**AI ASSISTED CODING**

*Lab 13 – Code Refactoring: Improving Legacy Code with AI  
Suggestions*

Name :D.Nagamrutha

Roll no: 2503A51L06

**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.

**Prompt:**

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 \*

Give refactored version with separate functions and clear modular design.

**Code Generated:**

**A screen shot of a computer program

AI-generated content may be incorrect.**

**Output:**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Observation:**

This refactored code demonstrates a clean, modular approach to calculating the area of different shapes. By separating each shape’s logic into its own function and using a dictionary-based dispatch in `calculate\_area`, the code is easy to read, maintain, and extend. Error handling is centralized, and the design supports scalability if more shapes are added in the future. The example usage at the end clearly shows how to use the main function.

**Task Description #2 – 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

**Expected Output:**  
AI refactors with open() and try-except:

**Prompt:**

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

f.close()  
return data

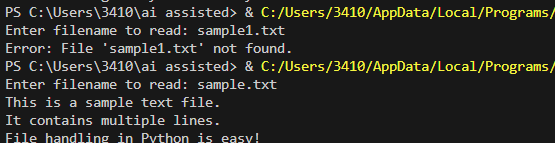
Give refactored version with open()and try-except block

**Code Generated:**

A computer screen shot of a program code

AI-generated content may be incorrect.

**Output:**

****

**Observation:**

This code provides a safe and user-friendly way to read the contents of a file in Python. It uses a `with` statement to ensure the file is properly opened and closed, and includes exception handling to manage missing files or other read errors gracefully. The function is well-documented, making it easy to understand and maintain. The user is prompted to enter the filename, making the script interactive and flexible for different file inputs.

**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.

**Prompt:**

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

Give refactored version by improving names ,readability and add docstrings

**Code Generated:**

A screen shot of a computer program

AI-generated content may be incorrect.

**Output:**

A black screen with yellow and white text

AI-generated content may be incorrect.**Observation:**

This code provides a well-structured and readable implementation of a `Student` class. It uses clear attribute names (`name`, `age`, `marks`), stores marks as a list for flexibility, and includes meaningful docstrings for documentation. The `details` method prints student information in a user-friendly format, and the `total` method efficiently calculates the sum of marks. The example usage demonstrates how to create a student object and display their details and total marks, making the code easy to understand and extend.

**Task Description #4 – 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)

**Expected Output:**

AI suggested a list comprehension

**Prompt:**

nums = [1,2,3,4,5,6,7,8,9,10]  
squares = []  
for i in nums:  
squares.append(i \* i)  
Give refactored version with efficient loop and list comprehension

**A screenshot of a computer program

AI-generated content may be incorrect.Code Generated:**

A screenshot of a computer

AI-generated content may be incorrect.**Output:**

**Observation:**

This code efficiently computes the squares of numbers from 1 to 10 using a list comprehension, which is both concise and fast. It first prints the original list of numbers and then their corresponding squares, making the output clear and easy to interpret. The use of list comprehension improves readability and performance compared to a traditional loop.