**AI ASSIGNMENT-13**

NAME OF THE LAB: AI ASSISTED CODING

NAME OF THE STUDENT: Thirumalakonda shiva

ENROLLNO: 2503A51L33

BATCH: 20

TASK 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

elif shape == "square":

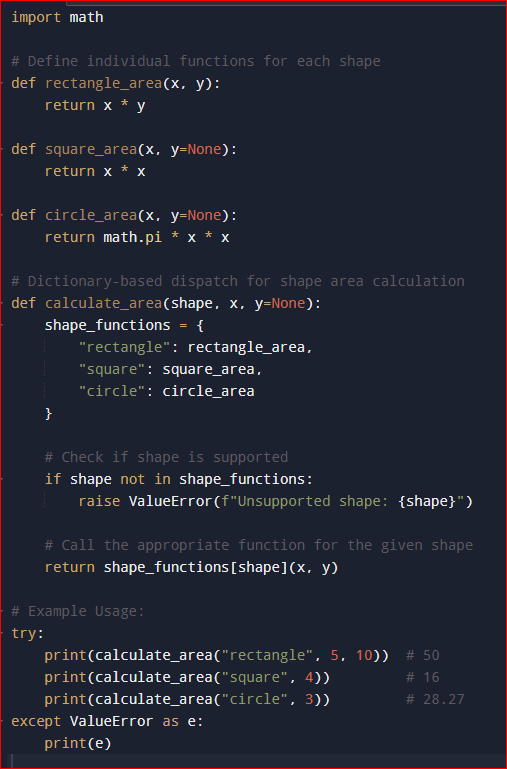
return x \* x

elif shape == "circle":

return 3.14 \* x \* x

**PROMPT:**Refactor the given Python code using dictionary-based dispatch or separate functions for each shape, aiming for cleaner, modular design and easier extensibility.

**CODE:**



**OUTPUT:**

****

**OBSERVATION:**

Your refactored code is clean and modular, with separate functions for each shape and a dictionary-based dispatch for easy extension. It includes proper error handling for unsupported shapes and uses math.pi for accuracy. One small improvement could be adding validation to ensure required parameters (like y for rectangles) are provided.

**TASK 2:**

Legacy function without proper error handling

**Python Code**

def read\_file(filename):

f = open(filename, "r")

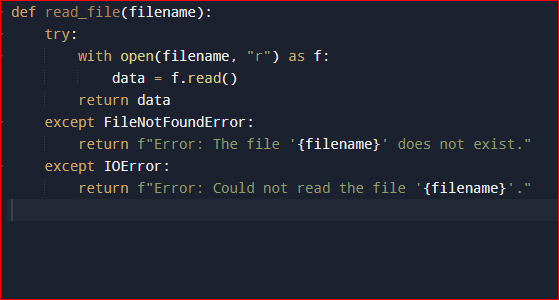
data = f.read()

f.close()

return data

**PROMPT:** Refactor the legacy read\_file function to use with open() for automatic file closing and add try-except blocks to handle file-related errors gracefully.

**CODE:**



**OBSERVATION:**

The original function lacks error handling and risks leaving the file open if an exception occurs. Using with open() ensures the file closes automatically, improving safety. Adding try-except blocks helps handle missing or unreadable files gracefully, preventing crashes.

**TASK 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)

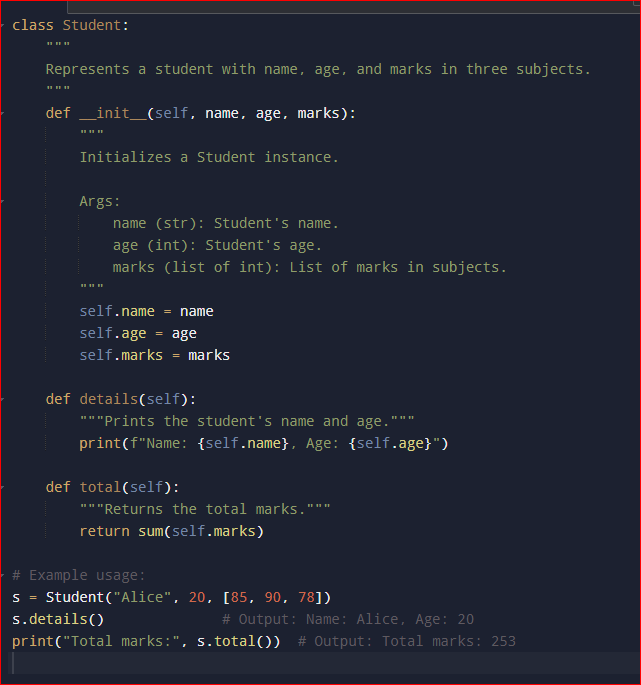
def total(self):

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

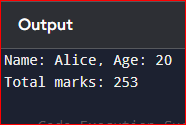
**PROMPT:**

Refactor the legacy Student class to improve readability and modularity by using meaningful variable names, adding docstrings, enhancing print formatting, and storing marks in a list for easier total calculation.

**CODE:**

****

**OUTPUT:**

****

**OBSERVATION:**

The legacy Student class uses unclear variable names and stores marks in separate attributes, making it less flexible. Its methods lack docstrings, and the print statement is basic, reducing readability. Refactoring to use descriptive names, docstrings, formatted printing, and a list for marks would improve clarity and modularity.

**TASK 4:**

Refactor this inefficient loop with AI help

**Python Code**

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

squares = []

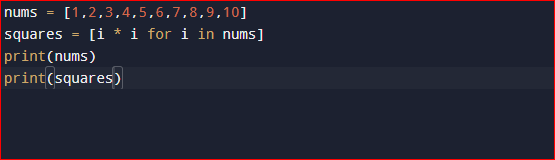
for i in nums:

squares.append(i \* i)

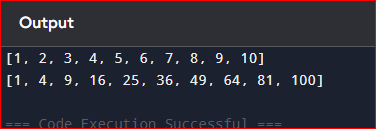
**PROMPT:**

Refactor the given loop to use a list comprehension for more efficient and concise creation of the squares list.

**CODE:**

****

**OUTPUT:**

****

**OBSERVATION:**

The original loop works correctly but is verbose and less efficient in style. Using a list comprehension simplifies the code into a single line, improving readability and performance by eliminating the need for repeated append calls. This makes the code more Pythonic and easier to maintain.