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)			

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).	Week5 - Monday

- 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: Write a Python program that asks the user to enter a shape (rectangle, square, or circle) and its dimensions.
The program should calculate and display the area.
Use a clean, refactored design — avoid repetitive if statements by using either a function dictionary or separate functions for each shape.
The program must take user input for shape and dimensions instead of using hardcoded values.

```python
import math
def calculate_rectangle_area(length, width):
  """Calculates the area of a rectangle."""
  return length * width
def calculate_square_area(side):
  """Calculates the area of a square."""
  return side * side
def calculate_circle_area(radius):
  """Calculates the area of a circle."""
  return math.pi * radius**2
area_functions = {
    "rectangle": calculate_rectangle_area,
    "square": calculate_square_area,
    "circle": calculate_circle_area,
}
try:
    shape_name = input("Enter the shape (rectangle, square, or circle): ").lower()
    if shape_name in area_functions:
        area_func = area_functions[shape_name]
        if shape_name == "rectangle":
            length = float(input("Enter the length of the rectangle: "))
            width = float(input("Enter the width of the rectangle: "))
            area = area_func(length, width)
        elif shape_name == "square":
            side = float(input("Enter the side length of the square: "))
            area = area_func(side)
        elif shape_name == "circle":
            radius = float(input("Enter the radius of the circle: "))
            area = area_func(radius)
        print(f"The area of the {shape_name} is: {area}")
    else:
        print("Invalid shape entered.")
except ValueError:
    print("Invalid input for dimensions. Please enter numeric values.")
```

```
Enter the shape (rectangle, square, or circle): rectangle
Enter the length of the rectangle: 7
Enter the width of the rectangle: 5
The area of the rectangle is: 35.0
```

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: Write a Python program that asks the user to enter a filename. The program should try to open and read the file content safely using with open().Add proper try-except blocks to handle possible errors like file not found or permission denied.Display an appropriate message for

each case.Make sure all inputs (like filename) are taken from the user — not hardcoded.

```
try:
    filename = input("Enter the name of the file you want to read: ")
    with open(filename, 'r') as file:
        file_content = file.read()
        print("File content:")
        print(file_content)
except FileNotFoundError:
    print(f"Error: The file '{filename}' was not found.")
except PermissionError:
    print(f"Error: Permission denied to read the file '{filename}'.")
except Exception as e:
    print(f"An unexpected error occurred: {e}")
```

```
Enter the name of the file you want to read: bhanu
Error: The file 'bhanu' was not found.
```

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: Create a Python class named Student that stores a student's name, age, and marks for three subjects.
Ask the user to enter all these details as input.
Add methods to display student details,calculate total marks, and calculate average marks.Write the code using meaningful variable names, proper indentation, and docstrings for every method.

Avoid hardcoded values — take all data from user input.

```
class Student:
    """Represents a student with name, age, and marks in three subjects."""

    def __init__(self, name, age, mark1, mark2, mark3):
        """
        Initializes a new Student instance.

        Args:
            name (str): The name of the student.
            age (int): The age of the student.
            mark1 (float): Marks for subject 1.
            mark2 (float): Marks for subject 2.
            mark3 (float): Marks for subject 3.
        """
        self.name = name
        self.age = age
        self.marks = [mark1, mark2, mark3]

    def display_details(self):
        """Displays the student's name, age, and marks."""
        print(f"Student Name: {self.name}")
        print(f"Student Age: {self.age}")
        print(f"Student Marks: {self.marks}")

    def calculate_total_marks(self):
        """Calculates and returns the total marks of the student."""
        return sum(self.marks)

    def calculate_average_marks(self):
        """Calculates and returns the average marks of the student."""
        return sum(self.marks) / len(self.marks)
```

```
# Get user input for student details
try:
    student_name = input("Enter student's name: ")
    student_age = int(input("Enter student's age: "))
    subject1_mark = float(input("Enter marks for subject 1: "))
    subject2_mark = float(input("Enter marks for subject 2: "))
    subject3_mark = float(input("Enter marks for subject 3: "))

    # Create a Student object
    student_object = Student(student_name, student_age, subject1_mark, subject2_mark, subject3_mark)

    # Call methods and display results
    student_object.display_details()
    total_marks = student_object.calculate_total_marks()
    print(f"\nTotal Marks: {total_marks}")
    average_marks = student_object.calculate_average_marks()
    print(f"Average Marks: {average_marks}")

except ValueError:
    print("Invalid input. Please ensure age is a whole number and marks are numeric values.")
```

```
Enter student's name: Bhanu
Enter student's age: 19
Enter marks for subject 1: 99
Enter marks for subject 2: 98
Enter marks for subject 3: 99
Student Name: Bhanu
Student Age: 19
Student Marks: [99.0, 98.0, 99.0]

Total Marks: 296.0
Average Marks: 98.66666666666667
```

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: Write a Python program that asks the user to enter a list of numbers (space-separated).
Then, use a list comprehension to generate a new list containing the squares of those numbers and display it.
The code should take numbers from user input (not hardcoded in the list).

```
try:
    number_string = input("Enter a list of numbers separated by spaces: ")
    numbers_list = [float(num) for num in number_string.split()]
    squared_numbers = [num ** 2 for num in numbers_list]
    print(squared_numbers)
except ValueError:
    print("Invalid input. Please enter numeric values separated by spaces.")
```

```
Enter a list of numbers separated by spaces: 1 2 3 4 5 6
[1.0, 4.0, 9.0, 16.0, 25.0, 36.0]


Enter a list of numbers separated by spaces: 1 2 3 4,6
Invalid input. Please enter numeric values separated by spaces.
```