

SCHOOL OF COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE		DEPARTMENT OF COMPUTER SCIENCE ENGINEERING	
Program Name: B. Tech		Assignment Type: Lab	Academic Year: 2025-2026
Course Coordinator Name		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)	
Course Code	24CS002PC215	Course Title	AI Assisted Coding
Year/Sem	II/I	Regulation	R24
Date and Day of Assignment	Week 7 - Wednesday	Time(s)	
Duration	2 Hours	Applicable to Batches	
Assignment Number: 13.3 (Present assignment number)/24 (Total number of assignments)			
Q.No.	Question		Expected Time 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).		Week 5 - 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

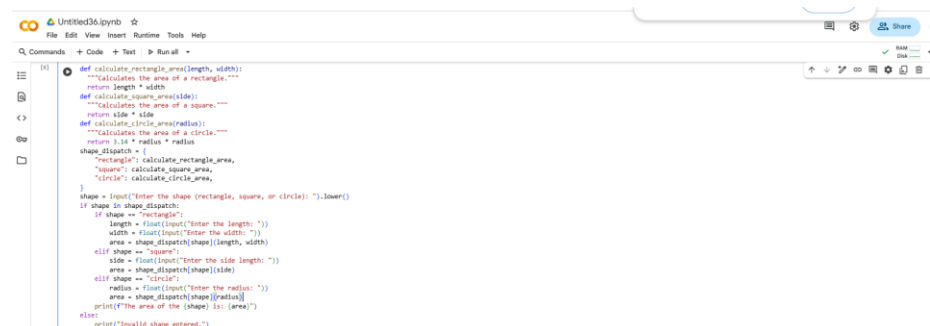
Python Code

```
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:



```
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 3.14 * radius * radius

shape_dispatch = {
    "rectangle": calculate_rectangle_area,
    "square": calculate_square_area,
    "circle": calculate_circle_area,
}

shape = input("Enter the shape (rectangle, square, or circle): ").lower()
if shape in shape_dispatch:
    if shape == "rectangle":
        length = float(input("Enter the length: "))
        width = float(input("Enter the width: "))
        area = shape_dispatch[shape](length, width)
    elif shape == "square":
        side = float(input("Enter the side length: "))
        area = shape_dispatch[shape](side)
    elif shape == "circle":
        radius = float(input("Enter the radius: "))
        area = shape_dispatch[shape](radius)
    print(f"The area of the {shape} is: {area}")
else:
    print("Invalid shape entered.")
```

OUTPUT:

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 original 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 into 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:



```
def read_file_with_error_handling(filename):  
    """  
    Reads a file with error handling and uses with open() for safer file handling.  
    """  
    try:  
        with open(filename, 'r') as f:  
            data = f.read()  
            return data  
    except FileNotFoundError:  
        print(f"Error: The file '{filename}' was not found.")  
        return None  
    except IOError:  
        print(f"Error: could not read file '{filename}'.")  
        return None  
  
# Call the function with the filename "sample"  
file_content = read_file_with_error_handling("sample")  
  
# You can now work with the file_content variable  
if file_content:  
    print("File content read successfully:")  
    print(file_content)
```

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 errors. 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

	<p>Python Code</p> <pre>nums = [1,2,3,4,5,6,7,8,9,10] squares = [] for i in nums: squares.append(i * i)</pre> <p>Expected Output: AI suggested a list comprehension</p>	
--	--	--