# **AI ASSISTED CODING**

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

**Roll no:** 2503A51L36

**Name:** Suhana Rehan

**Batch:** 25BTCAICSB20

**Assignment number:** 13.2

**Task 1:** **Remove Repetition**

Provide AI with the following redundant code and ask it to refactor

**Prompt:**

Refactor the given Python function calculate\_area that repeats logic into a cleaner, modular version using either dictionary-based dispatch or separate functions for rectangle, square, and circle.

**Code provided:**

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

if shape == "rectangle":

return x \* y

elif shape == "square":

return x \* x

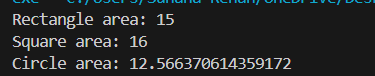
elif shape == "circle":

return 3.14\* x \* x

**Code:**

****

**Output:**

****

**Observation:**

* The refactored calculate\_area function uses dictionary-based dispatch and separate functions for each shape.
* This modular approach eliminates repeated logic, improves readability, and makes it easy to add new shapes or modify existing ones.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Task 2:** **Error Handling in Legacy Code**

Legacy function without proper error handling

**Prompt:**

Refactore the given python function with open() and try-except with an example

**Code provided:**

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

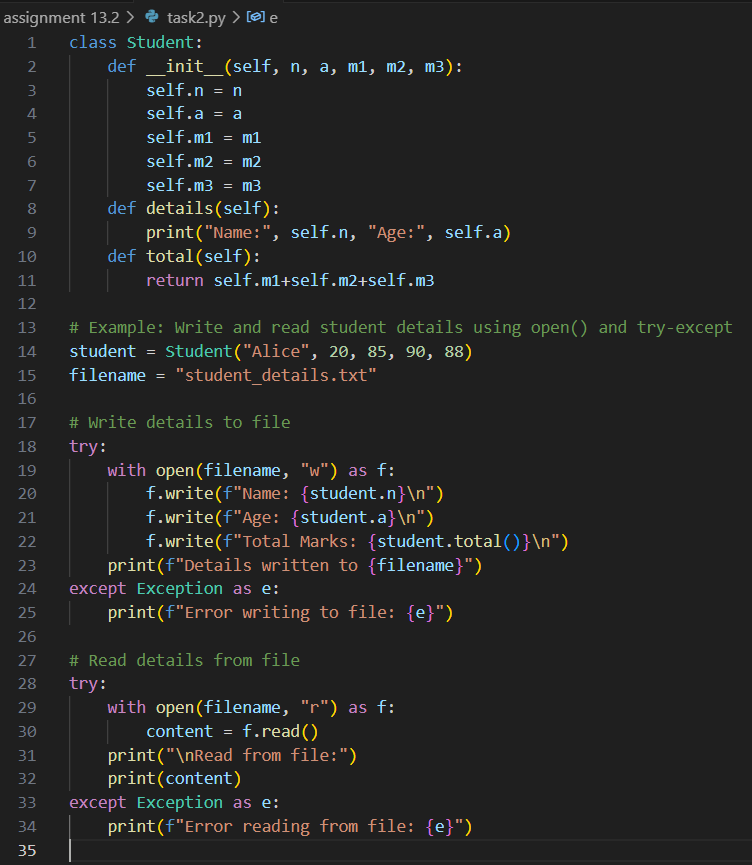
def details(self):

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

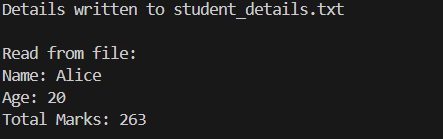
def total(self):

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

**Code:**

****

**Output:**

****

**Observation:**

The code demonstrates safe file operations using open() and try-except blocks.

AI has successfully writes and reads student details, handling any file errors gracefully.

This approach ensures robust file handling and prevents program crashes due to file-related issues.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Task 3:** **Complex Refactoring**

Provide this legacy class to AI for readability and modularity improvements:

**Prompt:**

Refactor the legacy Student class by improving variable naming (name, age, marks), adding docstrings, enhancing print readability, and modularizing marks handling (e.g., storing in a list and using sum).

**Code provided:**

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

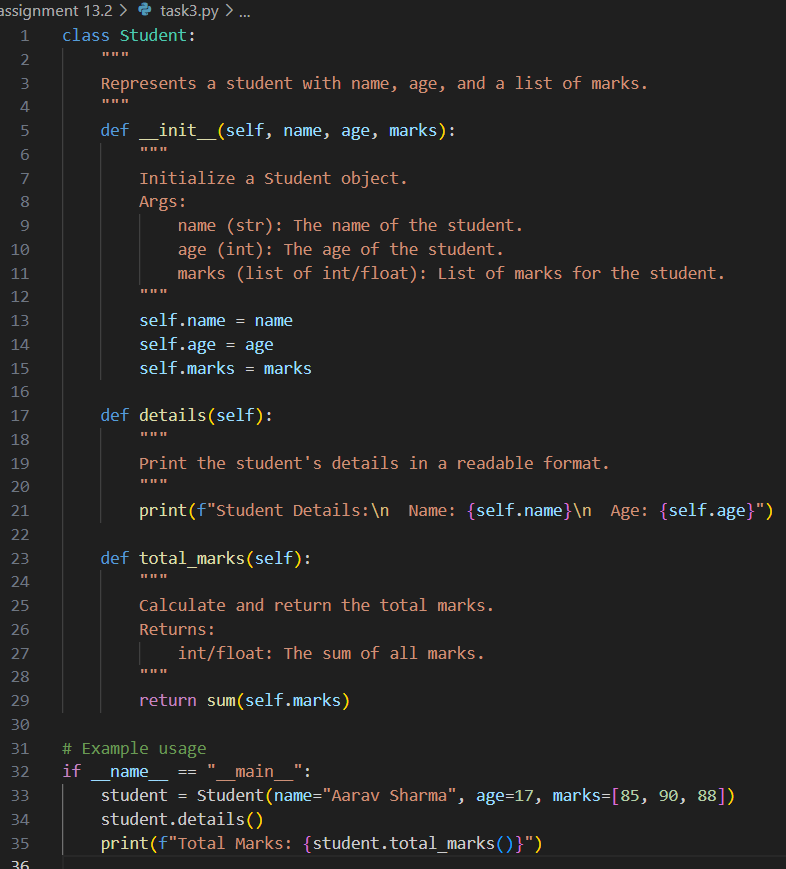
def details(self):

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

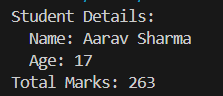
def total(self):

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

**Code:**

****

**Output:**

****

**Observation:**

**AI** made the program more readable and easy to understand and it added the doc string for more readability and it add sum fuction to calculate the total marks.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Task 4:** **Inefficient Loop Refactoring**

Refactor this inefficient loop with AI help

**Prompt:**

**Code provided:**

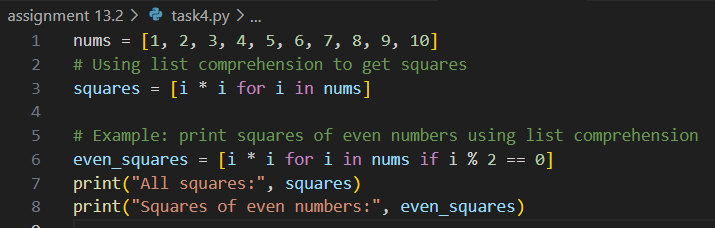
nums = [1,2,3,4,5,6,7,8,9,10]

squares = []

for i in nums:

squares.append(i \* i)

**Code:**

****

**Output:**

****

**Observation:**

* AI suggested a list comprehension
* Ai added the comment line for better understanding
* It made the inefficient loop , efficient and working