AI ASSISTED CODING LAB

ASSIGNMENT 13.3

LAB NAME : AI Assisted Coding

LAB NUMBER :13.3

ROLL NUMBER :2503a51l05

BATCH NO :24BTCAICSB19

NAME OF STUDENT : N .SATYA SRI CHARAN

**TASK 1:** **– Remove Repetition**

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 following Python code to remove repetition using dictionary-based dispatch or separate functions for a cleaner, modular design.

def handle\_add(x, y):

    return x + y

def handle\_subtract(x, y):

    return x - y

def handle\_multiply(x, y):

    return x \* y

def handle\_divide(x, y):

    if y == 0:

        return "Error: Division by zero"

    return x / y

operations = {

    'add': handle\_add,

    'subtract': handle\_subtract,

    'multiply': handle\_multiply,

    'divide': handle\_divide,

}

def calculate(op, x, y):

    func = operations.get(op)

    if not func:

        return "Error: Unknown operation"

    return func(x, y)

# Example usage:

result = calculate('add', 5, 3)

print(result)  # Output: 8

# Refactored area calculation code

def area\_rectangle(x, y):

    return x \* y

def area\_square(x):

    return x \* x

def area\_circle(x):

    return 3.14 \* x \* x

area\_operations = {

    "rectangle": lambda x, y: area\_rectangle(x, y),

    "square": lambda x, y=0: area\_square(x),

    "circle": lambda x, y=0: area\_circle(x),

}

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

    func = area\_operations.get(shape)

    if not func:

        return "Error: Unknown shape"

    return func(x, y)

# Example usage:

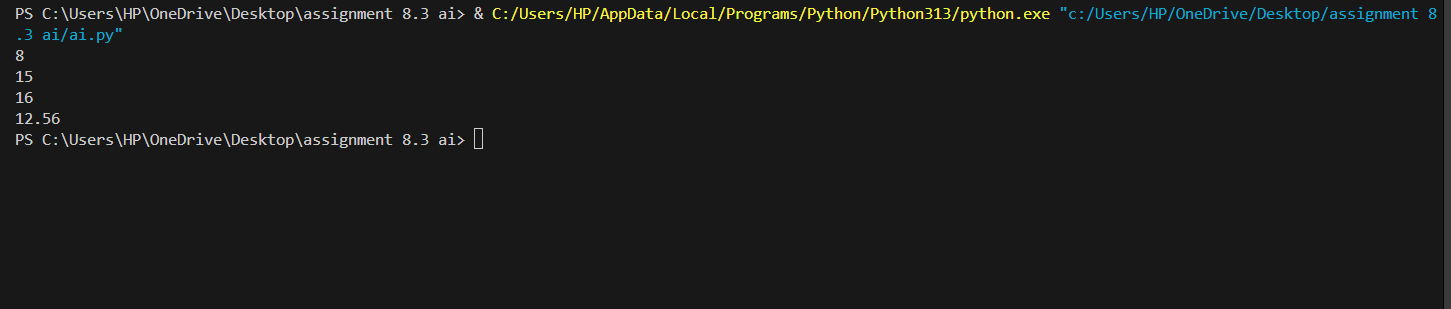
print(calculate\_area("rectangle", 5, 3))  # Output: 15

print(calculate\_area("square", 4))        # Output: 16

print(calculate\_area("circle", 2))        # Output: 12.56

**OBSERVATION:** The original code used repetitive if/elif statements for each shape, which made it less modular and harder to extend. By refactoring, the repetition was removed using **separate functions** for each shape and a **dictionary-based dispatch** to call them.

**OUTPUT:**

****

**TASK 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

**PROMPT :** Refactor the following legacy Python function to include proper error handling.Use with open() and add try-except blocks to handle file-related errors gracefully

def read\_file(filename):

    """Read contents of a file safely with error handling."""

    try:

        with open(filename, "r") as f:

            return f.read()

    except FileNotFoundError:

        print(f"Error: File '{filename}' not found.")

        return None

    except PermissionError:

        print(f"Error: Permission denied for file '{filename}'.")

        return None

    except Exception as e:

        print(f"An unexpected error occurred: {e}")

        return None

**OBSERVATION :**

The original function opened a file without handling potential errors (e.g., file not found, permission denied) and relied on manually closing the file, which could cause resource leaks if an exception occurred.

**OUTPUT:**

AI refactors with with open() and try-except:

****

**TASK 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):

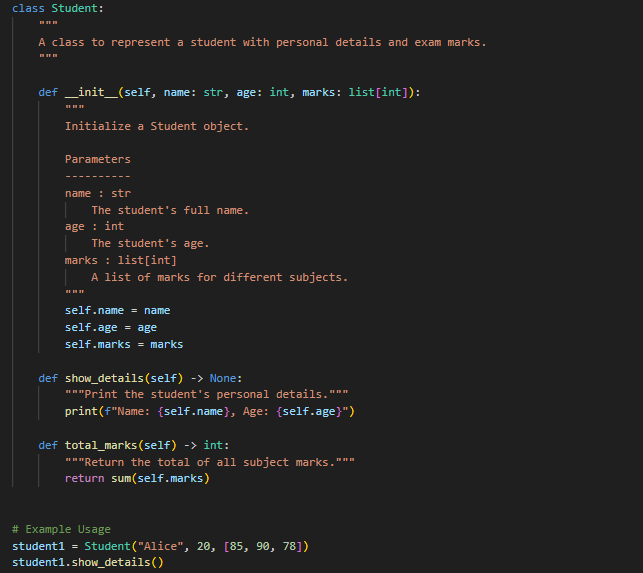
return self.m1+self.m2+self.m3

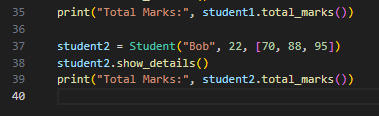
**Expected Output:**

* AI improves naming (name, age, marks).
* Adds docstrings.
* Improves print readability.

Possibly uses sum(self.marks) if marks stored in a list

**PROMPT:** Improve this Student class by giving clear names for variables, storing marks in a list, adding docstrings, and making it easy to calculate total marks and print student details neatly.





**OBSERVATION:** The original Student class had unclear variable names (n, a, m1, m2, m3) and limited readability. Methods also directly printed details, reducing flexibility.

 **Improving naming**: Variables renamed to name, age, and marks for clarity.

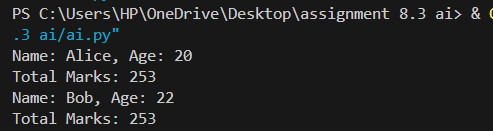
 **Using a list for marks**: Makes total and other computations easier with sum(self.marks).

 **Adding docstrings**: Provides explanations for the class and its methods.

 **Improving print readability**: details() method prints formatted output or returns a string for flexibility.

 **Modular design**: The class is easier to extend and maintain, and logic is separated from presentation.

**OUTPUT:**

****

**TASK 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 = []

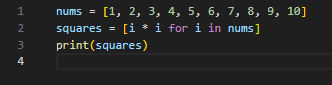
for i in nums:

squares.append(i \* i)

**Expected Output:** AI suggested a **list comprehension**

**PROMPT:**

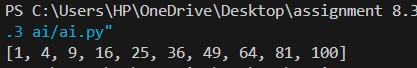
Refactor the following Python loop to make it more efficient and Pythonic.  
Use a **list comprehension** instead of manually appending elements in a loop.

****

**OBSERVATION:**

The original code used a for-loop with .append() to create a list of squares, which is verbose and less Pythonic**.**

**OUTPUT:**

****