AI ASSISTED CODING

LAB-13.2

NAME:M.SINDHUJA

ENROLLNO:2403A52060

BATCH:03

**TASK-01:**

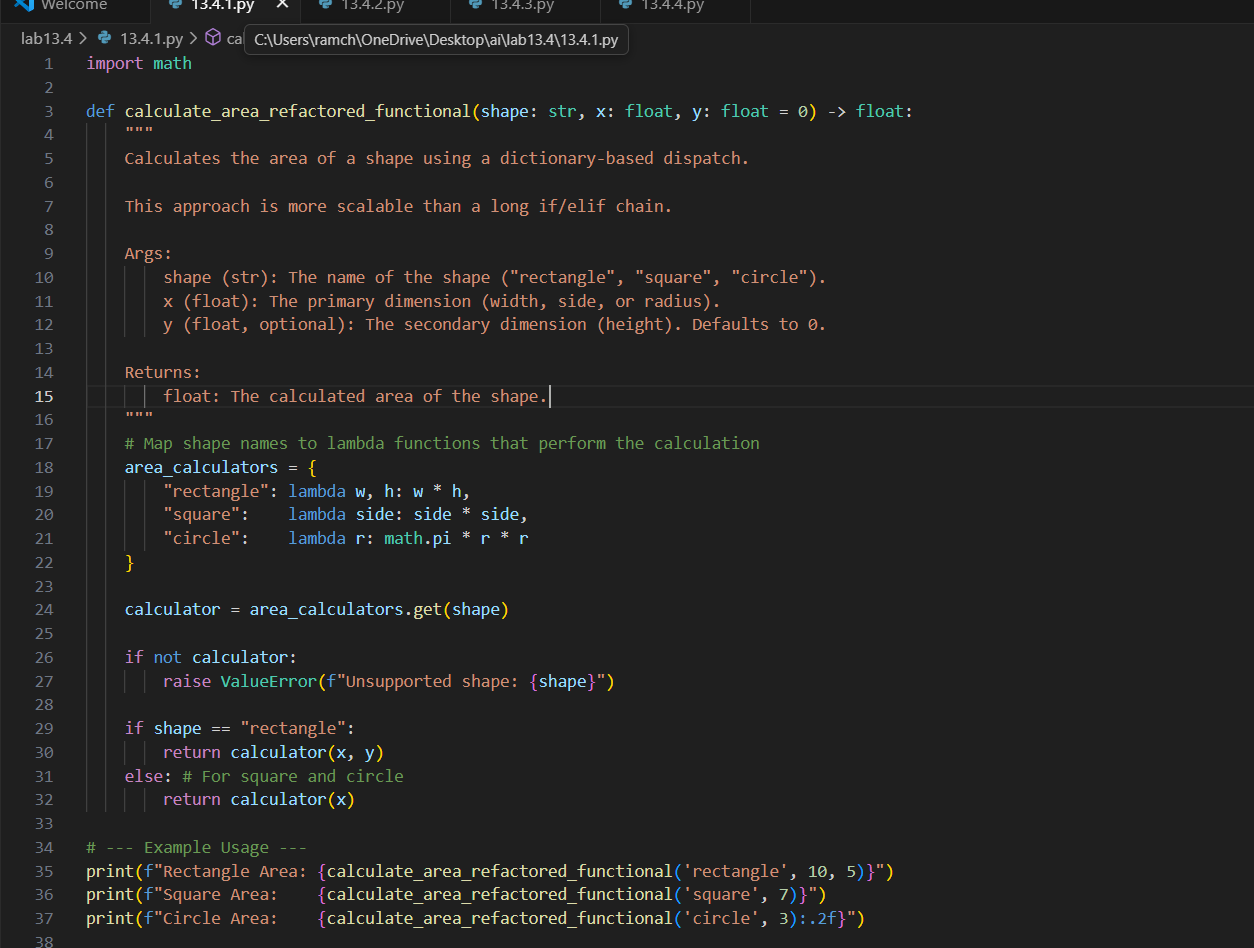
Remove Repetition.

**PROMPT:**

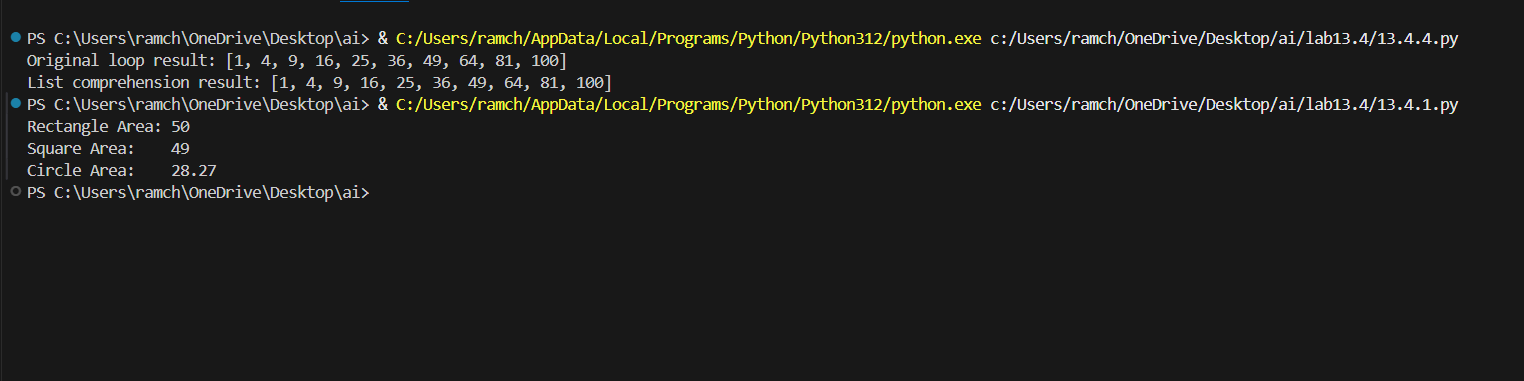
Refactor the following redundant code

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

**CODE:**

****

**OUTPUT:**

****

**OBSERVATION:**

The function calculate\_area computes the area of a rectangle, square, or circle based on the given dimensions. It converts the shape name to lowercase for consistency, uses x as the main dimension and y as an optional width for rectangles, and calculates the area accordingly. For rectangles, both x and y are required, squares use x\*\*2, and circles use math.pi \* x\*\*2. It raises an error if the shape is unknown or if required dimensions are missing. The docstring explains its usage and parameters.

**TASK-02:**

Error Handling in Legacy Code.

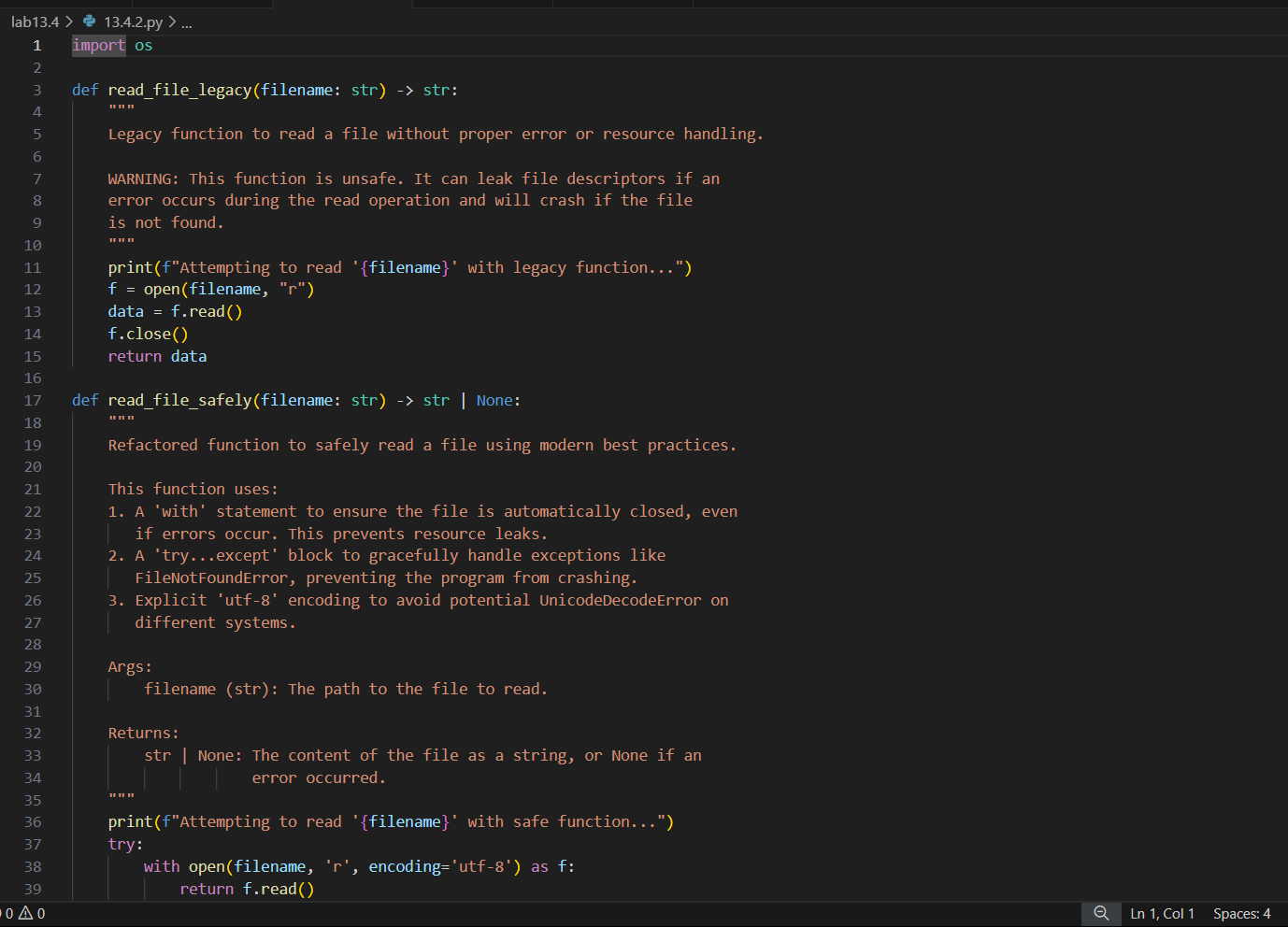
**PROMPT:**

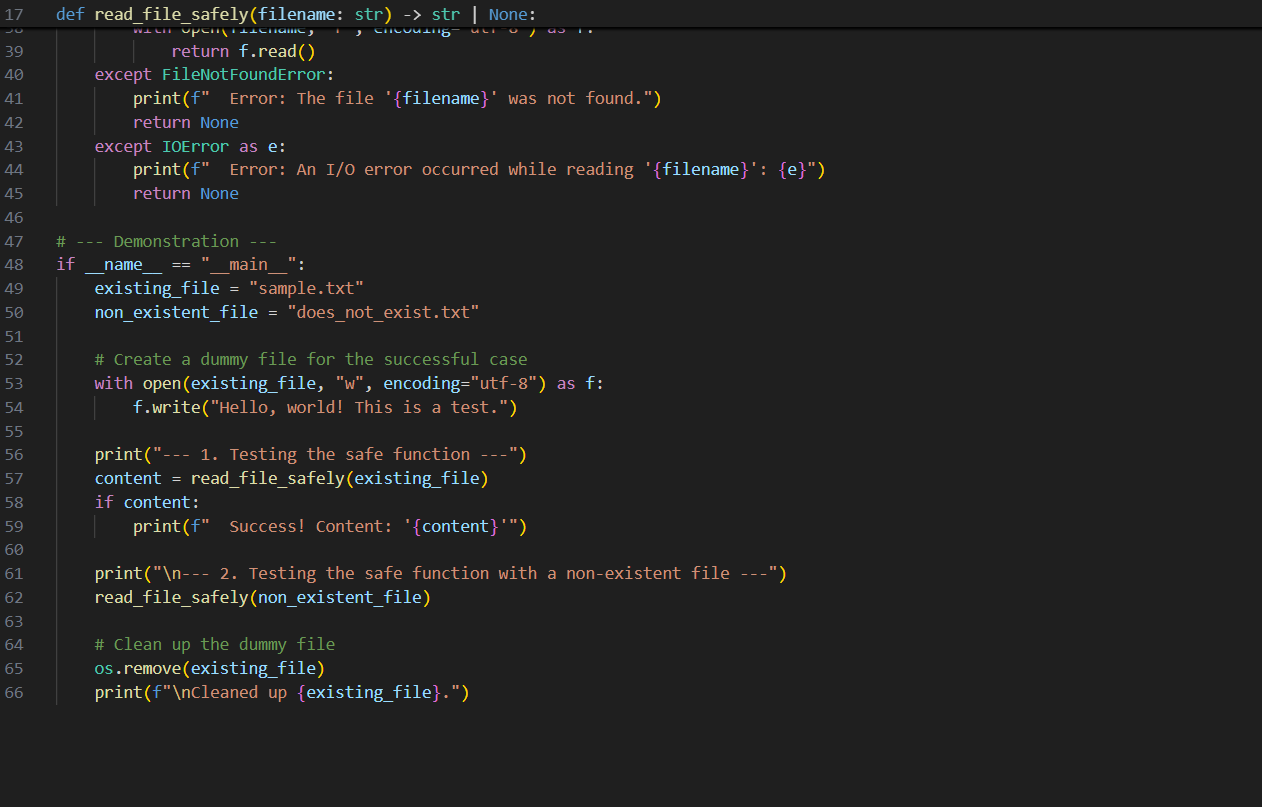
The following python code that reads the file but it doesn’t handle the errors . rewrite the code by correcting all the errors.

def read\_file(filename):  
f = open(filename, "r")  
data = f.read()

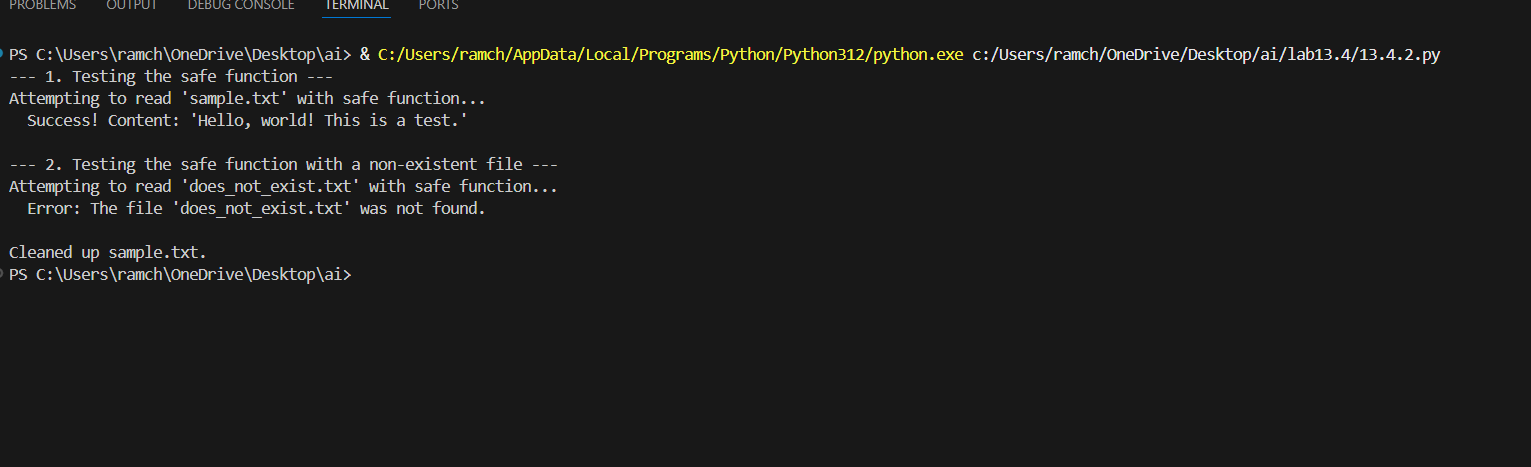
f.close()  
return data

**CODE:**

****

****

**OUTPUT:**

****

**OBSERVATION:**

The refactored function safely reads a file using with open(), ensuring the file is automatically closed, and uses try-except to handle errors like missing files or read failures. It provides clear error messages instead of crashing, making the code more robust and reliable.

**TASK-03:**

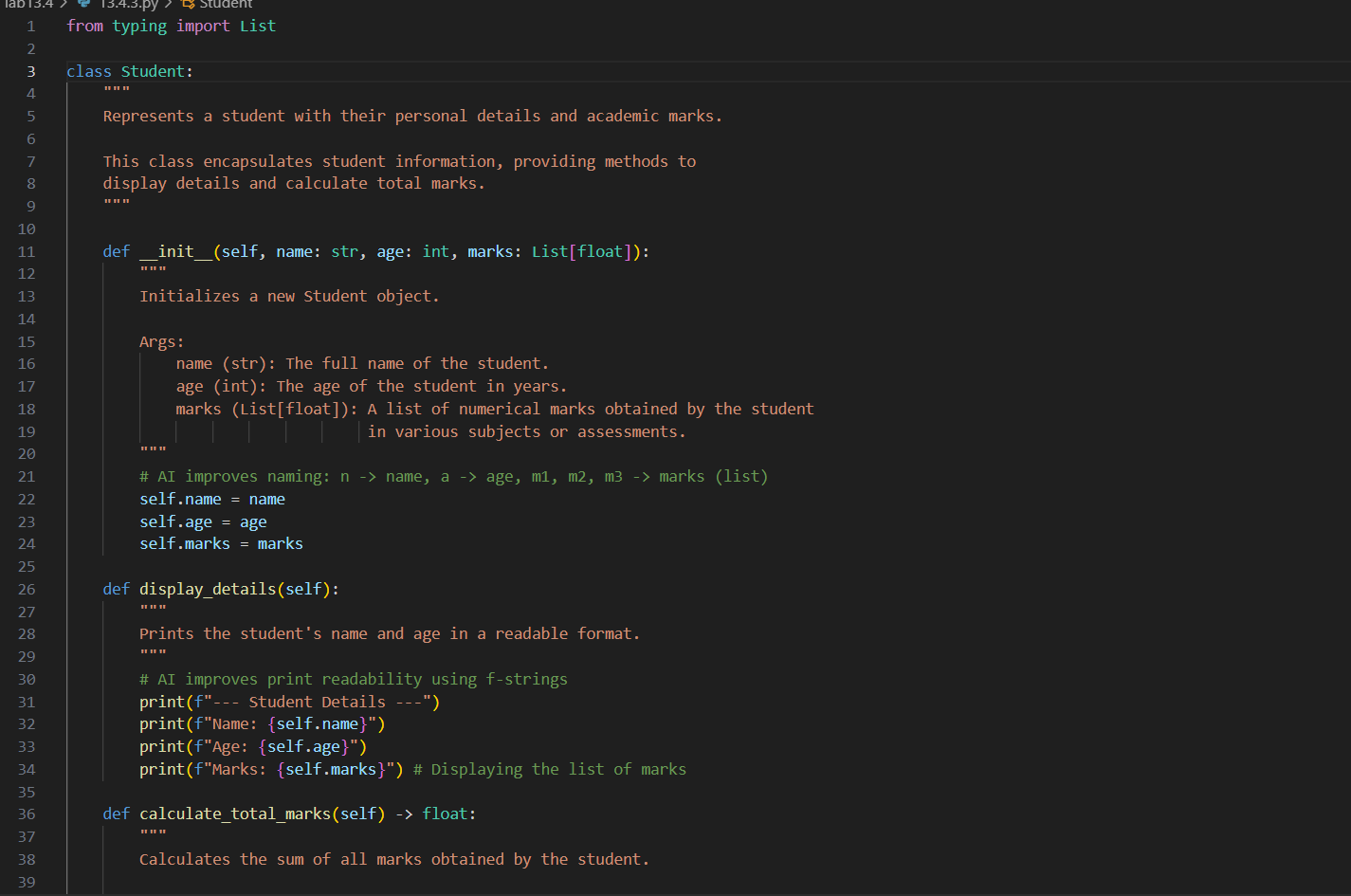
Complex refactoring

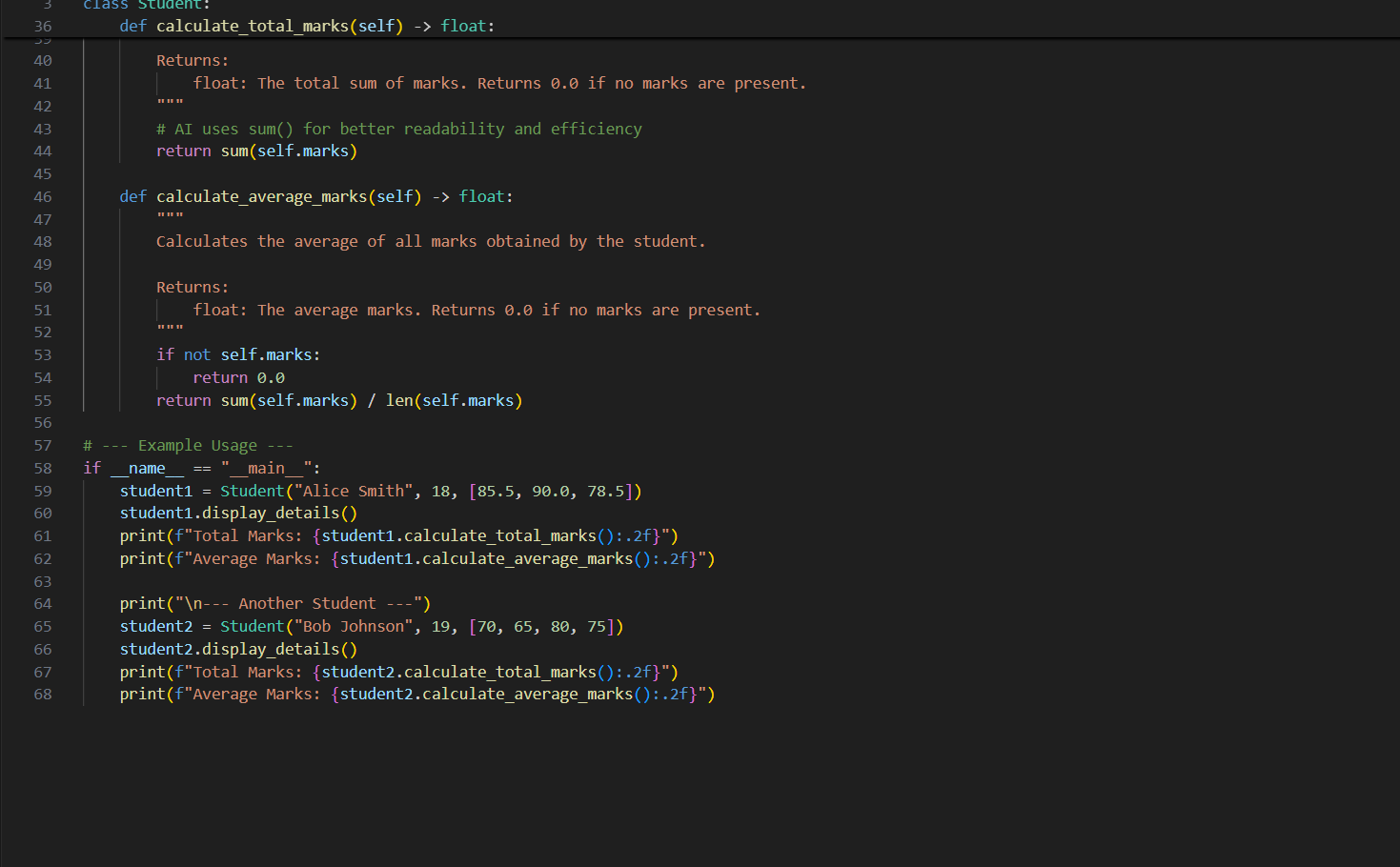
**PROMPT:**

Rewrite the following code by adding the proper variable names and refactor it in a proper way.

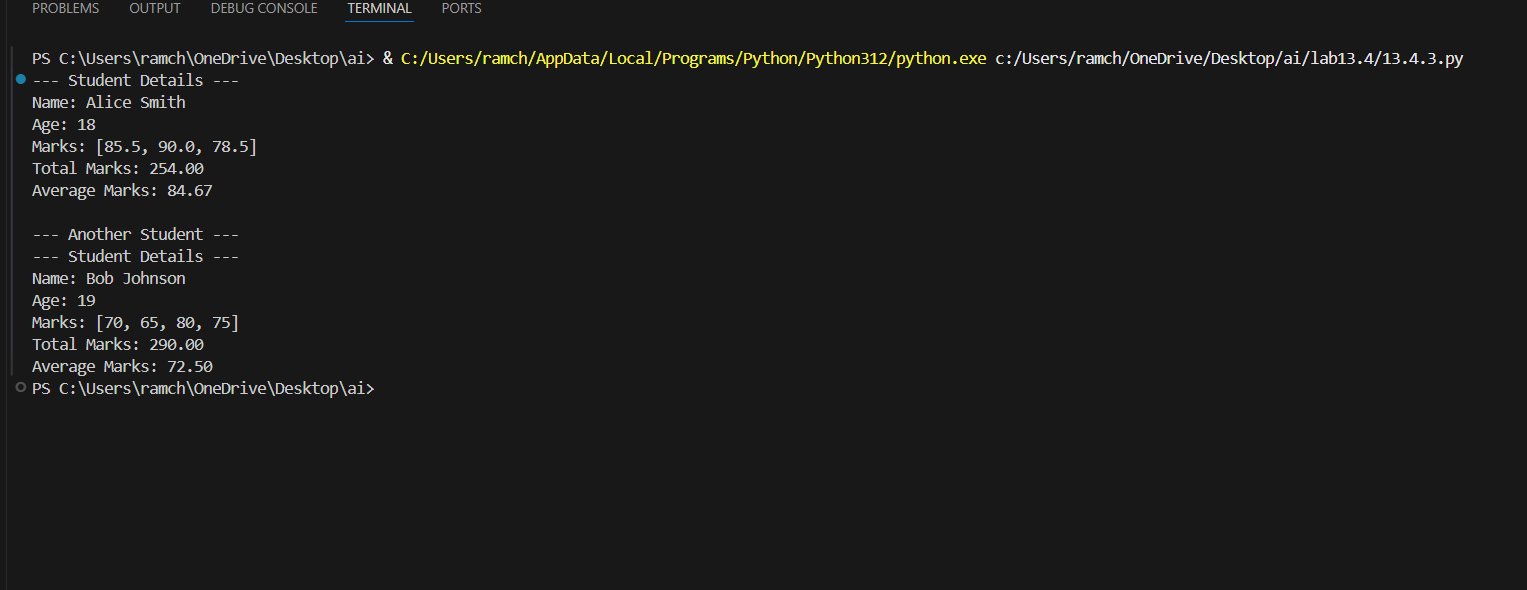
class Student:  
def \_\_init\_\_(self, n, a, m1, m2, m3):  
self.n = n  
self.a = a  
self.m1 = m1  
self.m2 = m2  
self.m3 = m3  
def details(self):  
print("Name:", self.n, "Age:", self.a)  
def total(self):  
return self.m1+self.m2+self.m3

**CODE:**

****

****

**OUTPUT:**

****

**OBSERVATION:**

The refactored Student class improves readability and modularity by using meaningful names, storing marks in a list, and adding docstrings. The details method prints information clearly with formatted strings, and total efficiently sums marks using sum(). The design is now more flexible and easier to extend.

**TASK-04:**

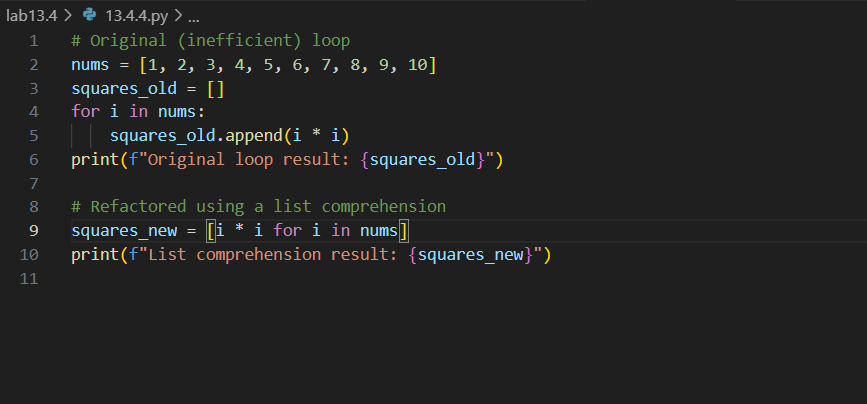
Inefficient Loop Refactoring

**PROMPT:**

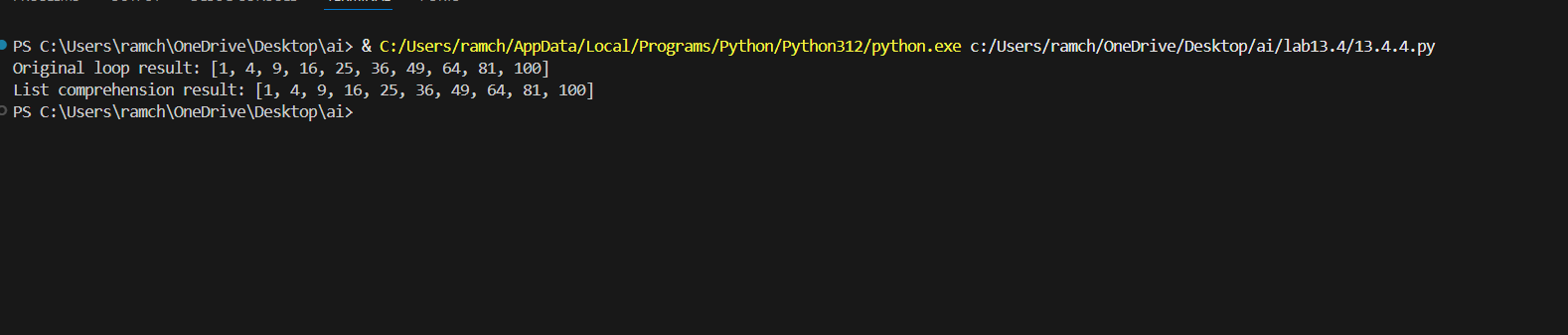
I have a Python loop that computes squares of numbers and appends them to a list, but it seems inefficient. Can you rewrite it in a shorter, more Pythonic way and explain why it’s better?

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

**CODE:**

****

**OUTPUT:**

****

**OBSERVATION:**

The list comprehension makes the code shorter, more readable, and efficient by replacing the explicit loop and append() method with a single expression.