AI ASSISTED CODING LAB

ASSIGNMENT-13.2

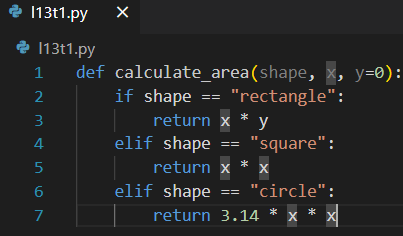
ENROLLMENT NO :2503A51L29 BATCH NO: 19

NAME: K. Shashikumar

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

**CODE GENERATED:**

****

**OUTPUT:**

**A black screen with white text

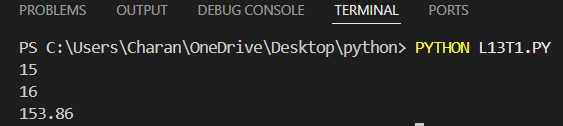
AI-generated content may be incorrect.**

**CORRECTED CODE:**

**A screen shot of a computer program

AI-generated content may be incorrect.**

**OUTPUT:**

****

**OBSERVATION:** 

**Repetition** – The code repeats logic (x \* x for both square and circle).

**Inconsistent Parameters** –

* rectangle uses both x and y.
* square and circle ignore y, which is confusing.

**Hard to Extend** – Adding new shapes means more elif blocks.

**No Error Handling** – Passing an unknown shape returns None.

**Magic Number** – Uses 3.14 instead of math.pi (less accurate)

**TASK DESCRIPTION 2:** Error Handling in Legacy CodeTask: Legacy function without proper error handling

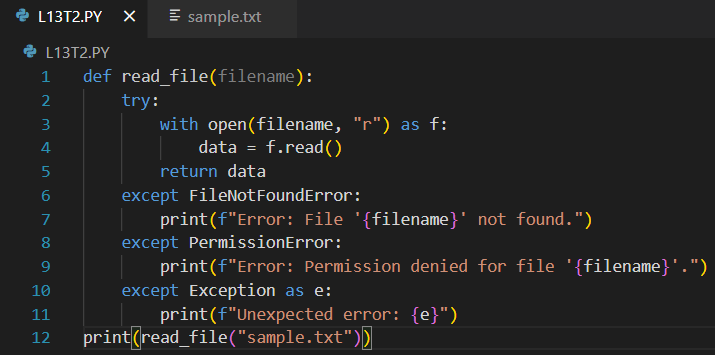
**Python Code**  
def read\_file(filename):  
f = open(filename, "r")  
data = f.read()

f.close()  
return data

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

**PROMPT:** Refactor the following legacy Python function to add proper error handling. The current code opens a file without using a context manager and does not handle exceptions. Rewrite it using with open () and try-except blocks to make it safe and Pythonic.

**CODE GENERATED:**

****

**OUTPUT:**

****

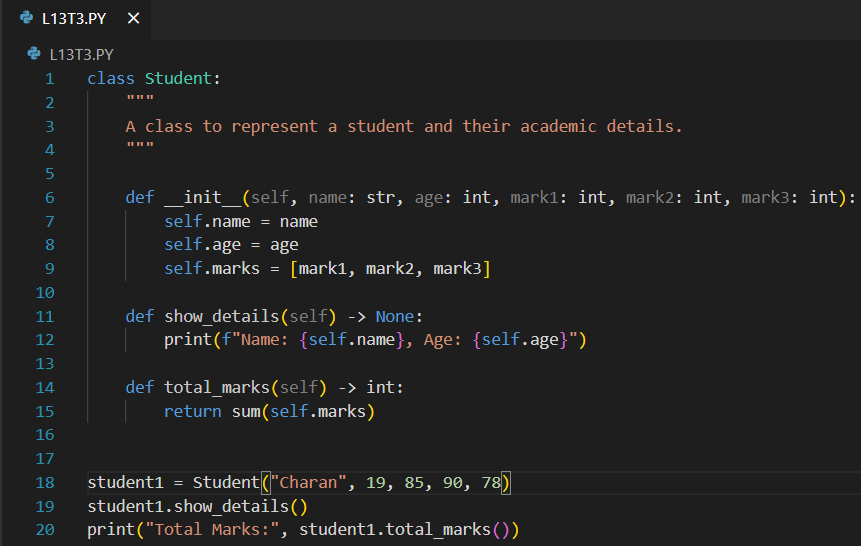
**Observation**

* The legacy code works only if the file exists and is readable. If the file is missing or inaccessible, it crashes with errors (FileNotFoundError, PermissionError, etc.).
* The refactored code uses with open () and try-except, so the file is safely closed and errors are handled gracefully with clear messages

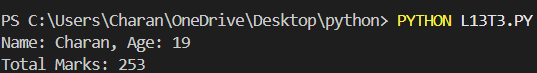
**TASK DESCRIPTION 3:** Complex Refactoring  
Task: Provide this legacy class to AI for readability and modularity improvements:  
**Python Code**  
class Student:  
def \_\_init\_\_(self, n, a, m1, m2, m3):  
self.n = n  
self.a = a  
self.m1 = m1  
self.m2 = m2  
self.m3 = m3  
def details(self):  
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:** Refactor the legacy Student class to improve readability and modularity. Store marks in a list, use descriptive variable names (name, age, marks), add docstrings, and provide methods to display student details and calculate total marks.

**CODE GENERATED:**

****

**OUTPUT:**

****

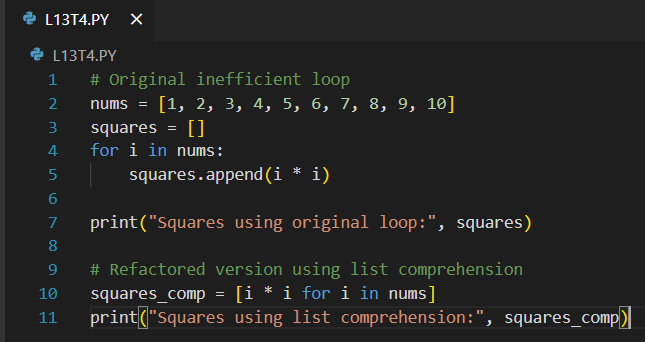
**Observation:**

* The class now has clear naming and docstrings.
* Marks are stored in a list, allowing easy sum calculation.
* Output is neatly formatted using f-strings.
* Methods show\_details () and total\_marks () work as expected.

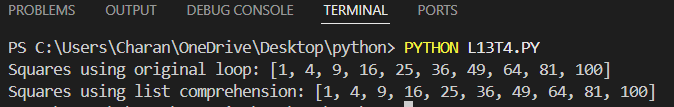
**TASK DESCRIPTION 4**: Inefficient Loop Refactoring  
Task: Refactor this inefficient loop with AI help  
**Python Code**nums = [1,2,3,4,5,6,7,8,9,10]  
squares = []  
for i in nums:  
squares. append (i \* i)  
**Expected Output:** AI suggested a list comprehension

**PROMPT:** Refactor the following inefficient Python loop to make it more concise and Pythonic. The loop calculates the squares of numbers in a list. Suggest a version using list comprehension

**CODE GENERATED:**

****

**OUTPUT:**

****

**Observation:**

* The original code uses a for loop and append () which is less concise.
* The refactored version is more Pythonic, concise, and often faster, using list comprehension.
* Output for both codes will be: [1, 4, 9, 16, 25, 36, 49, 64, 81, 100]