|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **SCHOOLOFCOMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE** | | | | | **DEPARTMENTOFCOMPUTER SCIENCE ENGINEERING** | | | | |
| **ProgramName:**B. Tech | | | | **AssignmentType: Lab** | | | **AcademicYear:**2025-2026 | | |
| **CourseCoordinatorName** | | | | 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) | | | | | | |
| **CourseCode** | | | 24CS002PC215 | **CourseTitle** | | AI Assisted Coding | | | |
| **Year/Sem** | | | II/I | **Regulation** | | R24 | | | |
| **DateandDay**  **of Assignment** | | | Week7 - WednesDay | **Time(s)** | |  | | | |
| **Duration** | | | 2 Hours | **Applicableto**  **Batches** | |  | | | |
| **AssignmentNumber:13.3**(Presentassignmentnumber)/**24**(Totalnumberofassignments) | | | | | | | | | |
| **Name: GUNDU MEGHANA**  **Enrollment No: 2403A510C1** | | | | | | | | | |
|  | **Q.No.** | **Question** | | | | | | ***ExpectedTime***  ***to complete*** |  |
|  | 1 | **Lab 13 – Code Refactoring: Improving Legacy Code with AI Suggestions**  **Lab Objectives**   * To introduce the concept of code refactoring and why it matters (readability, maintainability, performance). * To practice using AI tools for identifying and suggesting improvements in legacy code. * To evaluate the before vs. after versions for clarity, performance, and correctness. * To reinforce responsible AI-assisted coding practices (avoiding over-reliance, validating outputs).   **Learning Outcomes**  After completing this lab, students will be able to:   1. Use AI to analyze and refactor poorly written Python code. 2. Improve code **readability, efficiency, and error handling**. 3. Document AI-suggested improvements through comments and explanations. 4. Apply refactoring strategies without changing functionality. 5. Critically reflect on AI’s refactoring suggestions.   **Task Description #1 – Remove Repetition**  Task: Provide AI with the following redundant code and ask it to refactor  **Python Code**  def calculate\_area(shape, x, y=0):  if shape == "rectangle":  return x \* y  elif shape == "square":  return x \* x  elif shape == "circle":  return 3.14 \* x \* x  **Expected Output**   * Refactored version with dictionary-based dispatch or separate functions. * Cleaner and modular design.   **Prompt:** Refactor the redundant code.  **Code:**    **Observation:** The original code contained repetitive if-elif statements for each shape, making it less efficient and harder to maintain. After refactoring, a dictionary-based approach was used to map shapes to their respective area functions, eliminating redundancy. This made the code cleaner, modular, and easier to extend for future shapes.  **Task Description #2 – Error Handling in Legacy Code**  Task: Legacy function without proper error handling  **Python Code**  def read\_file(filename):  f = open(filename, "r")  data = f.read()  f.close()  return data  **Expected Output:**  AI refactors with with open() and try-except:  **Prompt:** Refactor the given code with open() and try-except.  **Code:**    **Observation:** The refactored function improves error handling by using a try-except block and the context manager (with open() as f). This ensures the file is properly closed even if an error occurs, and allows for specific handling of FileNotFoundError and other exceptions. Returning None in case of errors provides a clear indication of failure, while printing error messages helps with debugging and monitoring.  **Task Description #3 – Complex Refactoring**  Task: Provide this legacy class to AI for readability and modularity improvements:  **Python Code**  class Student:  def \_\_init\_\_(self, n, a, m1, m2, m3):  self.n = n  self.a = a  self.m1 = m1  self.m2 = m2  self.m3 = m3  def details(self):  print("Name:", self.n, "Age:", self.a)  def total(self):  return self.m1+self.m2+self.m3  **Expected Output:**   * AI improves naming (name, age, marks). * Adds docstrings. * Improves print readability. * Possibly uses sum(self.marks) if marks stored in a list.   **Prompt:** Refactor the code by improving naming, add docstrings, improve readability and modularity, use sum(self.marks) function.  **Code:**    **Observation:** The Student class defines methods for storing and displaying a student's details and marks, as well as calculating the total marks. The \_\_init\_\_ method initializes the student's name, age, and three marks. The details() method prints the name and age, and the total() method correctly returns the sum of the three marks. The structure is good for basic encapsulation and operation on student data, but does not currently include error handling or validation for the input values.  **Task Description #4 – Inefficient Loop Refactoring**  Task: Refactor this inefficient loop with AI help  **Python Code**  nums = [1,2,3,4,5,6,7,8,9,10]  squares = []  for i in nums:  squares.append(i \* i)  **Expected Output:** AI suggested a **list comprehension**  **Prompt: Refactor this loop with proper list comprehension.**  **Code:**    **Observation: The initial loop uses a standard for-loop with append() to build the list of squares, which is concise but not the most efficient or Pythonic approach. Refactoring to a list comprehension (squares = [i \* i for i in nums]) makes the code more readable and compact, directly constructing the list in a single line. This improves code efficiency and style according to Python best practices.** | | | | | | Week5 - Monday |  |