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
```

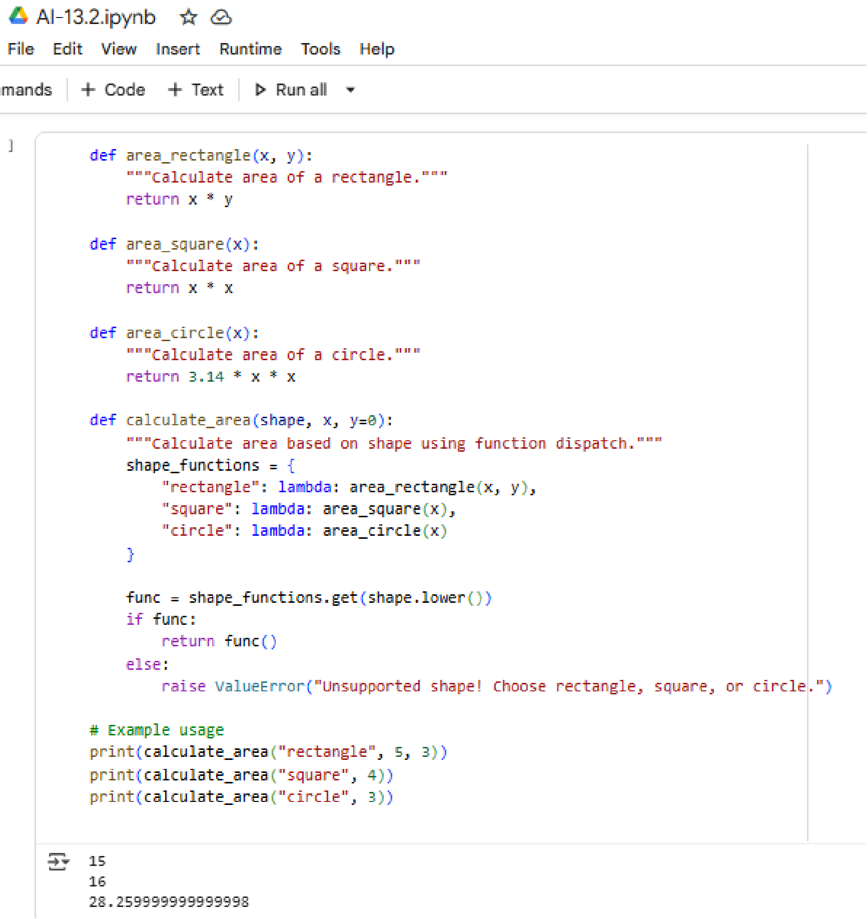
Expected Output

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

Prompt#1:

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

Code#1:


AI-13.2.ipynb
File Edit View Insert Runtime Tools Help
mands + Code + Text ▷ Run all ▾

```python
def area_rectangle(x, y):
    """Calculate area of a rectangle."""
    return x * y

def area_square(x):
    """Calculate area of a square."""
    return x * x

def area_circle(x):
    """Calculate area of a circle."""
    return 3.14 * x * x

def calculate_area(shape, x, y=0):
    """Calculate area based on shape using function dispatch."""
    shape_functions = {
        "rectangle": lambda: area_rectangle(x, y),
        "square": lambda: area_square(x),
        "circle": lambda: area_circle(x)
    }

    func = shape_functions.get(shape.lower())
    if func:
        return func()
    else:
        raise ValueError("Unsupported shape! Choose rectangle, square, or circle.")

# Example usage
print(calculate_area("rectangle", 5, 3))
print(calculate_area("square", 4))
print(calculate_area("circle", 3))
```

```
15
16
28.259999999999998
```

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
```

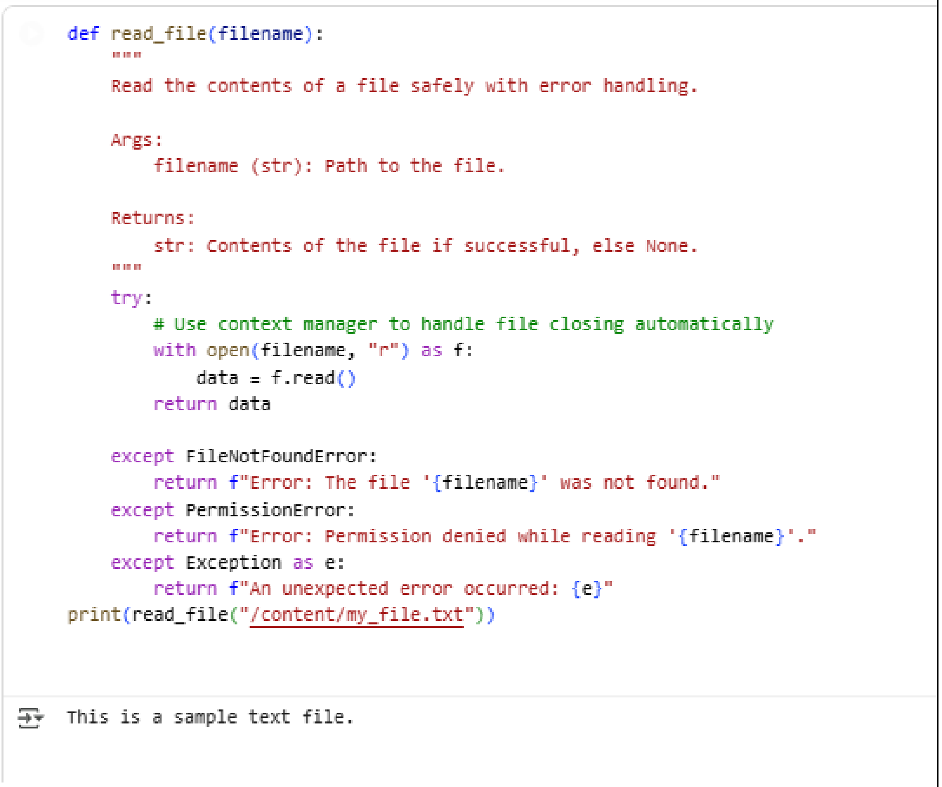
Expected Output:

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):
    """
    Read the contents of a file safely with error handling.

    Args:
        filename (str): Path to the file.

    Returns:
        str: Contents of the file if successful, else None.
    """
    try:
        # Use context manager to handle file closing automatically
        with open(filename, "r") as f:
            data = f.read()
        return data

    except FileNotFoundError:
        return f"Error: The file '{filename}' was not found."
    except PermissionError:
        return f"Error: Permission denied while reading '{filename}'."
    except Exception as e:
        return f"An unexpected error occurred: {e}"
print(read_file("/content/my_file.txt"))
```

```
This is a sample text file.
```

Task Description #3 – Complex Refactoring

Task: Provide this legacy class to AI for readability and modularity improvements:

Python Code

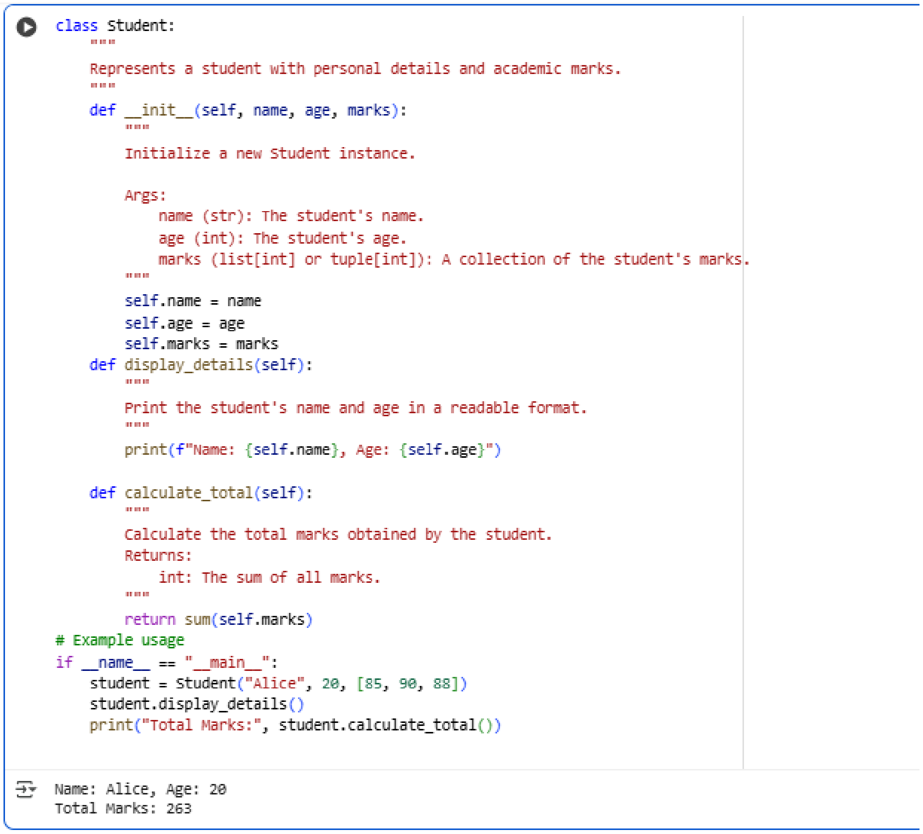
```
class Student:
    def __init__(self, n, a, m1, m2, m3):
        self.n = n
        self.a = a
        self.m1 = m1
        self.m2 = m2
        self.m3 = m3
    def details(self):
        print("Name:", self.n, "Age:", self.a)
    def total(self):
        return self.m1+self.m2+self.m3
```

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:



```python
class Student:
    """
    Represents a student with personal details and academic marks.
    """
    def __init__(self, name, age, marks):
        """
        Initialize a new Student instance.

        Args:
            name (str): The student's name.
            age (int): The student's age.
            marks (list[int] or tuple[int]): A collection of the student's marks.
        """
        self.name = name
        self.age = age
        self.marks = marks
    def display_details(self):
        """
        Print the student's name and age in a readable format.
        """
        print(f"Name: {self.name}, Age: {self.age}")

    def calculate_total(self):
        """
        Calculate the total marks obtained by the student.
        Returns:
            int: The sum of all marks.
        """
        return sum(self.marks)
# Example usage
if __name__ == "__main__":
    student = Student("Alice", 20, [85, 90, 88])
    student.display_details()
    print("Total Marks:", student.calculate_total())
```

```
Name: Alice, Age: 20
Total Marks: 263
```

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:

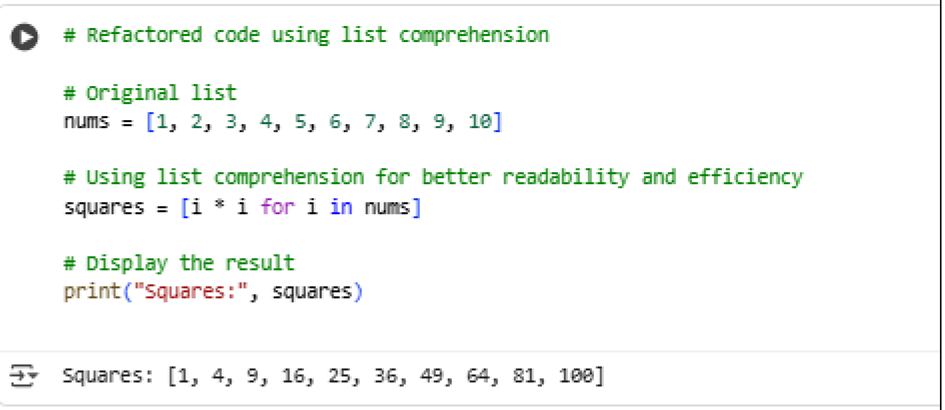
```
    squares.append(i * i)
```

Expected Output: AI suggested a **list comprehension**

Prompt#4:

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

Code#4:



```
# Refactored code using list comprehension

# Original list
nums = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

# Using list comprehension for better readability and efficiency
squares = [i * i for i in nums]

# Display the result
print("Squares:", squares)
```

```
Squares: [1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```