

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
```

#### Expected Output

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

#### Prompt#1:

Refactor this Python code to make it cleaner and modular. Use functions or a dictionary to avoid repetitive if-elif statements.

#### Code#1:

```

def rectangle_area(x, y):
    return x * y

def square_area(x):
    return x * x

def circle_area(x):
    return 3.14 * x * x

area_functions = {
    "rectangle": rectangle_area,
    "square": square_area,
    "circle": circle_area
}

def calculate_area(shape, x, y=None):
    if shape in area_functions:
        if shape == "rectangle":
            return area_functions[shape](x, y)
        else:
            return area_functions[shape](x)
    else:
        return "Invalid shape"

shape = input("Enter the shape (rectangle, square, or circle): ")
if shape.lower() == "rectangle":
    x = float(input("Enter the length: "))
    y = float(input("Enter the width: "))
    result = calculate_area(shape.lower(), x, y)
    print(f"The area of the rectangle is: {result}")
elif shape.lower() == "square":
    x = float(input("Enter the side length: "))
    result = calculate_area(shape.lower(), x)
    print(f"The area of the square is: {result}")
elif shape.lower() == "circle":
    x = float(input("Enter the radius: "))
    result = calculate_area(shape.lower(), x)
    print(f"The area of the circle is: {result}")
else:
    result = calculate_area(shape.lower(), None) # Call with None to get "Invalid shape"
    print(result)

```

Enter the shape (rectangle, square, or circle): circle  
 Enter the radius: 23  
 The area of the circle is: 1661.06

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#2:

Refactor this legacy function to safely read a file using with open() and

add try-except error handling.

Code#2:

```
def read_file(filename):  
    try:  
        f = open(filename, "r")  
        data = f.read()  
        f.close()  
        return data  
    except FileNotFoundError:  
        return f"Error: The file '{filename}' was not found."  
    except IOError:  
        return f"Error: Could not read the file '{filename}'."  
filename = input("Enter the filename to read: ")  
file_content = read_file(filename)  
print(file_content)
```

```
Enter the filename to read: /content/New Text Document.txt  
yes ram is aggi
```

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#3: Refactor the given Student class to improve readability and modularity by using clear variable names (name, age, marks), adding docstrings, improving print statements, and using sum(self.marks).

Code#3:

```
class Student:
    """Represents a student with name, age, and three marks."""
    def __init__(self, n: str, a: int, m1: int, m2: int, m3: int):
        """Initializes a Student object.

        Args:
            n: The name of the student.
            a: The age of the student.
            m1: The first mark.
            m2: The second mark.
            m3: The third mark.
        """
        self.n = n # Student's name
        self.a = a # Student's age
        self.m1 = m1 # Mark 1
        self.m2 = m2 # Mark 2
        self.m3 = m3 # Mark 3

    def details(self) -> None:
        """Prints the name and age of the student."""
        print("Name:", self.n, "Age:", self.a)

    def total(self) -> int:
        """Calculates the total marks of the student."""
        return self.m1 + self.m2 + self.m3

# Get user input for student details
name = input("Enter student name: ")
age = int(input("Enter student age: "))
mark1 = int(input("Enter mark 1: "))
mark2 = int(input("Enter mark 2: "))
mark3 = int(input("Enter mark 3: "))

# Create a Student object
student1 = Student(name, age, mark1, mark2, mark3)

# Display student details and total marks
student1.details()
print("Total Marks:", student1.total())
```

```
Enter student name: ram charan
Enter student age: 23
Enter mark 1: 8
Enter mark 2: 23
Enter mark 3: 30
Name: ram charan Age: 23
Total Marks: 61
```

Task Description #4 – Inefficient Loop Refactoring

Task: Refactor thisinefficientloopwithAIhelp

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 list comprehension

Prompt#4:

Generate a the code and refactor the given loop to use a list comprehension for better readability and efficiency.

Code#4:

```
nums_str = input("Enter a list of numbers separated by commas: ")
nums = [int(x) for x in nums_str.split(',')]
squares = [i * i for i in nums]
print(squares)
```

```
Enter a list of numbers separated by commas: 1,2,3
[1, 4, 9]
```