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** Task: Provide AI with the following redundant code and ask it to refactor	Week5- Tuesday

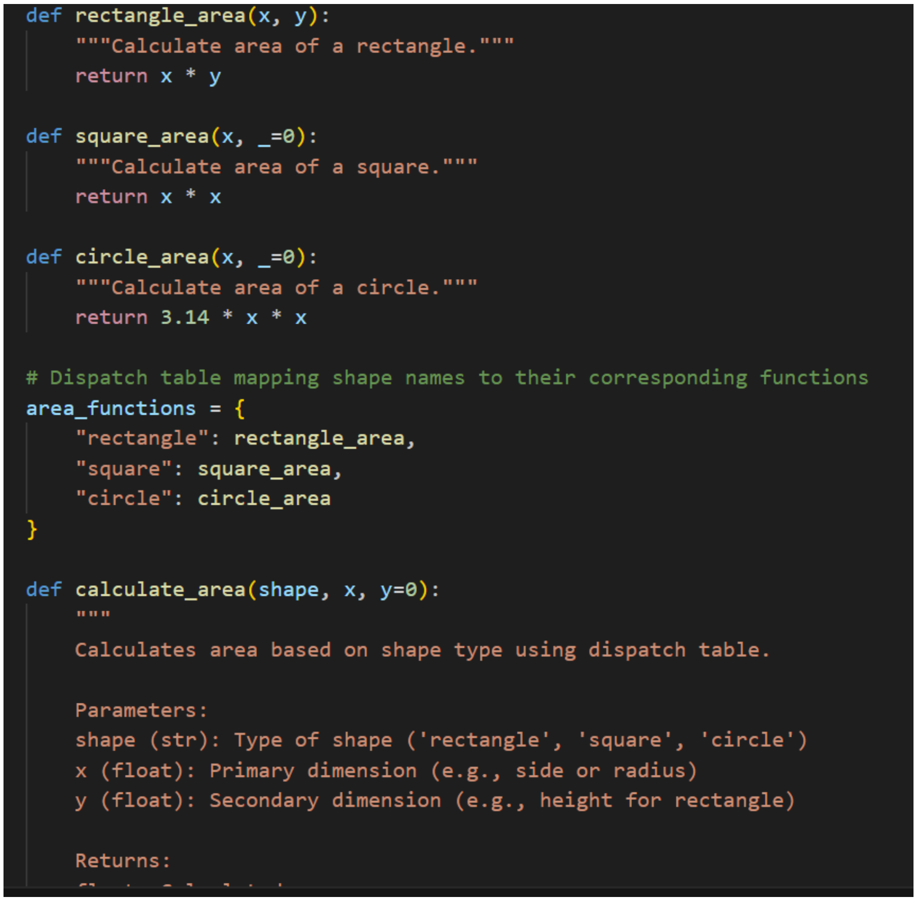
Python Code

```
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
```

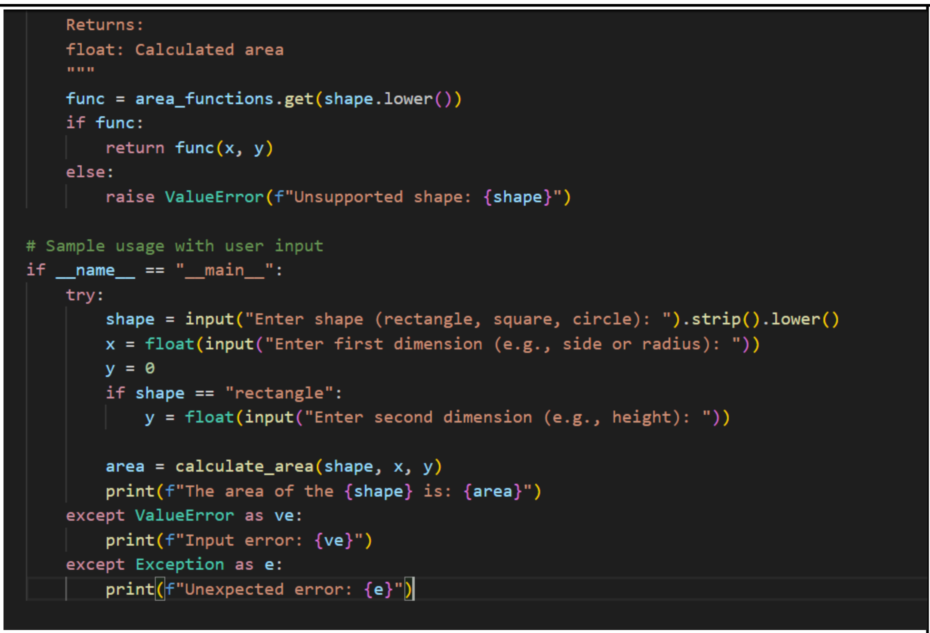
Expected Output

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

CODE:

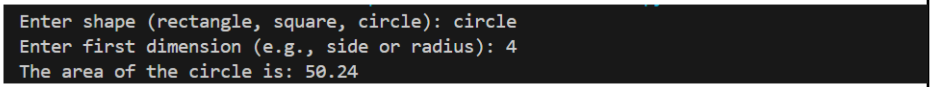


```
def rectangle_area(x, y):
    """Calculate area of a rectangle."""
    return x * y

def square_area(x, _=0):
    """Calculate area of a square."""
    return x * x

def circle_area(x, _=0):
    """Calculate area of a circle."""
    return 3.14 * x * x

# Dispatch table mapping shape names to their corresponding functions
area_functions = {
    "rectangle": rectangle_area,
    "square": square_area,
    "circle": circle_area
}

def calculate_area(shape, x, y=0):
    """
    Calculates area based on shape type using dispatch table.

    Parameters:
    shape (str): Type of shape ('rectangle', 'square', 'circle')
    x (float): Primary dimension (e.g., side or radius)
    y (float): Secondary dimension (e.g., height for rectangle)

    Returns:
```



```
    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}")
```

OUTPUT:



```
Enter shape (rectangle, square, circle): circle
Enter first dimension (e.g., side or radius): 4
The area of the circle is: 50.24
```

Task Description #2 – Error Handling in Legacy Code

Task: Legacy function without proper error handling

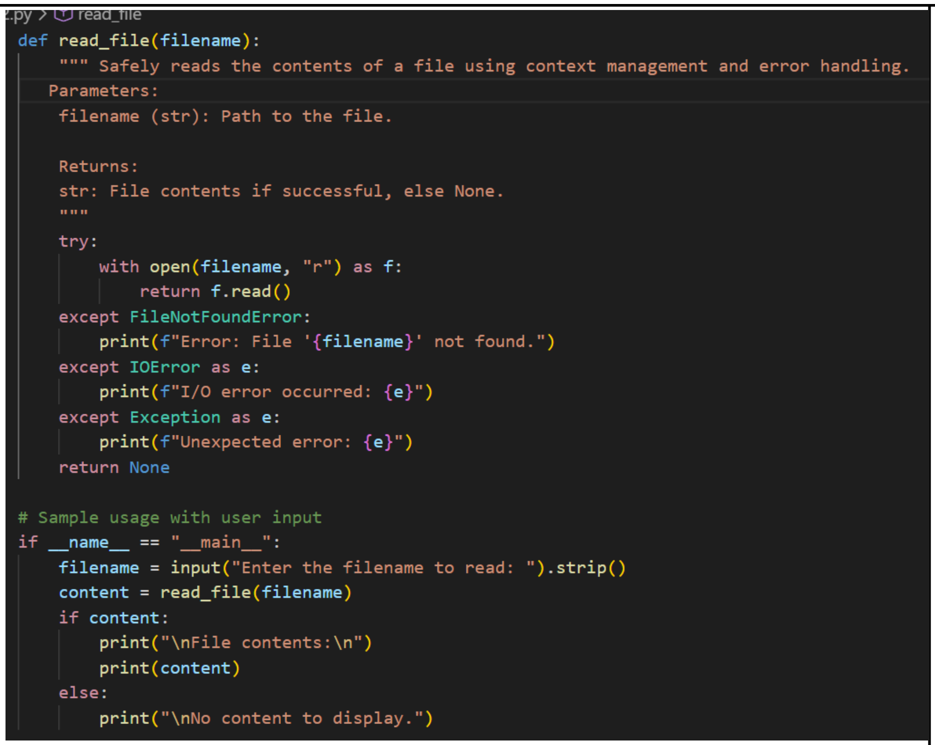
Python Code

```
def read_file(filename):
    f = open(filename, "r")
    data = f.read()
    f.close()
    return data
```

Expected Output:

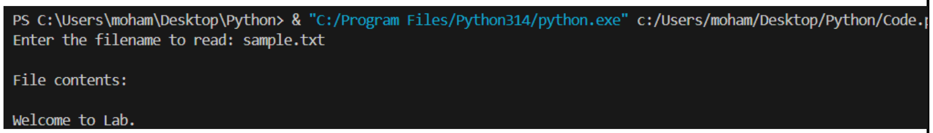
AI refactors with with open() and try-except:

CODE:



```
.py > read_file
def read_file(filename):
    """ Safely reads the contents of a file using context management and error handling.
    Parameters:
    filename (str): Path to the file.

    Returns:
    str: File contents if successful, else None.
    """
    try:
        with open(filename, "r") as f:
            return f.read()
    except FileNotFoundError:
        print(f"Error: File '{filename}' not found.")
    except IOError as e:
        print(f"I/O error occurred: {e}")
    except Exception as e:
        print(f"Unexpected error: {e}")
    return None

# Sample usage with user input
if __name__ == "__main__":
    filename = input("Enter the filename to read: ").strip()
    content = read_file(filename)
    if content:
        print("\nFile contents:\n")
        print(content)
    else:
        print("\nNo content to display.")
```

OUTPUT:



```
PS C:\Users\moham\Desktop\Python> & "C:/Program Files/Python314/python.exe" c:/Users/moham/Desktop/Python/Code.p
Enter the filename to read: sample.txt

File contents:

Welcome to Lab.
```

Task Description #3 – Complex Refactoring

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

Python Code

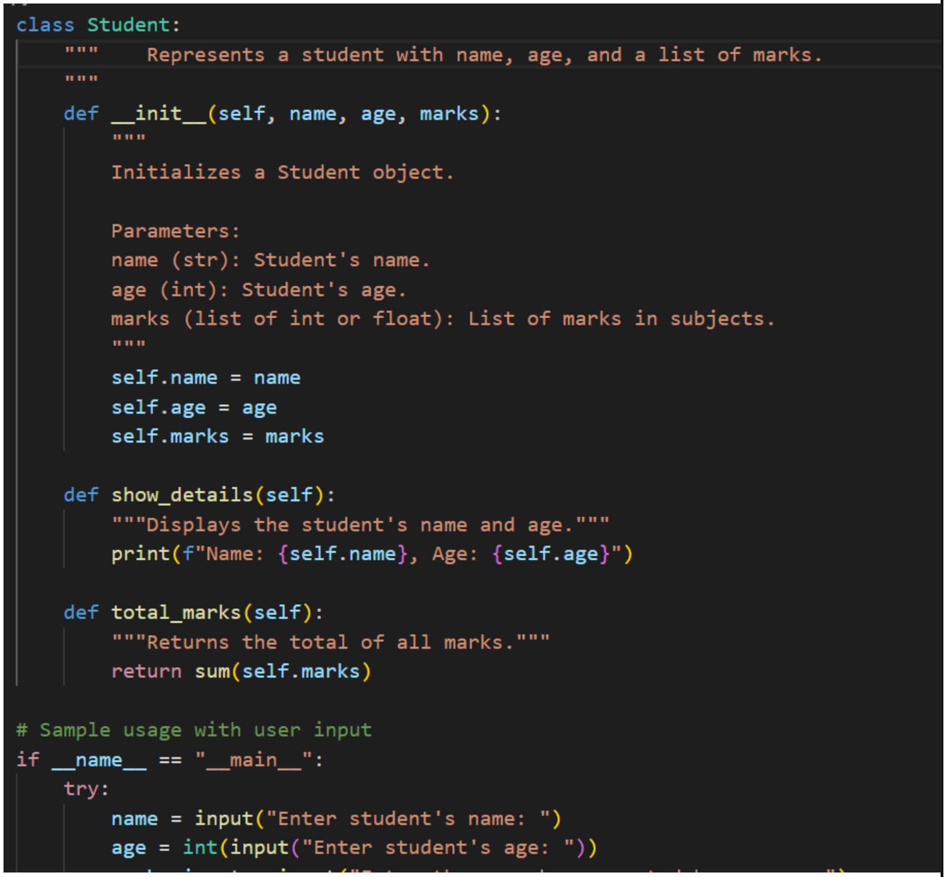
```
class Student:
    def __init__(self, n, a, m1, m2, m3):
        self.n = n
        self.a = a
        self.m1 = m1
        self.m2 = m2
        self.m3 = m3
    def details(self):
        print("Name:", self.n, "Age:", self.a)
    def total(self):
        return self.m1+self.m2+self.m3
```

Expected Output:

- AI improves naming (`name`, `age`, `marks`).
- Adds docstrings.

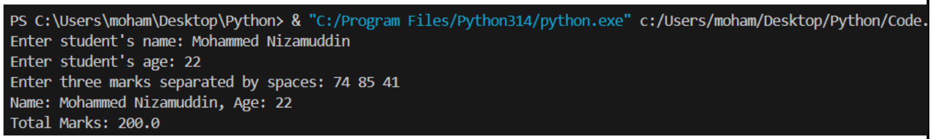
- Improves print readability.
- Possibly uses `sum(self.marks)` if marks stored in a list.

CODE:



```
class Student:
    """    Represents a student with name, age, and a list of marks.
    """
    def __init__(self, name, age, marks):
        """
        Initializes a Student object.

        Parameters:
        name (str): Student's name.
        age (int): Student's age.
        marks (list of int or float): List of marks in subjects.
        """
        self.name = name
        self.age = age
        self.marks = marks

    def show_details(self):
        """Displays the student's name and age."""
        print(f"Name: {self.name}, Age: {self.age}")

    def total_marks(self):
        """Returns the total of all marks."""
        return sum(self.marks)

# Sample usage with user input
if __name__ == "__main__":
    try:
        name = input("Enter student's name: ")
        age = int(input("Enter student's age: "))
```

OUTPUT:



```
PS C:\Users\moham\Desktop\Python> & "C:/Program Files/Python314/python.exe" c:/Users/moham/Desktop/Python/Code.
Enter student's name: Mohammed Nizamuddin
Enter student's age: 22
Enter three marks separated by spaces: 74 85 41
Name: Mohammed Nizamuddin, Age: 22
Total Marks: 200.0
```

Task Description #4 – Inefficient Loop Refactoring

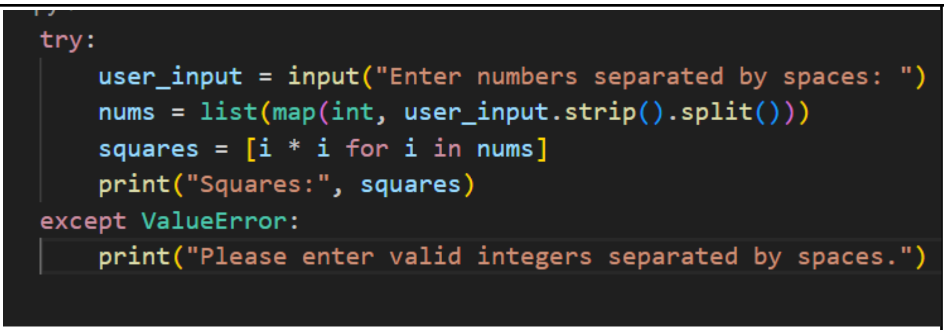
Task: Refactor this inefficient loop with AI help

Python Code

```
nums = [1,2,3,4,5,6,7,8,9,10]
squares = []
for i in nums:
    squares.append(i * i)
```

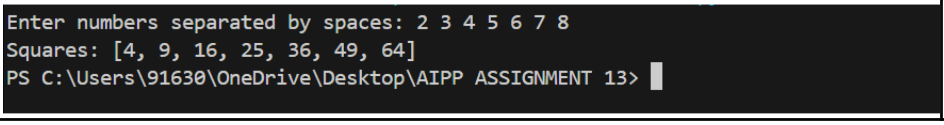
Expected Output: AI suggested a **list comprehension**

CODE:



```
try:
    user_input = input("Enter numbers separated by spaces: ")
    nums = list(map(int, user_input.strip().split()))
    squares = [i * i for i in nums]
    print("Squares:", squares)
except ValueError:
    print("Please enter valid integers separated by spaces.")
```

OUTPUT:



```
Enter numbers separated by spaces: 2 3 4 5 6 7 8
Squares: [4, 9, 16, 25, 36, 49, 64]
PS C:\Users\91630\OneDrive\Desktop\AIPP ASSIGNMENT 13>
```