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

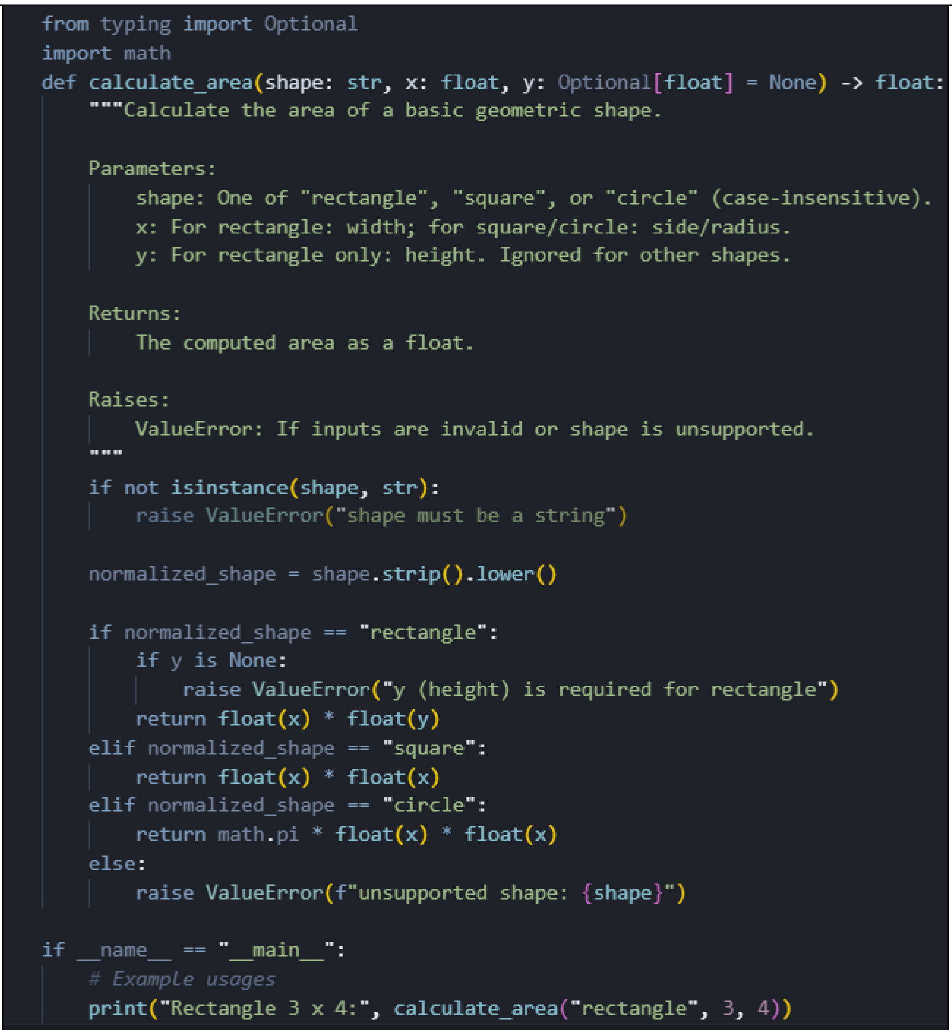
Python Code

```
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
```

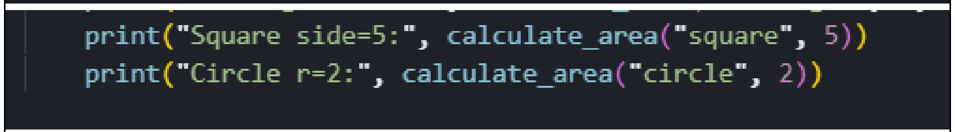
Expected Output

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

CODE:

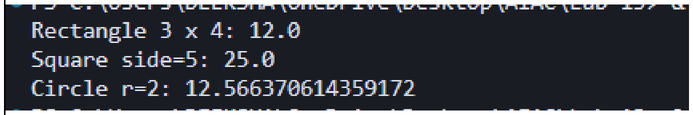


```python
from typing import Optional
import math
def calculate_area(shape: str, x: float, y: Optional[float] = None) -> float:
    """Calculate the area of a basic geometric shape.

    Parameters:
        shape: One of "rectangle", "square", or "circle" (case-insensitive).
        x: For rectangle: width; for square/circle: side/radius.
        y: For rectangle only: height. Ignored for other shapes.

    Returns:
        The computed area as a float.

    Raises:
        ValueError: If inputs are invalid or shape is unsupported.
    """
    if not isinstance(shape, str):
        raise ValueError("shape must be a string")

    normalized_shape = shape.strip().lower()

    if normalized_shape == "rectangle":
        if y is None:
            raise ValueError("y (height) is required for rectangle")
        return float(x) * float(y)
    elif normalized_shape == "square":
        return float(x) * float(x)
    elif normalized_shape == "circle":
        return math.pi * float(x) * float(x)
    else:
        raise ValueError(f"unsupported shape: {shape}")

if __name__ == "__main__":
    # Example usages
    print("Rectangle 3 x 4:", calculate_area("rectangle", 3, 4))
```



```python
    print("Square side=5:", calculate_area("square", 5))
    print("Circle r=2:", calculate_area("circle", 2))
```

OUTPUT:



```
Rectangle 3 x 4: 12.0
Square side=5: 25.0
Circle r=2: 12.566370614359172
```

Task Description #2 – Error Handling in Legacy Code

Task: Legacy function without proper error handling

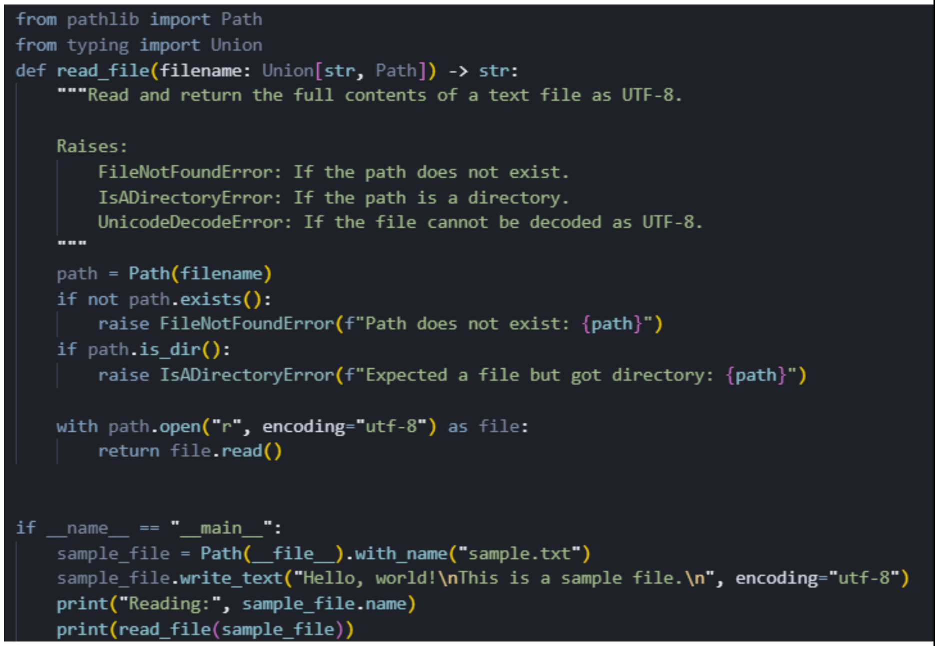
Python Code

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

Expected Output:

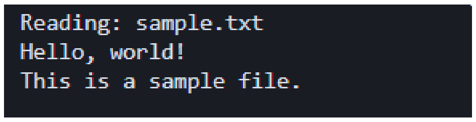
AI refactors with with open() and try-except:

CODE:



```python
from pathlib import Path
from typing import Union
def read_file(filename: Union[str, Path]) -> str:
    """Read and return the full contents of a text file as UTF-8.

    Raises:
        FileNotFoundError: If the path does not exist.
        IsADirectoryError: If the path is a directory.
        UnicodeDecodeError: If the file cannot be decoded as UTF-8.
    """
    path = Path(filename)
    if not path.exists():
        raise FileNotFoundError(f"Path does not exist: {path}")
    if path.is_dir():
        raise IsADirectoryError(f"Expected a file but got directory: {path}")

    with path.open("r", encoding="utf-8") as file:
        return file.read()


if __name__ == "__main__":
    sample_file = Path(__file__).with_name("sample.txt")
    sample_file.write_text("Hello, world!\nThis is a sample file.\n", encoding="utf-8")
    print("Reading:", sample_file.name)
    print(read_file(sample_file))
```

OUTPUT:



```
Reading: sample.txt
Hello, world!
This is a sample file.
```

Task Description #3 – Complex Refactoring

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

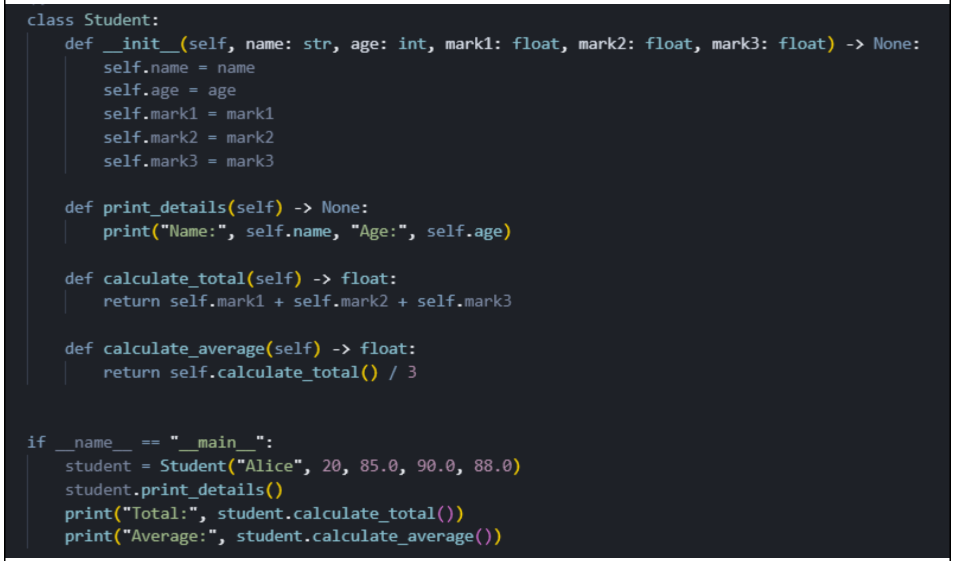
Python Code

```
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:

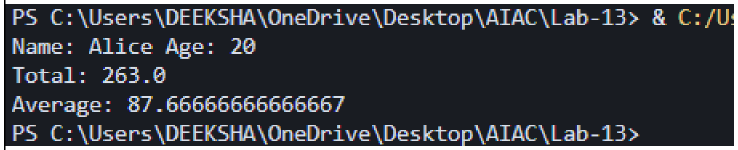
- AI improves naming (`name`, `age`, `marks`).
- Adds docstrings.
- Improves print readability.
- Possibly uses `sum(self.marks)` if marks stored in a list.

CODE:



```python
class Student:
    def __init__(self, name: str, age: int, mark1: float, mark2: float, mark3: float) -> None:
        self.name = name
        self.age = age
        self.mark1 = mark1
        self.mark2 = mark2
        self.mark3 = mark3

    def print_details(self) -> None:
        print("Name:", self.name, "Age:", self.age)

    def calculate_total(self) -> float:
        return self.mark1 + self.mark2 + self.mark3

    def calculate_average(self) -> float:
        return self.calculate_total() / 3


if __name__ == "__main__":
    student = Student("Alice", 20, 85.0, 90.0, 88.0)
    student.print_details()
    print("Total:", student.calculate_total())
    print("Average:", student.calculate_average())
```

OUTPUT:



```
PS C:\Users\DEEKSHA\OneDrive\Desktop\AIAC\Lab-13> & C:/U
Name: Alice Age: 20
Total: 263.0
Average: 87.66666666666667
PS C:\Users\DEEKSHA\OneDrive\Desktop\AIAC\Lab-13>
```

Task Description #4 – Inefficient Loop Refactoring

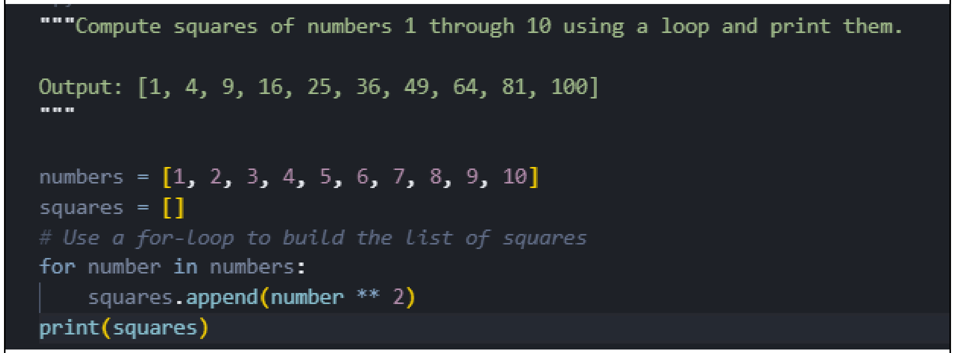
Task: Refactor this inefficient loop with AI help

Python Code

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

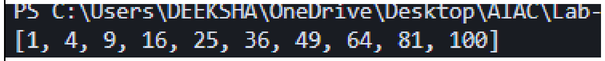
Expected Output: AI suggested a **list comprehension**

CODE:



```
"""Compute squares of numbers 1 through 10 using a loop and print them.

Output: [1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
"""

numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
squares = []
# Use a for-loop to build the list of squares
for number in numbers:
    squares.append(number ** 2)
print(squares)
```

OUTPUT:



```
PS C:\Users\DEEKSHA\OneDrive\Desktop\AIAC\Lab-
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```