

SCHOOL OF COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE		DEPARTMENT OF COMPUTER SCIENCE ENGINEERING	
Program Name: M. Tech/MCA/MSC		Assignment Type: Lab	Academic Year: 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	Week 5- Tuesday	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	<p><b>Lab 13 – Code Refactoring: Improving Legacy Code with AI Suggestions</b></p> <p><b>Lab Objectives</b></p> <ul style="list-style-type: none"> <li>To introduce the concept of code refactoring and why it matters (readability, maintainability, performance).</li> <li>To practice using AI tools for identifying and suggesting improvements in legacy code.</li> <li>To evaluate the before vs. after versions for clarity, performance, and correctness.</li> <li>To reinforce responsible AI-assisted coding practices (avoiding over-reliance, validating outputs).</li> </ul> <p><b>Learning Outcomes</b> After completing this lab, students will be able to:</p> <ol style="list-style-type: none"> <li>Use AI to analyze and refactor poorly written Python code.</li> <li>Improve code <b>readability, efficiency, and error handling</b>.</li> <li>Document AI-suggested improvements through comments and explanations.</li> <li>Apply refactoring strategies without changing functionality.</li> <li>Critically reflect on AI's refactoring suggestions.</li> </ol> <p><b>Task Description #1 – Remove Repetition</b></p>		Week 5- Tuesday

	<p>Task: Provide AI with the following redundant code and ask it to refactor</p> <p><b>Python Code</b></p> <pre>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</pre> <p><b>Expected Output</b></p> <ul style="list-style-type: none"> <li>• Refactored version with dictionary-based dispatch or separate functions.</li> <li>• Cleaner and modular design.</li> </ul> <p><b>Task Description #2 – Error Handling in Legacy Code</b></p> <p>Task: Legacy function without proper error handling</p> <p><b>Python Code</b></p> <pre>def read_file(filename):     f = open(filename, "r")     data = f.read()     f.close()     return data</pre> <p><b>Expected Output:</b></p> <p>AI refactors with with open() and try-except:</p> <p><b>Task Description #3 – Complex Refactoring</b></p> <p>Task: Provide this legacy class to AI for readability and modularity improvements:</p> <p><b>Python Code</b></p> <pre>class Student:     def __init__(self, n, a, m1, m2, m3):         self.n = n         self.a = a         self.m1 = m1</pre>	
--	---	--

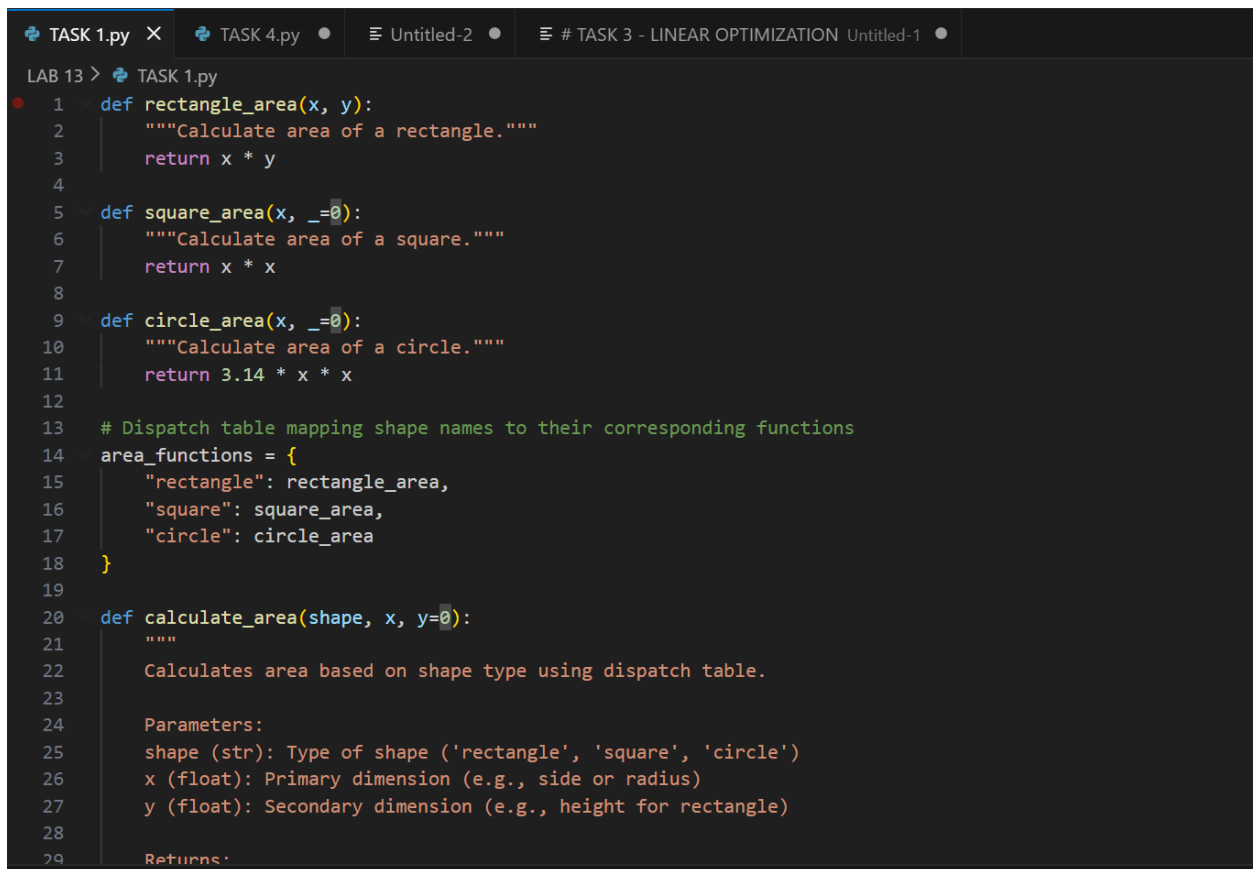
	<pre> self.m2 = m2 self.m3 = m3 def details(self):     print("Name:", self.n, "Age:", self.a) def total(self):     return self.m1+self.m2+self.m3 </pre> <p><b>Expected Output:</b></p> <ul style="list-style-type: none"> <li>• AI improves naming (name, age, marks).</li> <li>• Adds docstrings.</li> <li>• Improves print readability.</li> <li>• Possibly uses <code>sum(self.marks)</code> if marks stored in a list.</li> </ul> <p><b>Task Description #4 – Inefficient Loop Refactoring</b>  Task: Refactor this inefficient loop with AI help</p> <p><b>Python Code</b></p> <pre> nums = [1,2,3,4,5,6,7,8,9,10] squares = [] for i in nums:     squares.append(i * i) </pre> <p><b>Expected Output:</b> AI suggested a <b>list comprehension</b></p>	

## 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
```



The screenshot shows a code editor with a dark theme. The top bar displays several tabs: 'TASK 1.py' (active), 'TASK 4.py', 'Untitled-2', and '# TASK 3 - LINEAR OPTIMIZATION' (disabled). The main editor area shows the following Python code:

```
LAB 13 > TASK 1.py  
1 def rectangle_area(x, y):  
2     """Calculate area of a rectangle."""  
3     return x * y  
4  
5 def square_area(x, _=0):  
6     """Calculate area of a square."""  
7     return x * x  
8  
9 def circle_area(x, _=0):  
10    """Calculate area of a circle."""  
11    return 3.14 * x * x  
12  
13 # Dispatch table mapping shape names to their corresponding functions  
14 area_functions = {  
15     "rectangle": rectangle_area,  
16     "square": square_area,  
17     "circle": circle_area  
18 }  
19  
20 def calculate_area(shape, x, y=0):  
21     """  
22     Calculates area based on shape type using dispatch table.  
23  
24     Parameters:  
25     shape (str): Type of shape ('rectangle', 'square', 'circle')  
26     x (float): Primary dimension (e.g., side or radius)  
27     y (float): Secondary dimension (e.g., height for rectangle)  
28  
29     Returns:
```

```
TASK 1.py X TASK 4.py • Untitled-2 • # TASK 3 - LINEAR OPTIMIZATION Untitled-1 •
LAB 13 > TASK 1.py > ...
20 def calculate_area(shape, x, y=0):
21
22     Returns:
23     float: Calculated area
24     """
25
26     func = area_functions.get(shape.lower())
27     if func:
28         return func(x, y)
29     else:
30         raise ValueError(f"Unsupported shape: {shape}")
31
32 # Sample usage with user input
33 if __name__ == "__main__":
34     try:
35         shape = input("Enter shape (rectangle, square, circle): ").strip().lower()
36         x = float(input("Enter first dimension (e.g., side or radius): "))
37         y = 0
38         if shape == "rectangle":
39             y = float(input("Enter second dimension (e.g., height): "))
40
41         area = calculate_area(shape, x, y)
42         print(f"The area of the {shape} is: {area}")
43     except ValueError as ve:
44         print(f"Input error: {ve}")
45     except Exception as e:
46         print(f"Unexpected error: {e}")
47
48
49
50
51
52
```

## PRACTICAL OUTPUT:

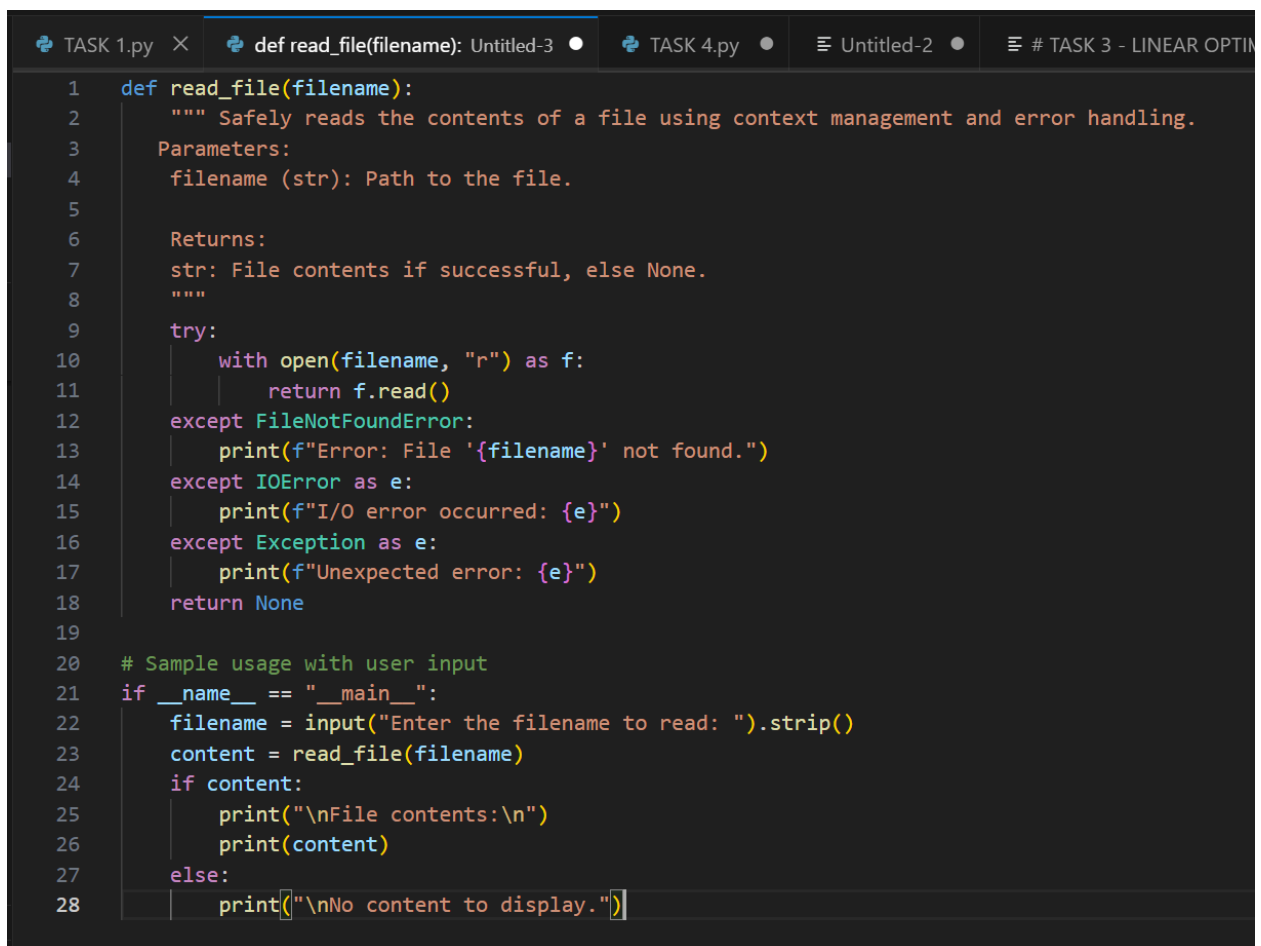
```
▼ TERMINAL Code - Moha
PS C:\Users\rimsha\OneDrive\Desktop\Mohammed Farnas Ali Mudabbir> python -u "c:\Users\rimsha\OneDrive
Mudabbir\tempCodeRunnerFile.python"
● Enter shape (rectangle, square, circle): square
Enter first dimension (e.g., side or radius): 6
The area of the square is: 36.0
PS C:\Users\rimsha\OneDrive\Desktop\Mohammed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



```
TASK 1.py × def read_file(filename): Untitled-3 TASK 4.py Untitled-2 # TASK 3 - LINEAR OPTIM  
1 def read_file(filename):  
2     """ Safely reads the contents of a file using context management and error handling.  
3     Parameters:  
4     filename (str): Path to the file.  
5  
6     Returns:  
7     str: File contents if successful, else None.  
8     """  
9     try:  
10         with open(filename, "r") as f:  
11             return f.read()  
12     except FileNotFoundError:  
13         print(f"Error: File '{filename}' not found.")  
14     except IOError as e:  
15         print(f"I/O error occurred: {e}")  
16     except Exception as e:  
17         print(f"Unexpected error: {e}")  
18     return None  
19  
20 # Sample usage with user input  
21 if __name__ == "__main__":  
22     filename = input("Enter the filename to read: ").strip()  
23     content = read_file(filename)  
24     if content:  
25         print("\nFile contents:\n")  
26         print(content)  
27     else:  
28         print("\nNo content to display.")
```

## PRACTICAL OUTPUT:

```
FARNAS_ALI@LAPTOP-UPQ9TB8M MINGW64 ~/OneDrive/Desktop/AASE
$ C:/Python313/python.exe "c:/Users/FARNAS_ALI/OneDrive/Desktop/AASE
Enter the filename to read: hello.txt

File contents:

HEL HELLO HELLO ALL

FARNAS_ALI@LAPTOP-UPQ9TB8M MINGW64 ~/OneDrive/Desktop/AASE
$
```

### 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

1  # TASK 3 - COMPLEC REFACTORING (Class Improvement)
2  # LEGACY CODE
3  class Student:
4      def __init__(self, n, a, m1, m2, m3):
5          self.n = n
6          self.a = a
7          self.m1 = m1
8          self.m2 = m2
9          self.m3 = m3
10     def details(self):
11         print("Name:", self.n, "Age:", self.a)
12
13     def total(self):
14         return self.m1+self.m2+self.m3
15  # REFACTORED CODE
16  class Student:
17      """Represents a student and their academic details."""
18     def __init__(self, name, age, marks):
19         self.name = name
20         self.age = age
21         self.marks = marks # list of marks
22
23     def show_details(self):
24         print(f"Name: {self.name}, Age: {self.age}")
25
26     def total_marks(self):
27         return sum(self.marks)
28  # Creating a student object
29  student1 = Student("FARNAS ALI", 22, [93, 88, 96])
30
```

```
21         self.marks = marks # list of marks
22
23     def show_details(self):
24         print(f"Name: {self.name}, Age: {self.age}")
25
26     def total_marks(self):
27         return sum(self.marks)
28  # Creating a student object
29  student1 = Student("FARNAS ALI", 22, [93, 88, 96])
30
31  # Printing student details
32  student1.show_details()
33
34  # Printing total marks
35  print("Total Marks:", student1.total_marks())
```



## PRACTICAL OUTPUT:

```
▼ TERMINAL
PS C:\Users\rimsha\OneDrive\Desktop\rimsha python assessment\lab 13 assessment> python -u "c:\Users\rimsha\OneDrive\Desktop\rimsha python assessment\lab 13 assessment\tempCodeRunnerFile.python"
● Name: FARNAS ALI, Age: 22
  Total Marks: 277
❖ PS C:\Users\rimsha\OneDrive\Desktop\rimsha python assessment\lab 13 assessment>
```

### 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)
```

```
TASK 1.py  def read_file(filename): Untitled-3  TASK 3.py  # TASK 4 - INEFFICIENT LOOP REFACTORING Untitled-4
1  # TASK 4 - INEFFICIENT LOOP REFACTORING
2  # LEGACY CODE
3  nums = [1,2,3,4,5,6,7,8,9,10]
4  squares = []
5  for i in nums:
6      squares.append(i * i)
7
8  # REFACTORED CODE
9  nums = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
10 squares = [x * x for x in nums]
11
12 print("Numbers:", nums)
13 print("Squares:", squares)
```

## PRACTICAL OUTPUT:

```
PS C:\Users\rimsha\OneDrive\Desktop\rimsha python assessment\lab 13 assessment> python -u "c:\Users\rimsha\OneDrive\Desktop\rimsha python assessment\lab 13 assessment\med Farnas Ali Mudabbir\tempCodeRunnerFile.python"
Numbers: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
Squares: [1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
PS C:\Users\rimsha\OneDrive\Desktop\rimsha python assessment\lab 13 assessment>
```