develop effective prompting strategies for AI-assisted programming.   **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**   * Ask AI to write a function to calculate compound interest, starting with only the function name. Then add a docstring, then input-output example   **Expected Output#1**   * Comparison of AI-generated code styles   **PROMPT**:  Write a Python function to calculate compound interest. Start by giving only the function name. Then, in the next step, add a proper docstring explaining the function. Finally, add an input-output example demonstrating how the function works. Provide at least two variations of the code so I can compare different coding styles."  **OUTPUT:**    **OBSERVATION:**  The first variation is simple and procedural, focusing on readability and ease of understanding for beginners. It avoids type hints and keeps the formula straightforward, making it quick to implement. The second variation, however, is more formal, with type hints, explicit formula documentation, and a professional style suited for production or collaborative projects.    **Task Description#2**   * Do math stuff, then refine it to: # Write a function to calculate average, median, and mode of a list of numbers.   **Expected Output#2**   * AI-generated function evolves from unclear to accurate multi-statistical operation.   **PROMPT:**  Start by writing a simple Python function that 'does math stuff' (keep it vague at first). Then refine and expand the function step by step until it becomes: # Write a function to calculate average, median, and mode of a list of numbers. Show how the function evolves from unclear to an accurate multi-statistical operation."  **OUTPUT:**    **OBSERVATION:**  When given the vague instruction to “do math stuff,” the AI typically starts with a simple or ambiguous function that lacks clear purpose. As the prompt is refined, the function gradually evolves—first introducing basic statistical operations like calculating the average, then extending to include the median, and finally adding the mode. This step-by-step refinement demonstrates how vague tasks can be clarified through iterative prompting, ultimately resulting in a precise and accurate function that computes multiple statistical measures in a structured way.  **Task Description#3**   * Provide multiple examples of input-output to the AI for convert\_to\_binary(num) function. Observe how AI uses few-shot prompting to generalize.   **Expected Output#3**   * Enhanced AI output with clearer prompts   **PROMPT**:  I want to write a Python function convert\_to\_binary(num) that converts an integer to its binary representation as a string. Here are some input-output examples you should learn from  **OUTPUT:**      **OBSERVATION:**  When given a few explicit input-output examples, the AI quickly recognizes the pattern that the function must convert integers into binary strings. This few-shot prompting approach helps it generalize beyond the provided cases, ensuring the function works for any positive integer, not just the examples shown. As a result, the AI produces clearer, more accurate code and often offers multiple implementation styles, such as using Python’s built-in bin() function or performing manual division by two.  **Task Description#4**   * Create an user interface for an hotel to generate bill based on customer requirements   **Expected Output#4**   * Consistent functions with shared logic   **PROMPT**:  The AI uses the prompt to build a hotel billing system with clear, reusable functions. It applies shared logic across variations, ensuring consistent bill generation whether procedural or class-based.  **OUTPUT:**        **OBSERVATION:**  The AI structures the hotel billing system with reusable functions, promoting clarity and reducing redundancy. Both procedural and class-based variations share consistent logic for generating bills. This approach ensures flexibility, maintainability, and uniform results across coding styles.  **Task Description#5**   * Analyzing Prompt Specificity: Improving Temperature Conversion Function with Clear Instructions   **Expected Output#5**   * Code quality difference analysis for various prompts   **Prompt**:  Write a Python function to convert temperatures between Celsius, Fahrenheit, and Kelvin. Start with a minimal version that only handles Celsius to Fahrenheit. Then, progressively expand the function to handle all conversions, add type hints, and include meaningful docstrings. Finally, compare how different levels of prompt detail affect the readability, reusability, and clarity of the generated code."  **OUTPUT**:    **OBSERVATION:**  The user input of 100 with a unit of 'C' resulted in the correct conversion to Fahrenheit, displaying that 100.0°C is equal to 212.00°F, which is the boiling point of water in Fahrenheit.  **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** | | --- | --- | | Task#1 | 0.5 | | Task#2 | 0.5 | | Task #3 | 0.5 | | Task #4 | 0.5 | | Task #5 | 0.5 | | **Total** | **2.5 Marks** | | | | | | | 03.08.2025 EOD |  |