AI ASSISTED CODING LAB

ASSIGNMENT 13.2

ENROLLMENT NO :2503A51L14

BATCH NO: 19

NAME: ROHITH GOPAGANI

TASK DESCRIPTION 1: 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

elf shape == "square":

return x \* x

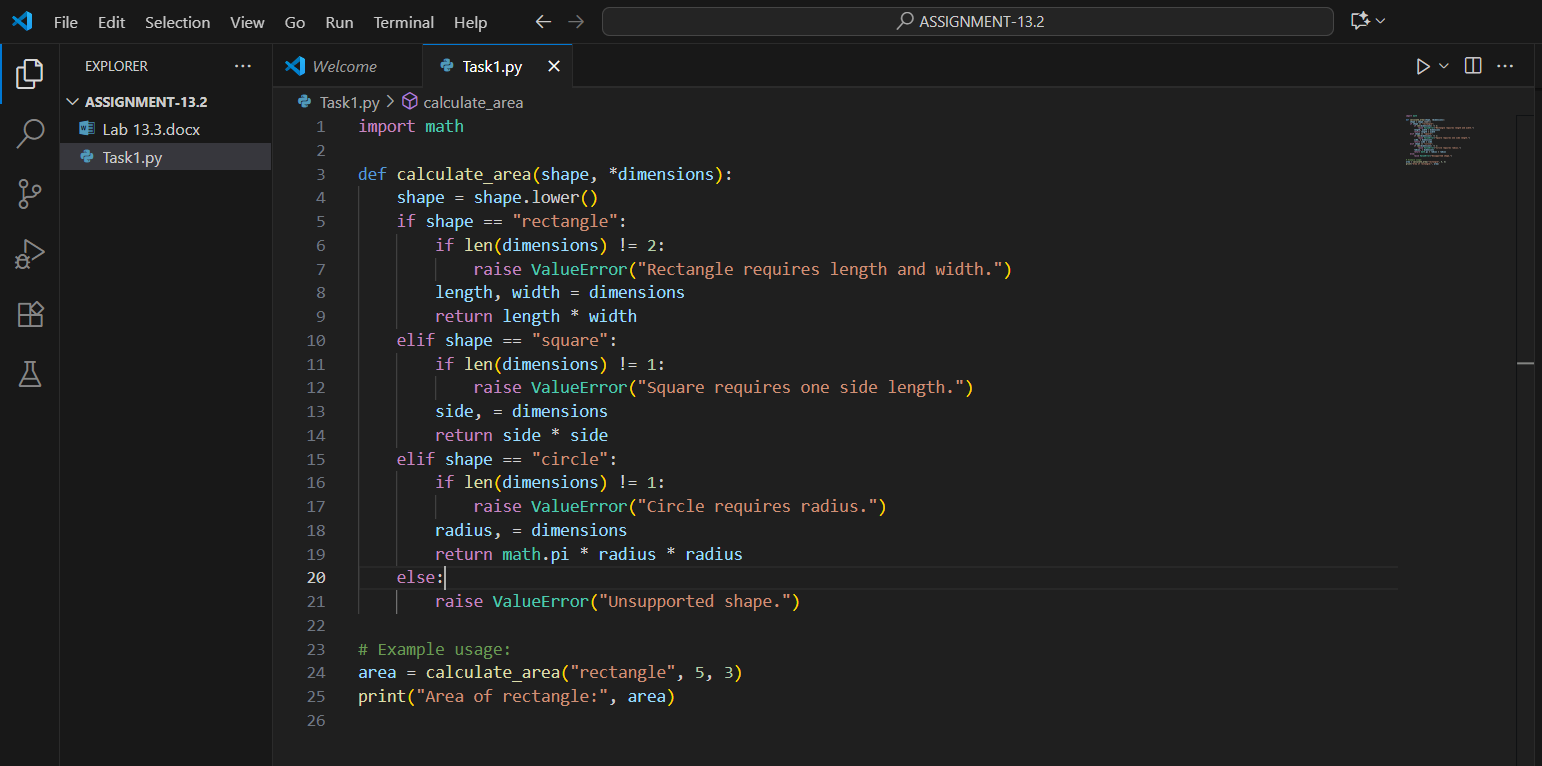
elif shape == "circle":

return 3.14 \* x \* x

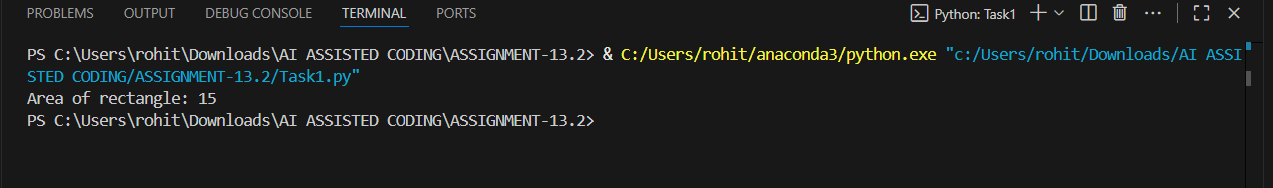
PROMPT :

Python function that calculates the area of different shapes, but it has some repeated logic. Can you refactor it to make the code cleaner and more efficient?

CODE GENERATED :



OUTPUT:



OBSERVATION:

The given Python function calculate area calculates the area of a rectangle, square, or circle. However, the code contains redundant logic since both the square and circle reuse the formula x \* x, only differing by a constant factor (π for a circle). The function also relies on multiple if-elif statements, which makes it less clean and harder to extend for additional shapes in the future. This can be improved by using a dictionary-based mapping of shapes to their respective area formulas, reducing repetition and making the code more efficient and scalable.

TASK DESCRIPTION 2: Legacy function without proper error handling

**Python Code**

def read\_file(filename):

f = open(filename, "r")

data = f.read()

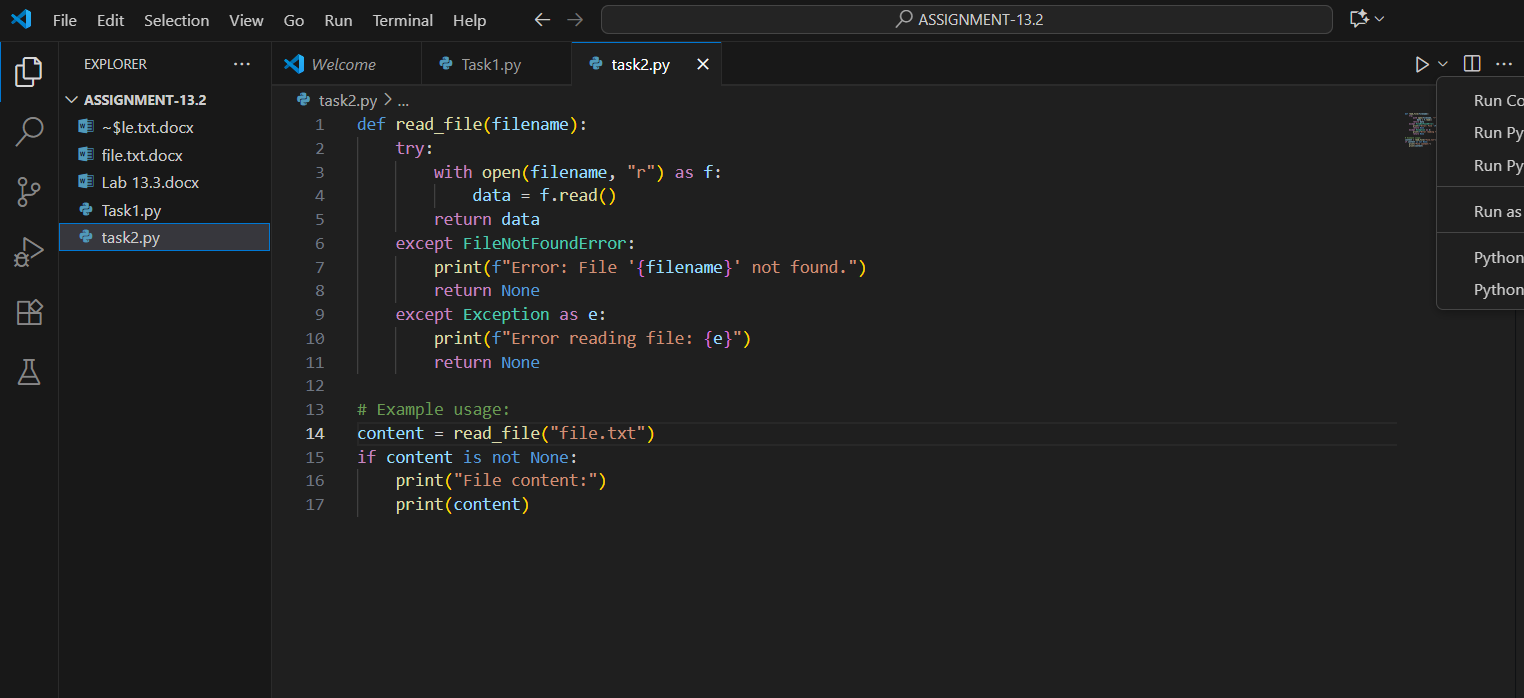
f.close()

return data

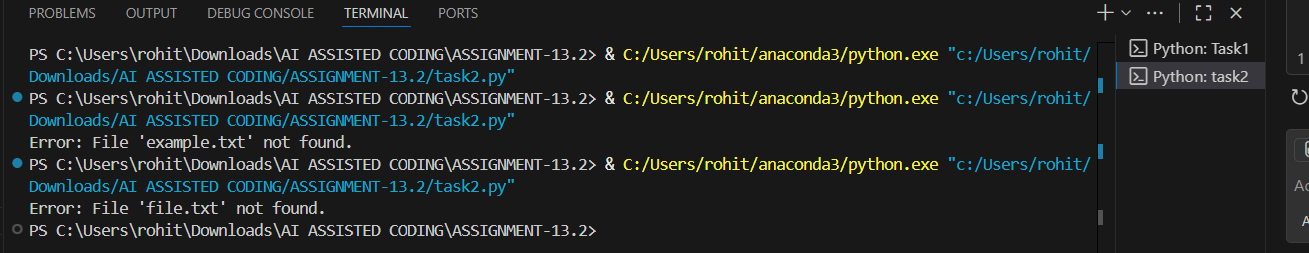
PROMPT:

I have a legacy Python function that reads the contents of a file, but it doesn’t have proper error handling or resource management. Can you refactor this code to make it safer and more efficient, ensuring that the file is always closed properly and errors like missing files are handled gracefully?

CODE GENERATED:



OUTPUT:



OBSERVATION:

The given legacy function read\_file is able to read data from a file, but it does not handle errors or manage resources safely. If the specified file does not exist, or if an error occurs during reading, the function will raise an exception and may not close the file properly. The use of manual open() and close() also makes the code less reliable. A better approach is to use Python’s context manager (with open(...) as f:), which ensures that the file is automatically closed even if an error occurs. Additionally, implementing try-except blocks will make the function more robust by gracefully handling common errors like FileNotFoundError or IOError.

TASK DESCRIPTION 3: 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)

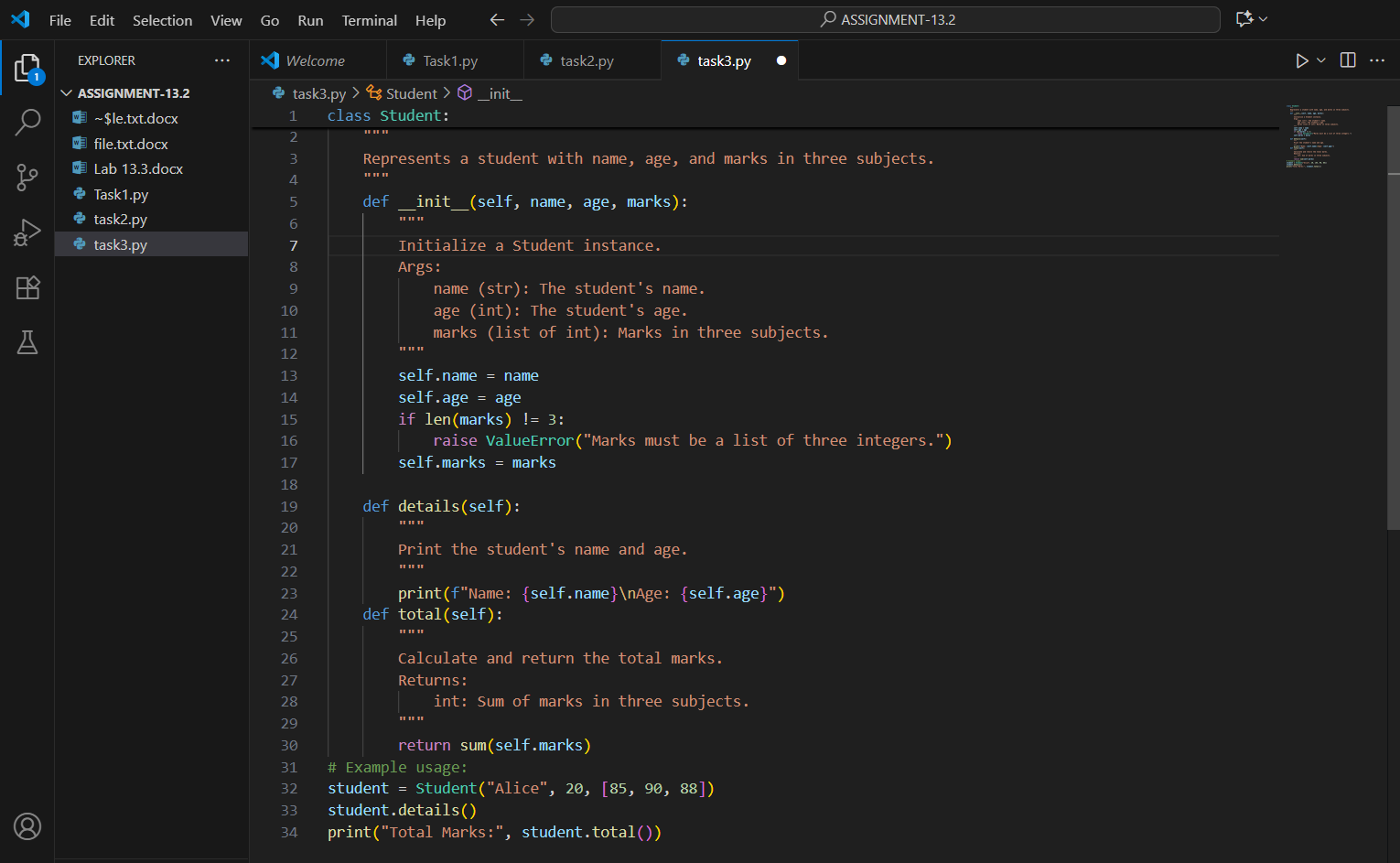
def total(self):

return self.m1+self.m2+self.m3

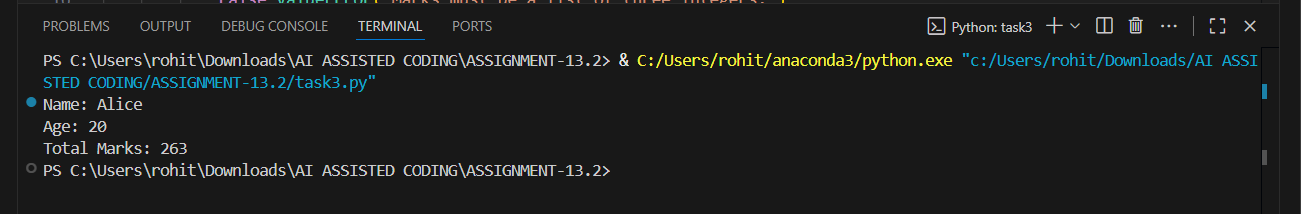
PROMPT :

I have a legacy Python class Student that stores a student’s name, age, and marks. However, the code lacks readability and modularity since it uses short variable names and has limited functionality. Can you refactor this class to improve readability by using meaningful variable names, make it more modular, and enhance its structure so it follows good object-oriented programming practices?

CODE GENERATED :



OUTPUT :



OBSERVATION :

The given Student class demonstrates basic functionality for storing a student’s name, age, and marks, along with methods to display details and calculate the total marks. However, the code lacks readability and modularity. The use of short variable names (n, a, m1, m2, m3) makes the class harder to understand and maintain. The methods provide minimal functionality, and the structure does not fully utilize object-oriented programming best practices such as encapsulation, clear naming conventions, or potential extensibility (e.g., handling more subjects, calculating averages, or storing data in a structured way). Refactoring is needed to improve readability, use meaningful variable names, and make the class more modular and scalable.

TASL DESCRIPTION 4 :

Refactor this inefficient loop with AI help

**Python Code**

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

squares = []

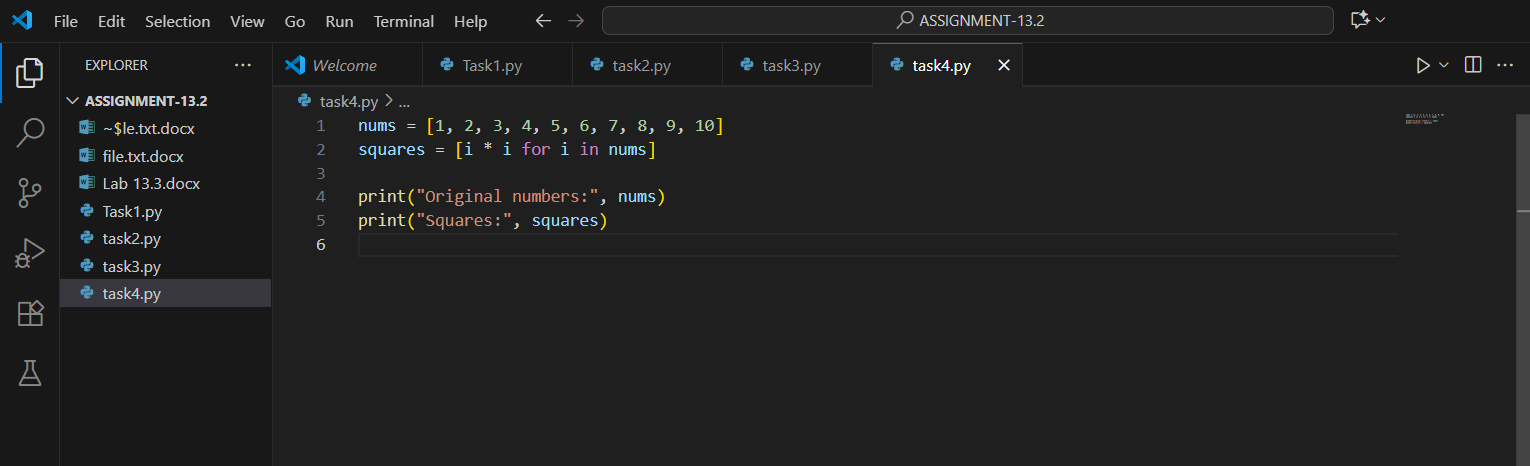
for i in nums:

squares.append(i \* i)

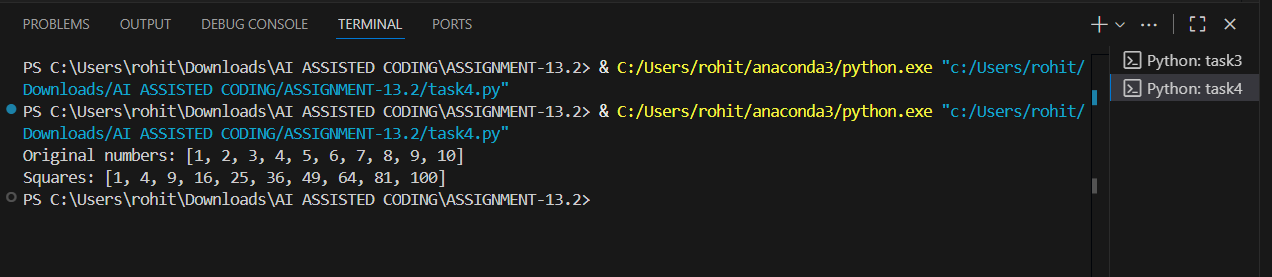
PROMPT :

I have a Python loop that generates the squares of numbers in a list, but the implementation is not very efficient. Can you help me refactor this code to make it cleaner and more Pythonic while achieving the same result?

CODE GENERATED :



OUTPUT :



OBSERVATION :

The given code correctly generates the squares of numbers in a list, but the approach is not efficient or Pythonic. It uses a manual loop along with the append() method, which makes the code longer and less readable. In Python, list comprehensions or functional approaches (like map) are more concise and efficient for such tasks. Refactoring this loop with a list comprehension will make the code shorter, cleaner, and easier to understand while producing the same result.