

SCHOOL OF COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE		DEPARTMENT OF COMPUTER SCIENCE ENGINEERING		
Program Name: M. Tech/MCA/MSC		Assignment Type: Lab		
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		
Assignment Number: 13.3 (Present assignment number) / 24 (Total number of assignments)				
Q.No.	Question			
1	<p>Lab 13 – Code Refactoring: Improving Legacy Code with AI Suggestions</p> <p>Lab Objectives</p> <ul style="list-style-type: none"> 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). <p>Learning Outcomes</p> <p>After completing this lab, students will be able to:</p> <ol style="list-style-type: none"> Use AI to analyze and refactor poorly written Python code. Improve code readability, efficiency, and error handling. Document AI-suggested improvements through comments and explanations. Apply refactoring strategies without changing functionality. Critically reflect on AI's refactoring suggestions. <p>Task Description #1 – Remove Repetition</p>			

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

```

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)

```

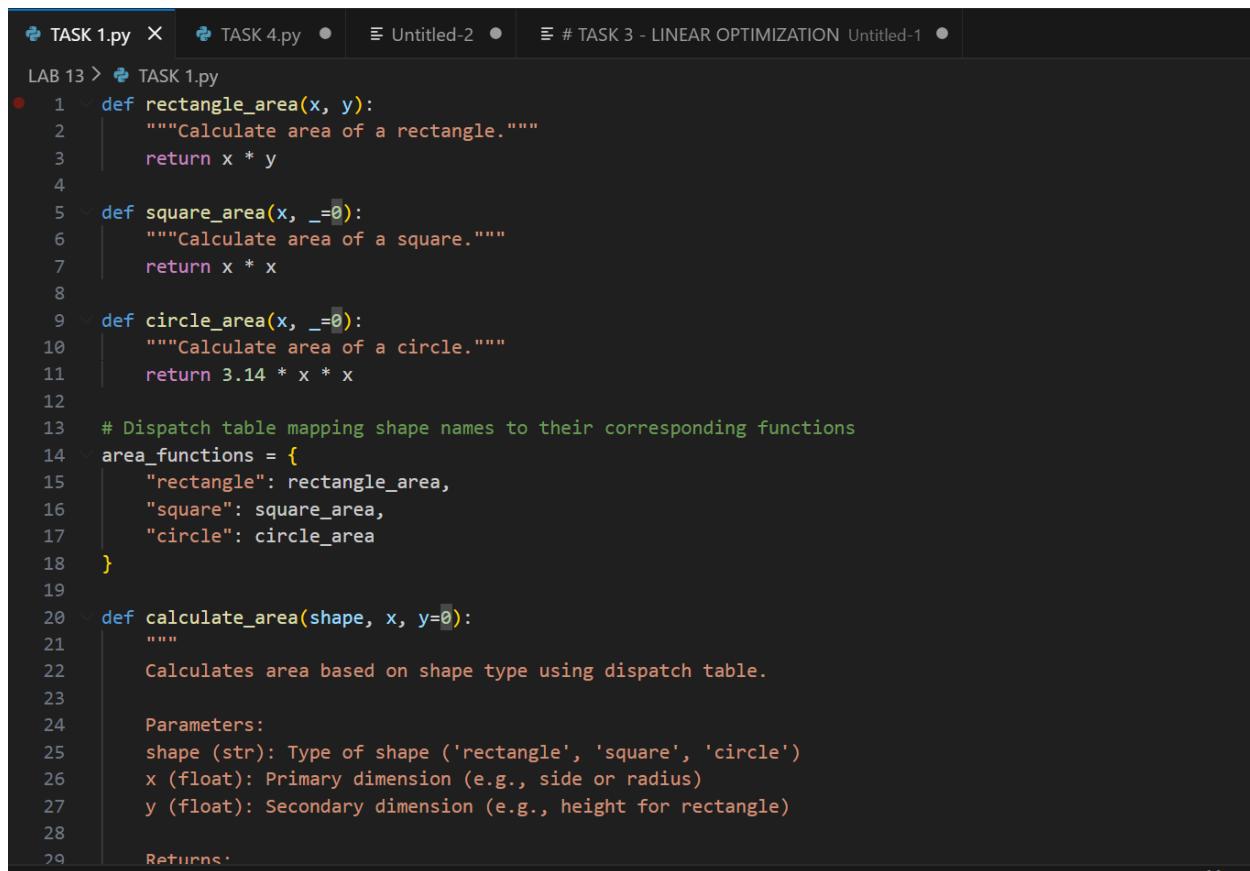
Expected Output: AI suggested a **list comprehension**

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
```



The screenshot shows a code editor with multiple tabs at the top: 'TASK 1.py' (selected), 'TASK 4.py', 'Untitled-2', and '# TASK 3 - LINEAR OPTIMIZATION Untitled-1'. The code in 'TASK 1.py' is as follows:

```
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:
```

The screenshot shows a code editor with four tabs at the top: "TASK 1.py", "TASK 4.py", "Untitled-2", and "# TASK 3 - LINEAR OPTIMIZATION Untitled-1". The main pane displays Python code for calculating the area of shapes based on user input. The code defines a function `calculate_area` that returns the calculated area. It uses a dictionary `area_functions` to map shape names to their respective area calculation functions. It handles user input for shape type and dimensions, and prints the result or handles exceptions.

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

PRACTICAL OUTPUT:

The screenshot shows a terminal window with a command-line interface. The user runs the script "TASK 1.py" using Python. They are prompted to enter a shape ("square") and a dimension ("8"). The program calculates the area of the square and prints it out ("The area of the square is: 64.0").

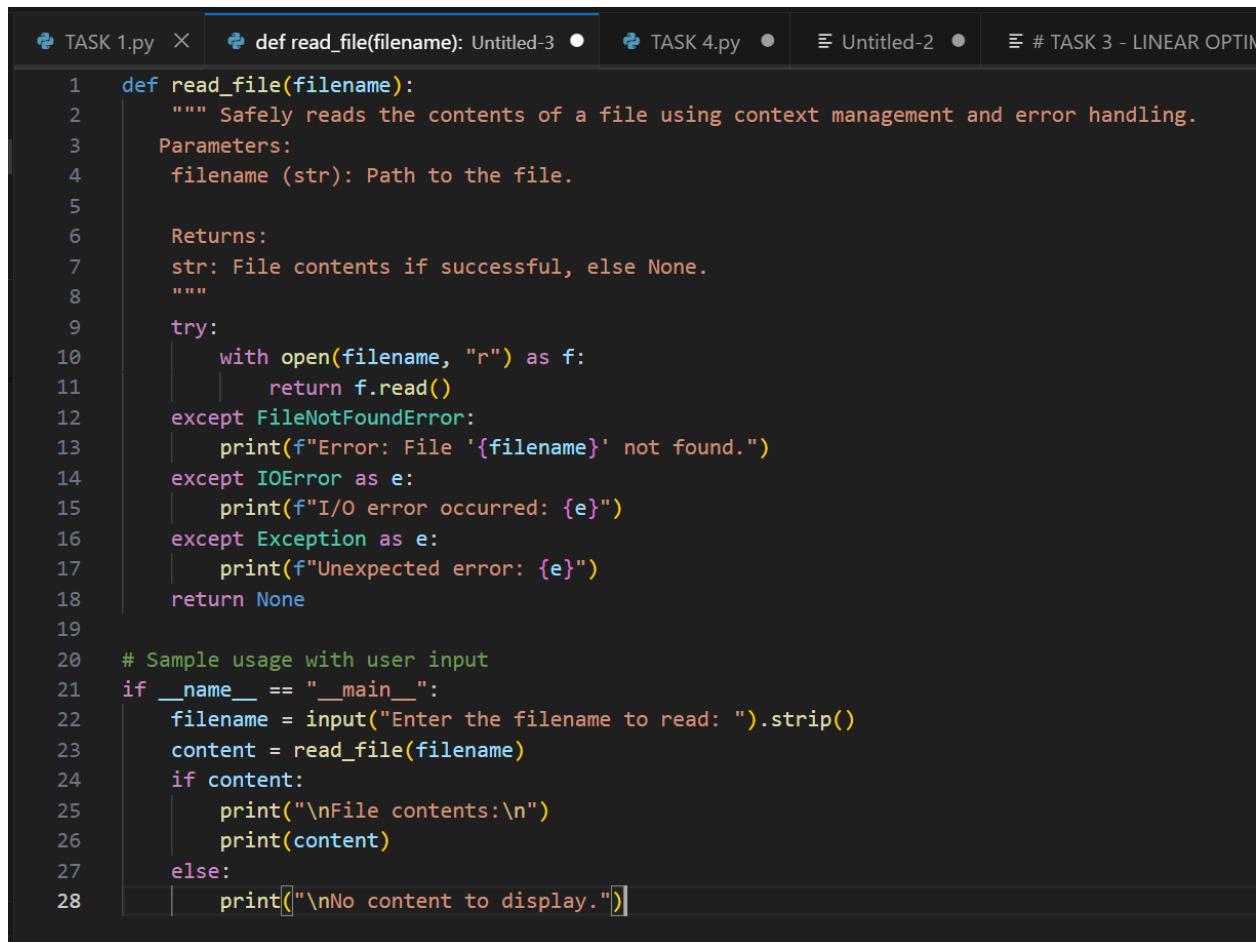
```
> TERMINAL
PS C:\Users\rimsha\OneDrive\Desktop\Mohammed Farnas Ali Mudabbir\LAB 13> python -u "c:\Users\rimsha\OneDrive\Desktop\Mohammed Farnas Ali Mudabbir\LAB 13\TASK 1.py"
Enter shape (rectangle, square, circle): square
Enter first dimension (e.g., side or radius): 8
The area of the square is: 64.0
PS C:\Users\rimsha\OneDrive\Desktop\Mohammed Farnas Ali Mudabbir\LAB 13>
```

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 screenshot shows a code editor with multiple tabs at the top: 'TASK 1.py' (closed), 'def read_file(filename): Untitled-3' (active), 'TASK 4.py' (closed), 'Untitled-2' (closed), and '# TASK 3 - LINEAR OPTIMIZATION' (closed). The active tab contains the following Python code:

```
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-UPQ9TBBM 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-UPQ9TBBM 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\Mohammed Farnas Ali Mudabbir\LAB 13> python -u "c:\Users\rimsha\Ali Mudabbir\tempCodeRunnerFile.py"
Name: FARNAS ALI, Age: 22
Total Marks: 277
❖ PS C:\Users\rimsha\OneDrive\Desktop\Mohammed Farnas Ali Mudabbir\LAB 13>
```

Task Description #4 – Inefficient Loop Refactoring

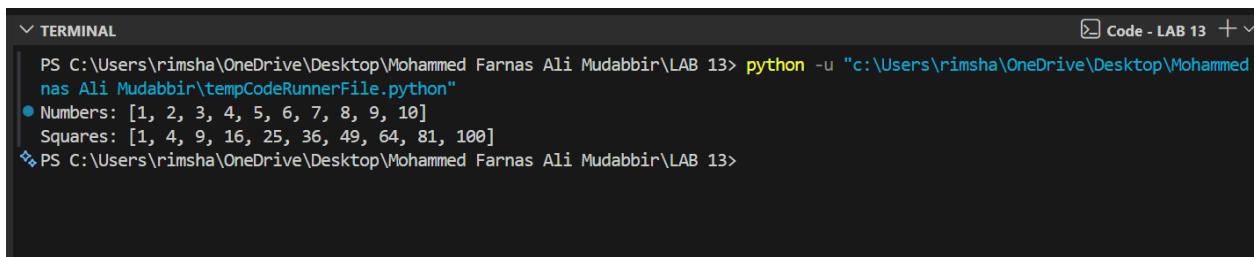
Task: Refactor this inefficient loop with AI help

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:



The screenshot shows a terminal window titled "TERMINAL" with the identifier "Code - LAB 13". The terminal displays the following command and its output:

```
PS C:\Users\rimsha\OneDrive\Desktop\Mohammed Farnas Ali Mudabbir\LAB 13> python -u "c:\Users\rimsha\OneDrive\Desktop\Mohammed Farnas Ali Mudabbir\tempCodeRunnerFile.py"
● 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\Mohammed Farnas Ali Mudabbir\LAB 13>
```

The terminal shows the execution of a Python script named "tempCodeRunnerFile.py". The script prints two lists: "Numbers" containing integers from 1 to 10, and "Squares" containing the squares of those numbers.