SCHOOLOFCOMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE		DEPARTMENTOFCOMPUTER SCIENCE ENGINEERING	
Program Name: M. Tech/MCA/MSC		**Assignment Type: Lab**	**AcademicYear:**2025-2026
Course Coordinator Name		Venkataramana Veeramsetty	
Course Code	24CS002PC215	**Course Title**	AI Assisted Problem Solving Using Python
Year/Sem	II/I	**Regulation**	R24
Date and Day of Assignment	Week5- Tuesday	**Time(s)**	
Duration	2 Hours	**Applicable to 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). • 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**	Week5-Tuesday

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.

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:

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
```

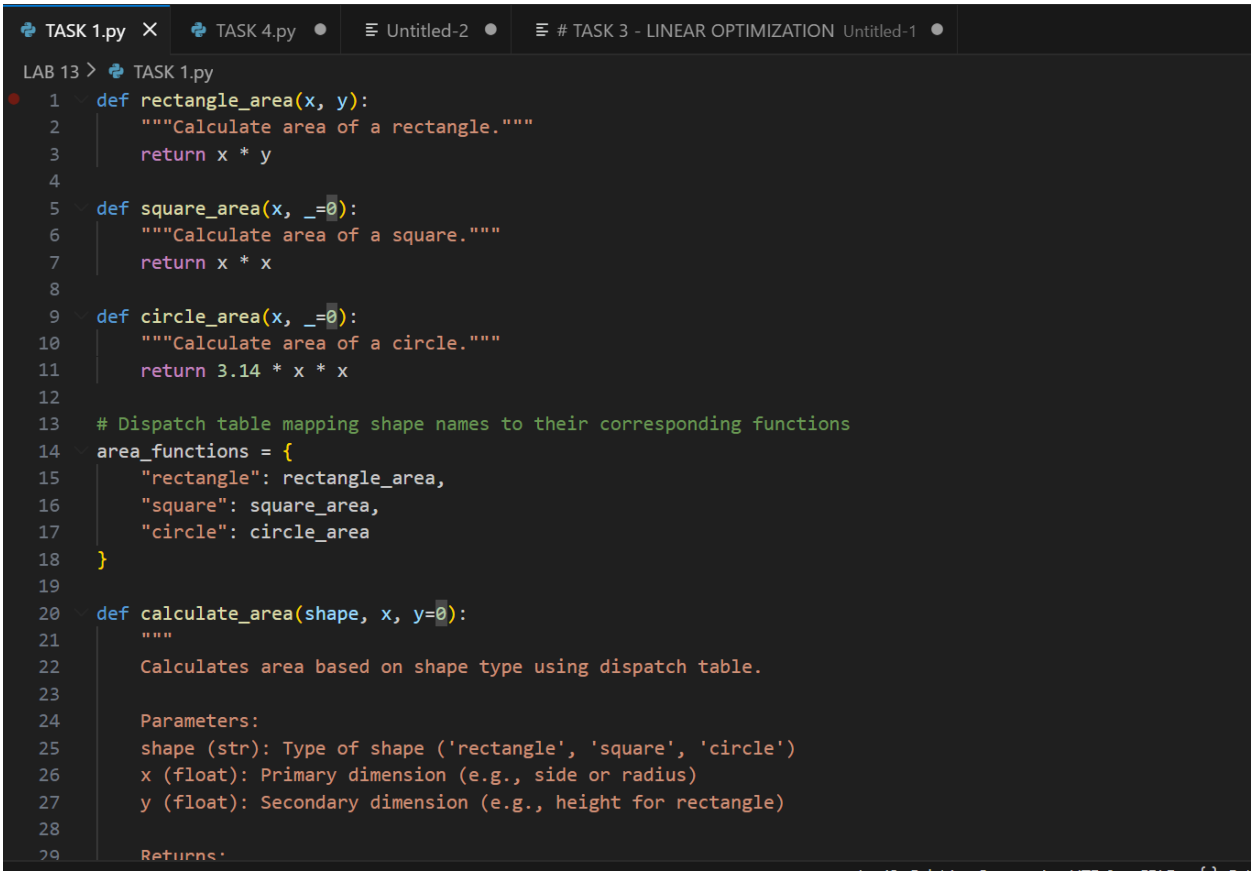
	self.m2 = m2
	self.m3 = m3
	def details(self):
	print("Name:", self.n, "Age:", self.a)
	def total(self):
	return self.m1+self.m2+self.m3
	
	Expected Output:
	
	• AI improves naming (`name`, `age`, `marks`).
	• Adds docstrings.
	• Improves print readability.
	• Possibly uses `sum(self.marks)` if marks 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**	

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
```



```python
def calculate_area(shape, x, y=0):
    Returns:
    float: Calculated area
    """
    func = area_functions.get(shape.lower())
    if func:
        return func(x, y)
    else:
        raise ValueError(f"Unsupported shape: {shape}")

# Sample usage with user input
if __name__ == "__main__":
    try:
        shape = input("Enter shape (rectangle, square, circle): ").strip().lower()
        x = float(input("Enter first dimension (e.g., side or radius): "))
        y = 0
        if shape == "rectangle":
            y = float(input("Enter second dimension (e.g., height): "))

        area = calculate_area(shape, x, y)
        print(f"The area of the {shape} is: {area}")
    except ValueError as ve:
        print(f"Input error: {ve}")
    except Exception as e:
        print(f"Unexpected error: {e}")
```

PRACTICAL OUTPUT:

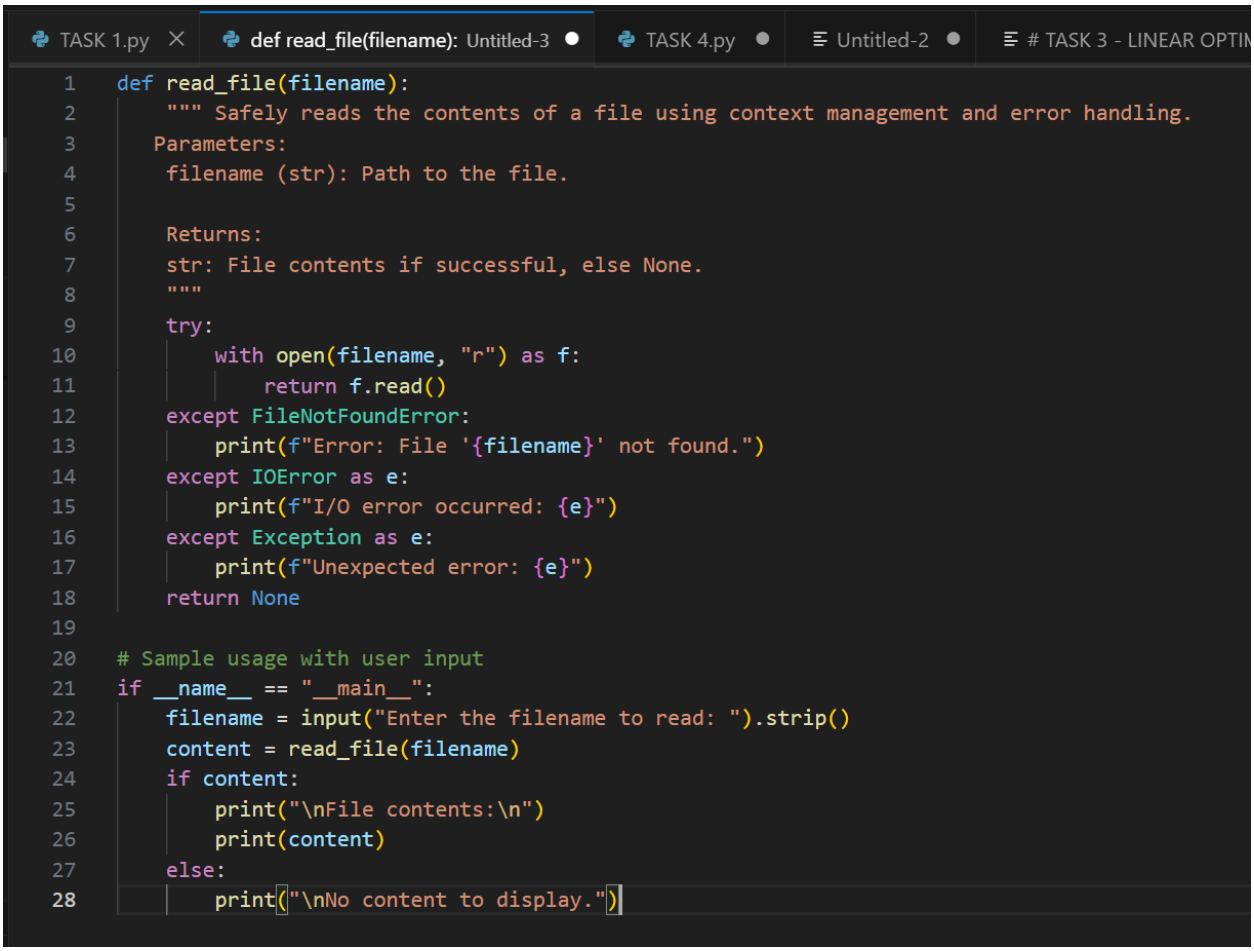
```
TERMINAL                                                    Code - Moha

PS C:\Users\rimsha\OneDrive\Desktop\Mohammed Farnas Ali Mudabbir> python -u "c:\Users\rimsha\OneDrive
 Mudabbir\tempCodeRunnerFile.python"
Enter shape (rectangle, square, circle): square
Enter first dimension (e.g., side or radius): 6
The area of the square is: 36.0
PS C:\Users\rimsha\OneDrive\Desktop\Mohammed Farnas Ali Mudabbir>
```

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
```



PRACTICAL OUTPUT:

Task Description #3 – Complex Refactoring

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

Python Code

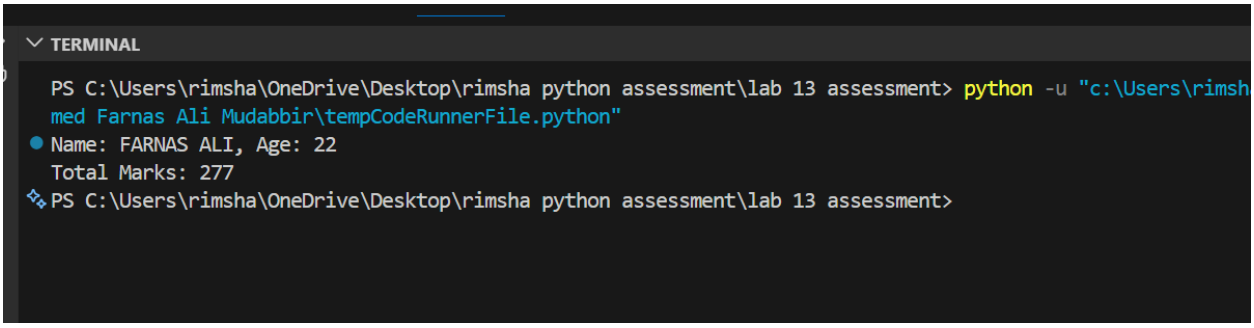
```
class Student:
    def __init__(self, n, a, m1, m2, m3):
        self.n = n
        self.a = a
        self.m1 = m1
        self.m2 = m2
        self.m3 = m3
    def details(self):
        print("Name:", self.n, "Age:", self.a)
    def total(self):
        return self.m1+self.m2+self.m3
```

TASK 1.py | def read_file(filename): Untitled-3 | # TASK 3 - COMPLEC REFACTORING (Class Im Untitled-4 | TASK 4.py

```python
# TASK 3 - COMPLEC REFACTORING (Class Improvement)
# LEGACY 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
# REFACTORED CODE
class Student:
    """Represents a student and their academic details."""
    def __init__(self, name, age, marks):
        self.name = name
        self.age = age
        self.marks = marks  # list of marks

    def show_details(self):
        print(f"Name: {self.name}, Age: {self.age}")

    def total_marks(self):
        return sum(self.marks)
# Creating a student object
student1 = Student("FARNAS ALI", 22, [93, 88, 96])

```

```python
        self.marks = marks  # list of marks

    def show_details(self):
        print(f"Name: {self.name}, Age: {self.age}")

    def total_marks(self):
        return sum(self.marks)
# Creating a student object
student1 = Student("FARNAS ALI", 22, [93, 88, 96])

# Printing student details
student1.show_details()

# Printing total marks
print("Total Marks:", student1.total_marks())
```

PRACTICAL OUTPUT:

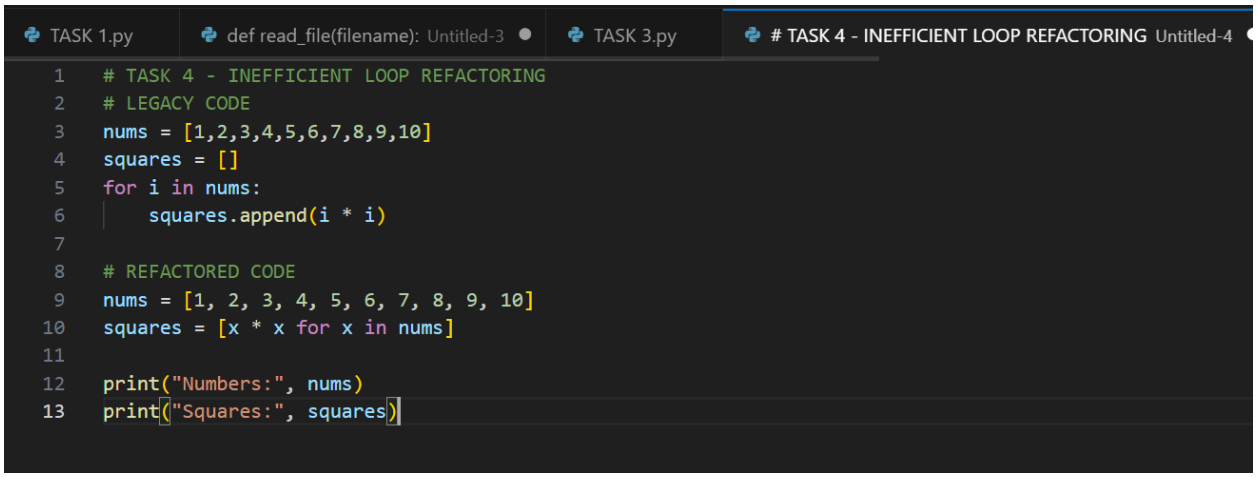


Task Description #4 – Inefficient Loop Refactoring

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)
```



PRACTICAL OUTPUT: