**AI ASSISTED CODING LAB ASSIGNMENT 13.2**

**ENROLLMENT NO :**2503A51L21

**BATCH NO:** 19

**NAME:** LUNGHIMPOU KAMEI

**TASK1: 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

**PROMPT: Legacy Code-**

def calculate\_area(shape, x, y=0):

if shape == "rectangle":

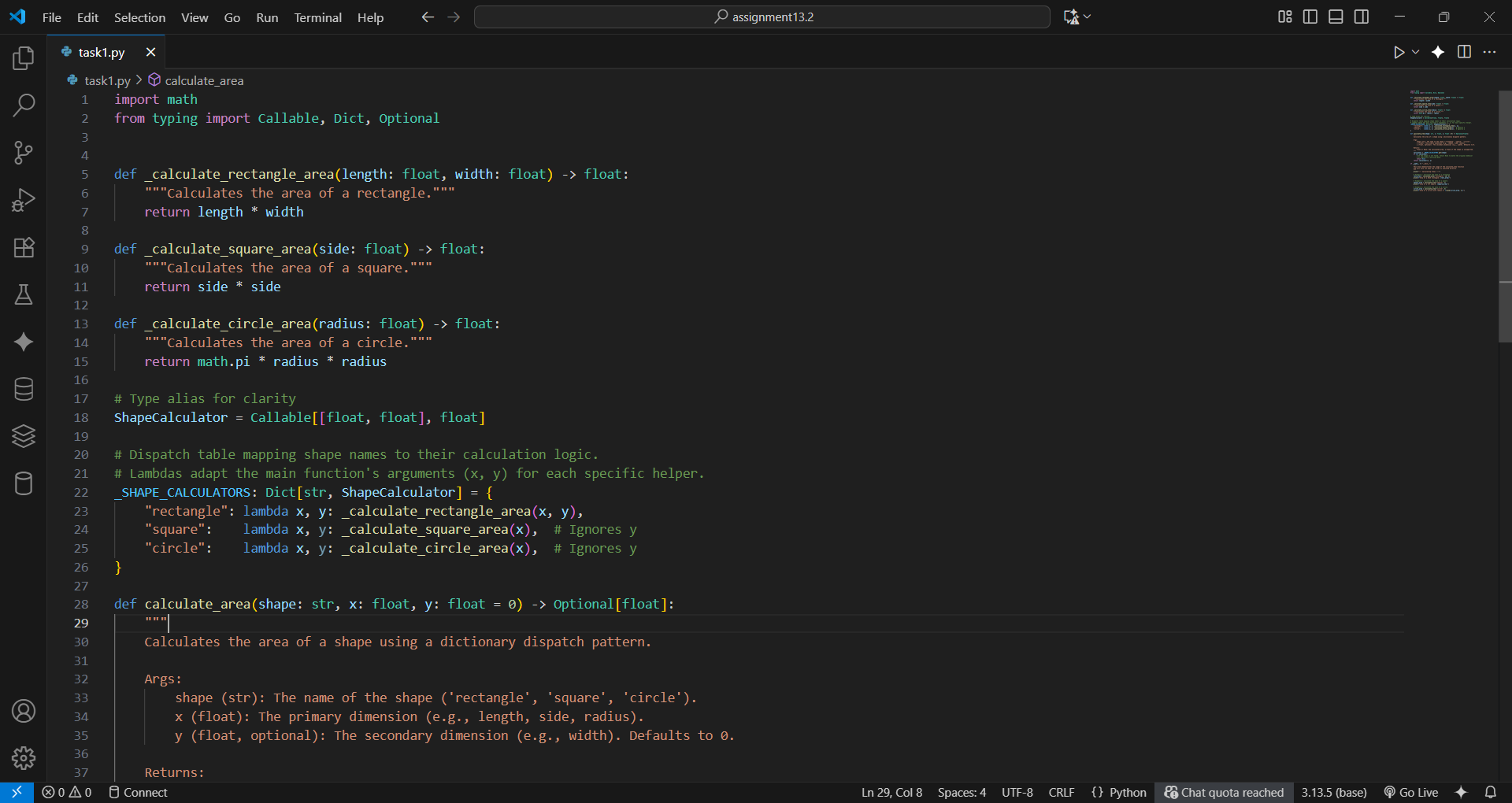
return x \* y

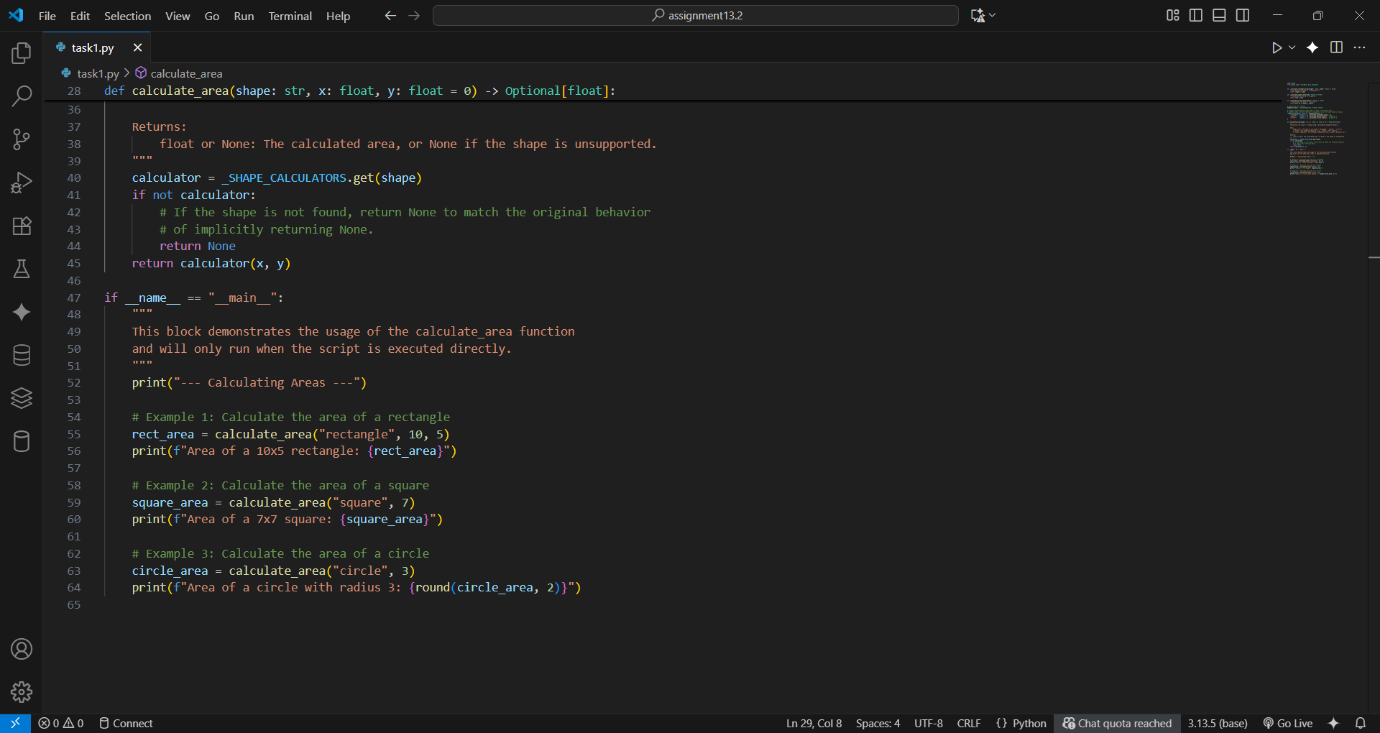
elif shape == "square":

return x \* x

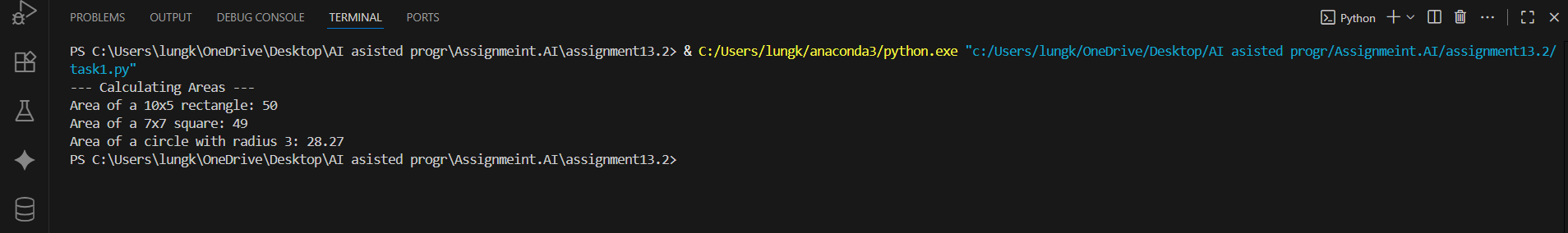
elif shape == "circle":

return 3.14 \* x \* x  
Generate a python code to refractor the following function to removed repetition, used dictionary-based dispatch or separate functions.

****

****

**OUTPUT:**



**OBSERVATION:**

Clean Dispatch Pattern: It replaces a rigid if/elif chain with a flexible dictionary dispatch, cleanly mapping shape names to their specific calculation functions.

Modular and Extensible: The code is highly modular, with each shape's logic isolated in its own function. This makes it easy to maintain and extend with new shapes without modifying existing logic.

**TASK2**: **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

**PROMPT: Legacy Code-**

def read\_file(filename):

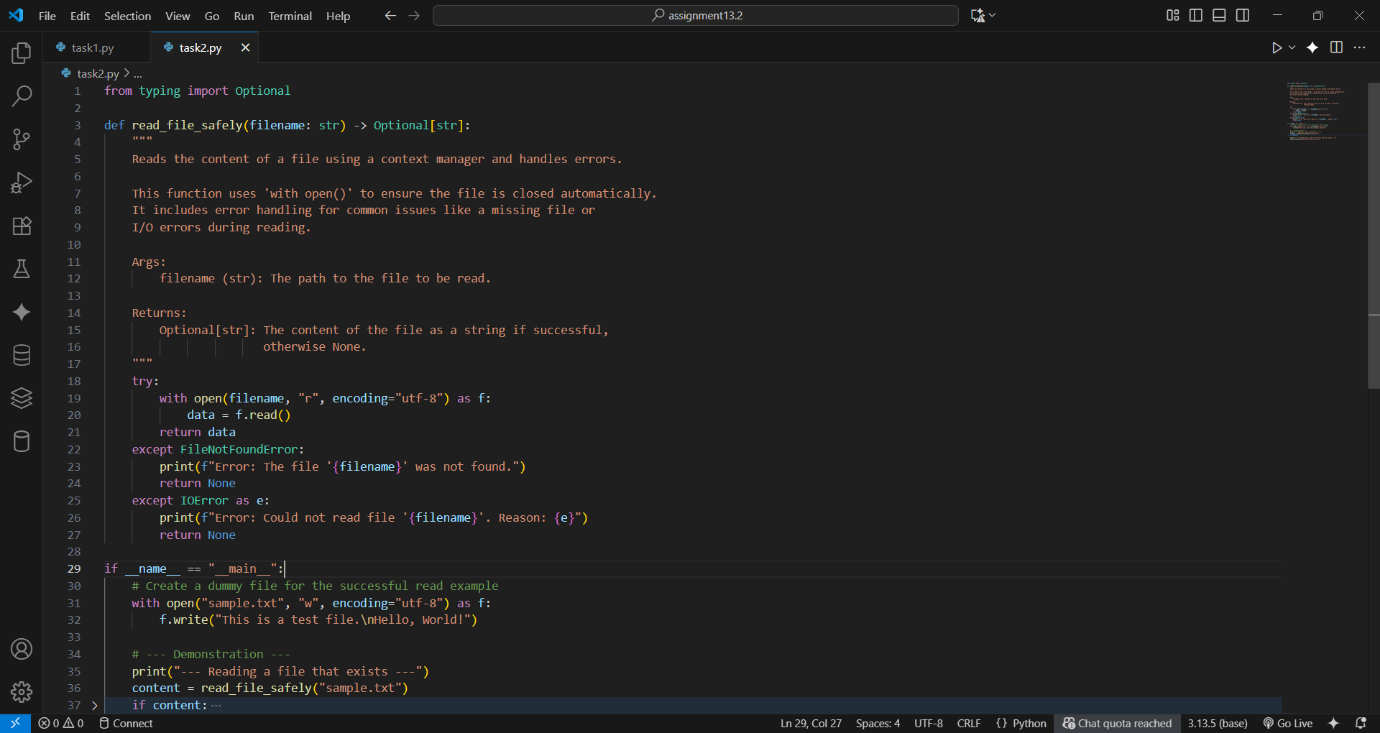
f = open(filename, "r")

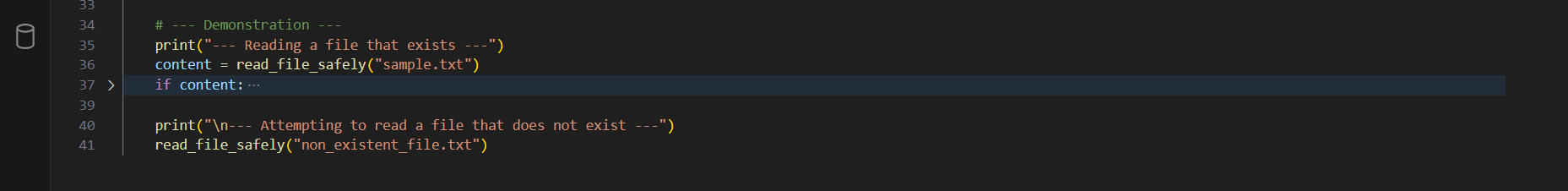
data = f.read()

f.close()

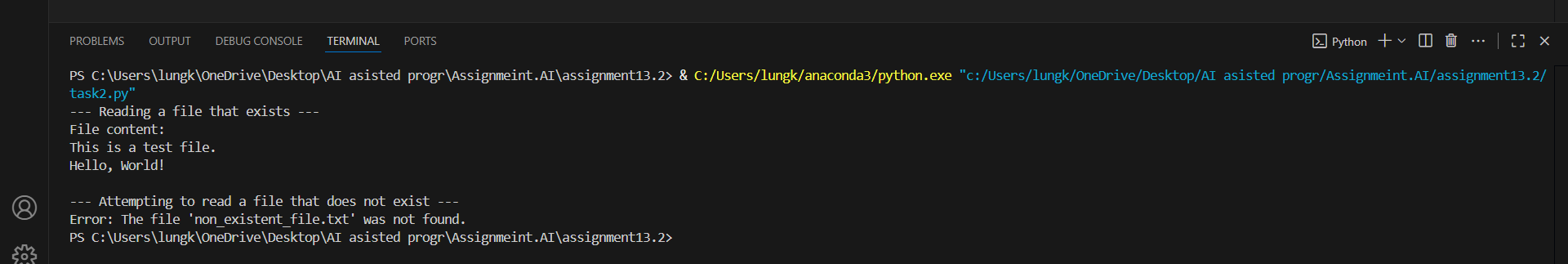
return data

Write a python code to improve this function by adding error handling and using with open() context manager and try-except.





**OUTPUT:**

****

**OBSERVATION:**

Resource-Safe: The function correctly uses with open (), which guarantees the file is always closed automatically, preventing resource leaks.

Robust Error Handling: It gracefully handles common errors like a missing file (FileNotFoundError), making the function resilient and preventing program crashes

**TASK3: 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

**PROMPT: Legacy Code-**

class Student:

def \_\_init\_\_(self, n, a, m1, m2, m3):

self.n = n

self.a = a

self.m1 = m1

self.m2 = m2

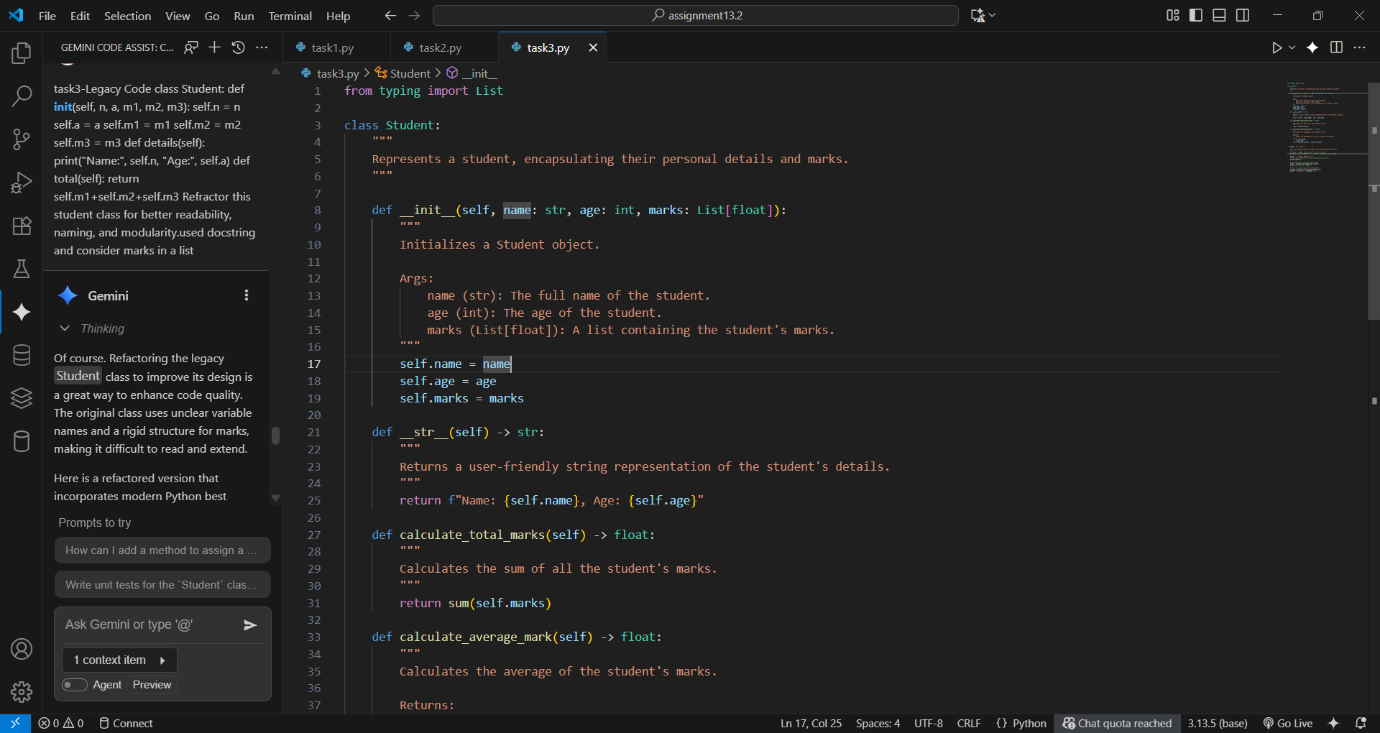
self.m3 = m3

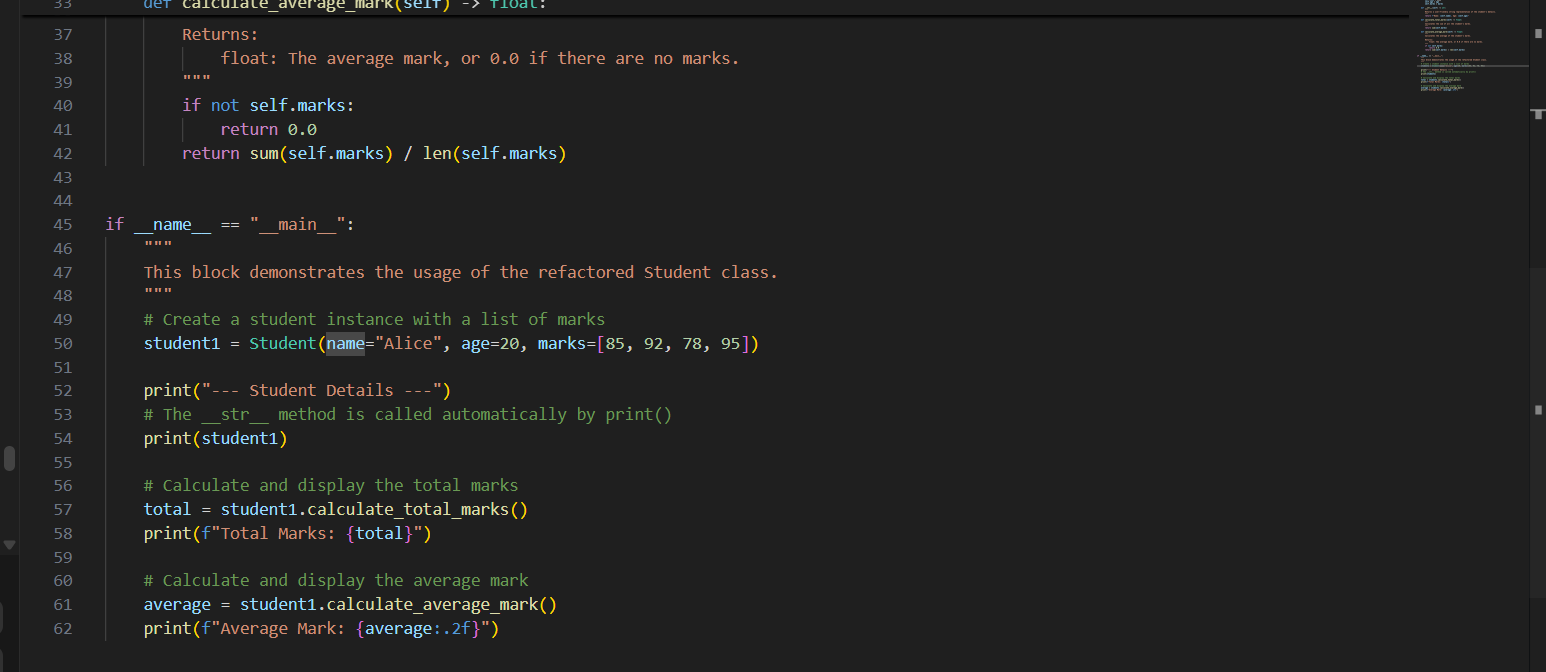
def details(self):

print("Name:", self.n, "Age:", self.a)

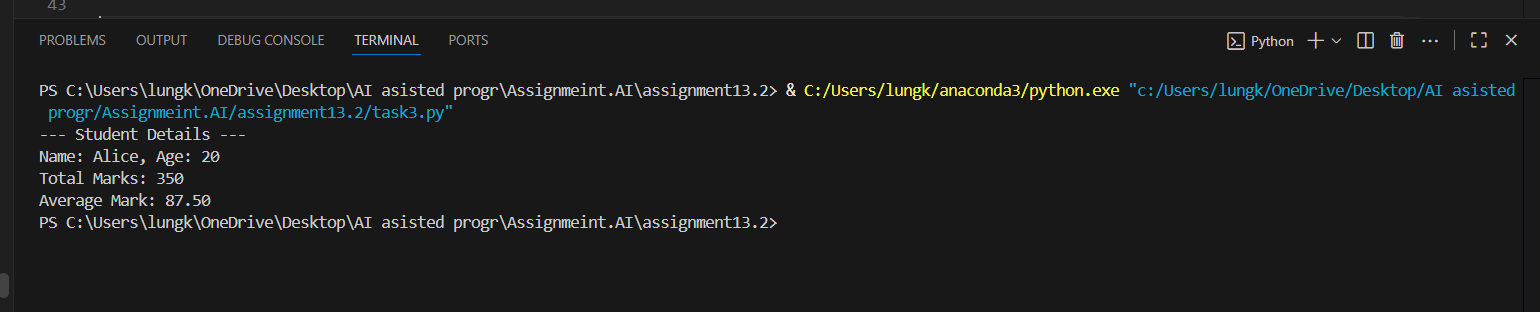
def total(self):

return self.m1+self.m2+self.m3  
Generate a code to refractor this student class for better readability, naming, and modularity. Used docstring and consider marks in a list

****

****

**OUTPUT:**

****

**OBSERVATION:**

Improved Readability: The class is significantly more readable than its legacy version, using clear names for attributes (name, age) and methods (calculate\_total\_marks), along with helpful docstrings and type hints.

Flexible and Pythonic Design: It replaces rigid, fixed attributes for marks with a flexible list, making the class more scalable. It also uses the Pythonic \_\_str\_\_ method for a clean, user-friendly string representation.

**TASK4**: **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)

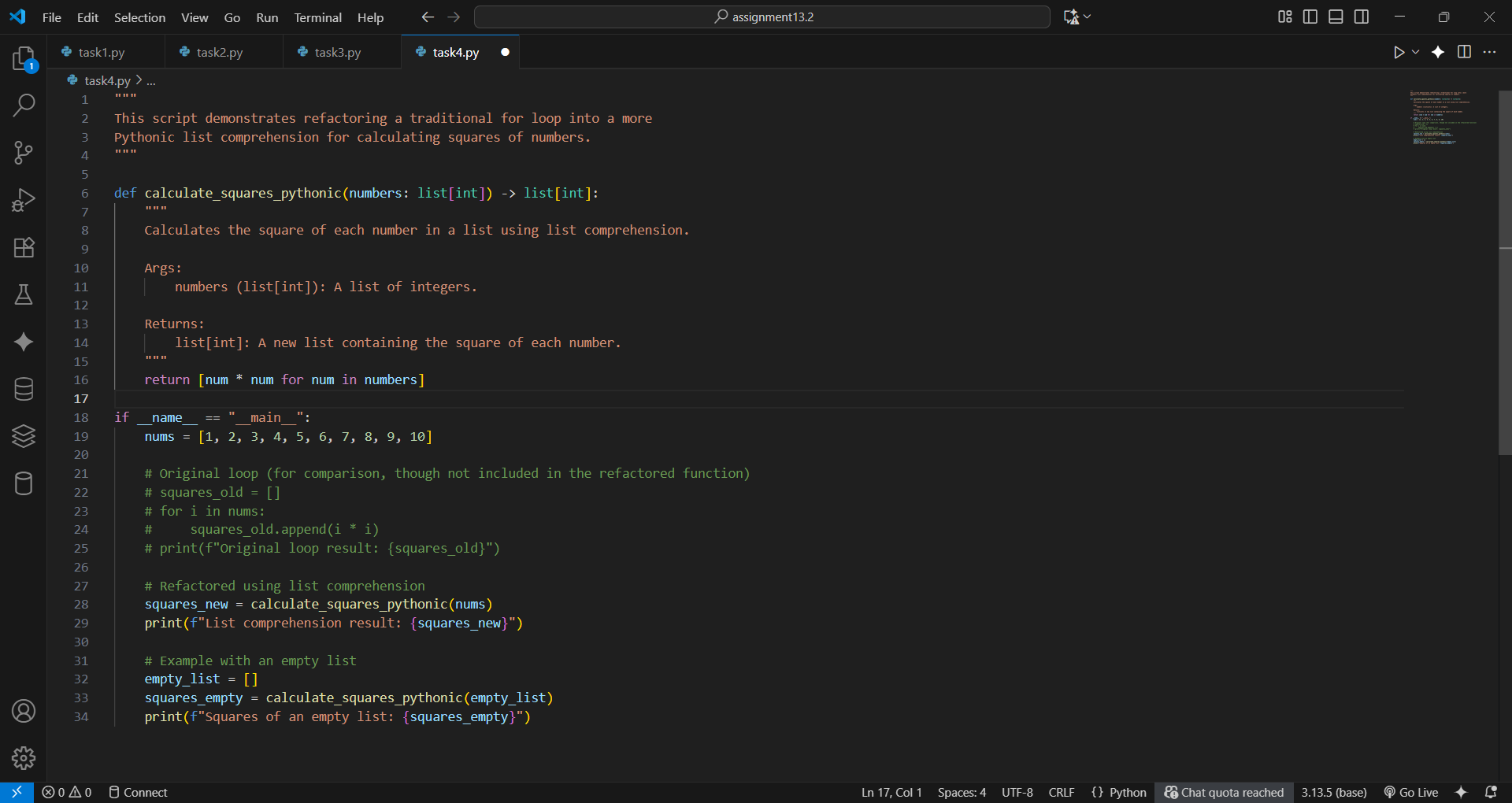
**PROMPT: Legacy Code**

nums = [1,2,3,4,5,6,7,8,9,10]

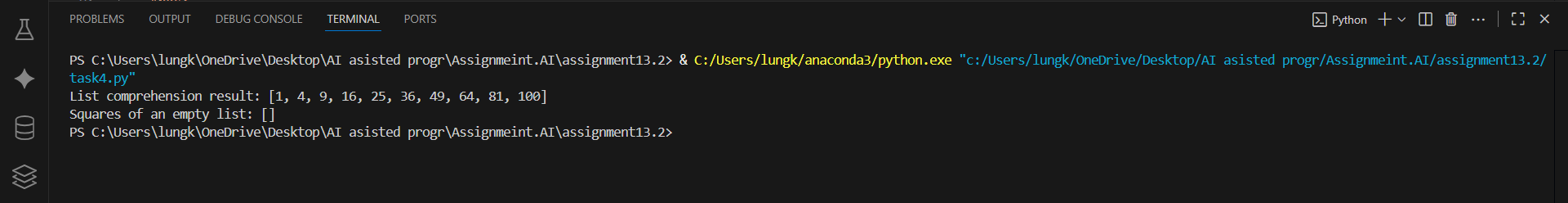
squares = []

for i in nums:

squares.append(i \* i)  
Write a code to Refactor this loop into a more efficient solution using list comprehension.

****

**OUTPUT:**

****

**OBSERVATION:**

**.** The list comprehension [num \* num for num in numbers] is a single, readable line that directly describes its action: creating a new list by squaring each number.

**.**This is the standard, preferred method in Python for this task. It's not only cleaner but also often faster than a traditional for loop because it's highly optimized.