

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)			
Question			Expected Time to complete
1	Lab 13 – Code Refactoring: Improving Legacy Code with AI Suggestions Lab Objectives		Week 5 - Monday

- 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 following redundant 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
```

CODE:

```
def calculate_area(shape, x, y=0):
    area_calculators = {
        "rectangle": lambda x, y: x * y,
        "square": lambda x: x * x,
        "circle": lambda x: 3.14 * x * x
    }
    if shape in area_calculators:
        if shape == "rectangle":
            return area_calculators[shape](x, y)
        else:
            return area_calculators[shape](x)
    else:
        return "Invalid shape"
```

```
# Example usage
```

```
print(calculate_area("rectangle", 5, 10))
```

```
print(calculate_area("square", 7))
```

```
print(calculate_area("circle", 3))
```

```
print(calculate_area("triangle", 4, 6))
```

```
50
49
28.259999999999998
Invalid shape
```

OBSERVATION:

This code defines a function `calculate_area` that computes the area of rectangles, squares, and circles using a dictionary to map shape names to their respective area calculation formulas. The output shows the calculated areas for the example shapes and indicates "Invalid shape" for a triangle, as it's not supported.

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  
the
```

Expected Output:

AI refactors with `with open()` and `try-except`:

PROMPT:

Generate Legacy function without proper error handling for given python code: `def read_file(filename):` `f = open(filename, "r")` `data = f.read()` `f.close()` `return data`
the

OUTPUT:

```
def read_file_legacy(filename):  
    """  
    A legacy function to read a file without proper error handling.  
    This function will raise an error if the file does not exist,  
    if there are permission issues, or if the file cannot be closed.  
    """  
    f = open(filename, "r")  
    data = f.read()  
    f.close()  
    return data  
  
# Example usage (will likely raise an error if the file doesn't exist)  
# try:  
#     content = read_file_legacy("non_existent_file.txt")  
#     print(content)  
# except FileNotFoundError:  
#     print("File not found, as expected with legacy code.")  
# except Exception as e:  
#     print(f"An unexpected error occurred: {e}")
```

OBSERVATION:

My observation is that the provided `read_file_legacy` function lacks error handling, making it vulnerable to crashes if the specified file doesn't exist or has permission issues.

THE ERROR IS:

Based on the code for `read_file_legacy` and its lack of error handling, the most likely error if the specified file does not exist is a `FileNotFoundError`.

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:

Generate this legacy class 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
```

OUTPUT:

```

class Student:
    """Represents a student with their details and marks."""

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

        Args:
            name (str): The name of the student.
            age (int): The age of the student.
            mark1 (int): The mark in the first subject.
            mark2 (int): The mark in the second subject.
            mark3 (int): The mark in the third subject.
        """
        self.name = name
        self.age = age
        self.marks = [mark1, mark2, mark3]

    def details(self):
        """Prints the student's name and age."""
        print(f"Student Details:\n Name: {self.name}\n Age: {self.age}")

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

# Example usage (will be added in a new cell):
# student1 = Student("Alice", 16, 85, 90, 78)
# student1.details()
# print(f" Total marks: {student1.total()}")

# student2 = Student("Bob", 17, 92, 88, 95)
# student2.details()
# print(f" Total marks: {student2.total()}")

```

OBSERVATION:

This code defines a `Student` class to manage student details and marks, including methods to display details and calculate the total marks.

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:

```
nums = [1,2,3,4,5,6,7,8,9,10]
```

```
squares = [] for i in nums:
```

```
    squares.append(i * i)
```

Refactor this code with inefficient loop with help of list comprehension.

OUTPUT:

◆ Gemini

```
# Original loop
nums = [1,2,3,4,5,6,7,8,9,10]
squares_loop = []
for i in nums:
    squares_loop.append(i * i)
print("Output from original loop:", squares_loop)

# Refactored with list comprehension
squares_comprehension = [i * i for i in nums]
print("Output from list comprehension:", squares_comprehension)
```

↔ Output from original loop: [1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
Output from list comprehension: [1, 4, 9, 16, 25, 36, 49, 64, 81, 100]

OBSERVATION:

The list comprehension `[i * i for i in nums]` effectively replaces the explicit loop for creating the squares list. Both the original loop and the list comprehension produce the same output, demonstrating that list comprehension is a concise and efficient way to achieve the same result as the loop in this case.