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

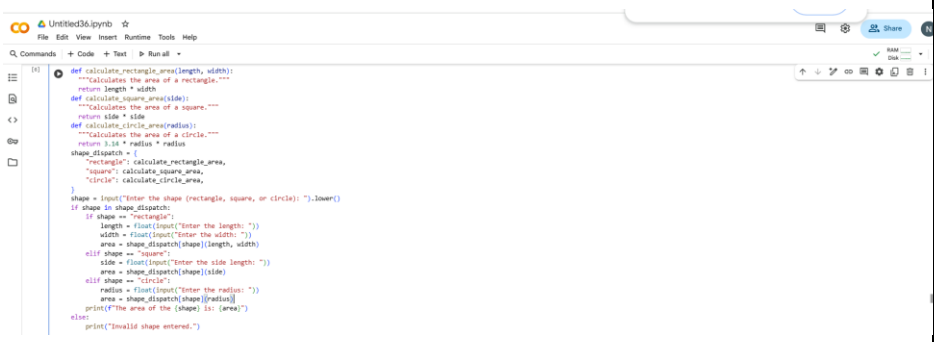
Python Code

```
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
```

PROMPT-1: write a python code ,refactor the code to remove repetition and make it cleaner and modular. separate functions for each shape.

PROMPT-2:Write inputs from the user.

CODE:



OUPUT:

```
Enter the shape (rectangle, square, or circle): circle
Enter the radius: 2.3
The area of the circle is: 16.610599999999998
```

OBSERVATION:

What I Observed is In the orginal code repeats logic inside one function,difficult to extend,add a new function by elif blocks and modularity is one . But In the Refactored Code Version logic can slip intp reusable functions,easy to read and modify,can add a new function and each shape has its own function.

Expected Output

- Refactored version with dictionary-based dispatch or separate functions.
- Cleaner and modular design.

Task Description #2 – Error Handling in Legacy Code

Task: Legacy function without proper error handling

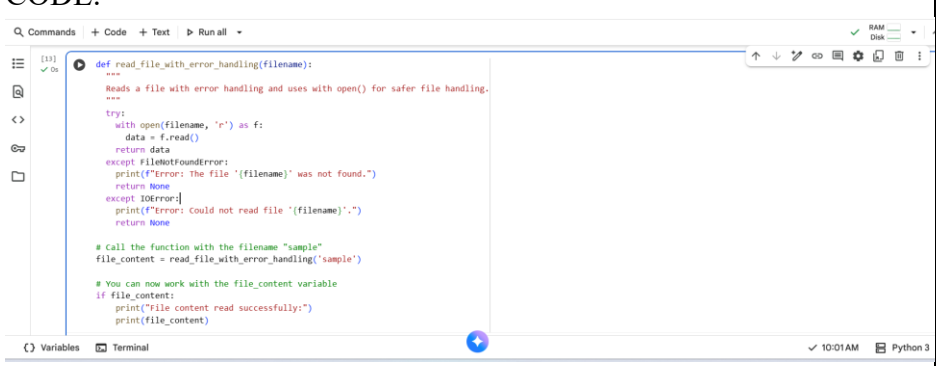
Python Code

```
def read_file(filename):
    f = open(filename, "r")
    data = f.read()
    f.close()
    return data
```

PROMPT-1: Write a python code,refactor the following legacy Python function to include proper error handling.

Prompt-2:Use with open() for safer file handling and wrap the code in a try-except block to handle errors.

CODE:



OUTPUT:

```
Error: The file 'sample' was not found.
```

OBSERVATION:
From the above legacy code the features are manual to open or close and error handling occurs shows None,basic readability and risk to file of not closing and occurs the error. Automatic with open() and handles missing files and input /output erros. Clear and professional way readability .

Expected Output:
AI refactors with with open() and try-except:

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.

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**	