AI ASSISTED CODING LAB

ASSIGNMENT-13.2

ENROLLMENT NO:2503A51L10

BATCH NO: 19

NAME: K.Praneeth

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
```

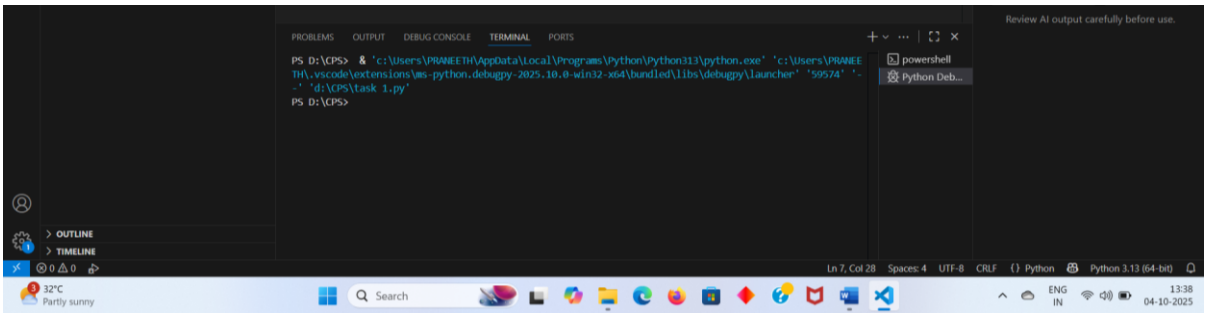
Expected Output

- Refactored version with dictionary-based dispatch or separate functions.
- Cleaner and modular design.

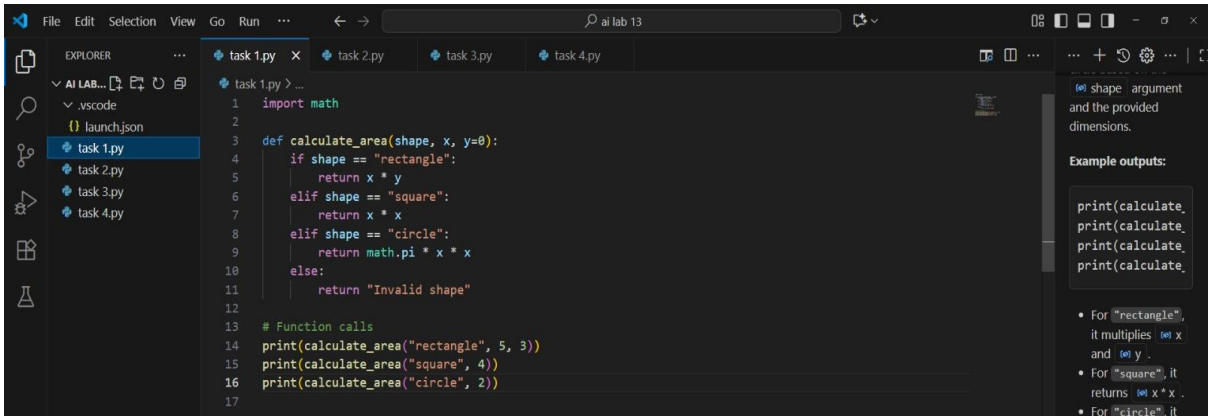
PROMPT: Provide AI with the following redundant code and ask it to refactor.

CODE GENERATED:

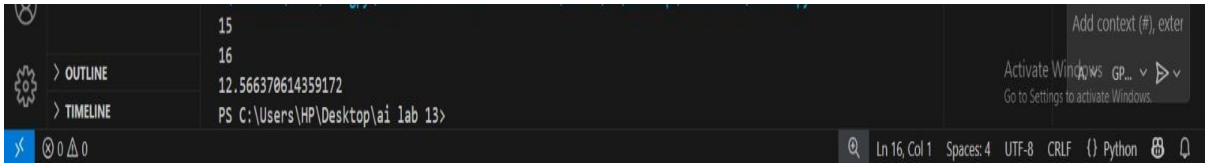
OUTPUT:



CORRECTED CODE:



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 Code

Task: Legacy function without proper error handling

Python Code

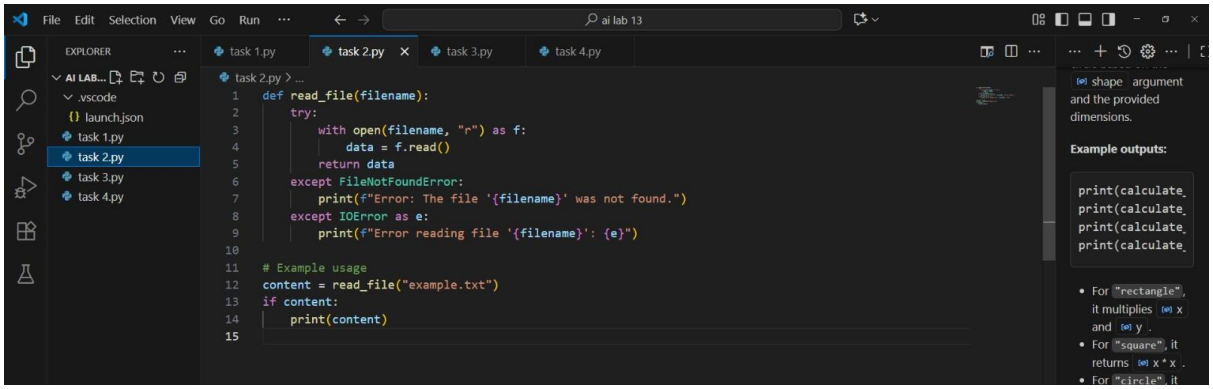
```
def read_file(filename):
f = open(filename, "r")
data = f.read()
f.close()
return data
```

Expected Output:

AI refactors with with open () and try-except

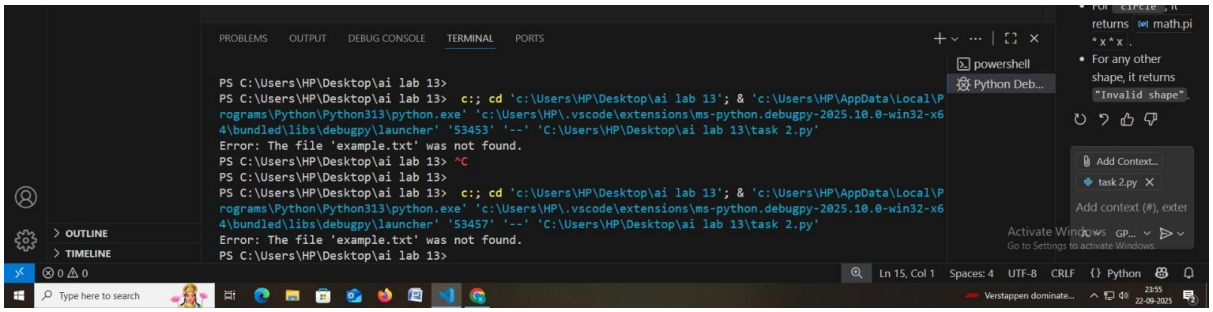
PROMPT: Refactor the following legacy Python function to add proper error handling. The current code opens a file without using a context manager and does not handle exceptions. Rewrite it using with open () and try-except blocks to make it safe and Pythonic.

CODE GENERATED:



```
def read_file(filename):
    try:
        with open(filename, "r") as f:
            data = f.read()
        return data
    except FileNotFoundError:
        print(f"Error: The file '{filename}' was not found.")
    except IOError as e:
        print(f"Error reading file '{filename}': {e}")

# Example usage
content = read_file("example.txt")
if content:
    print(content)
```

OUTPUT:



```
PS C:\Users\HP\Desktop\ai lab 13>
PS C:\Users\HP\Desktop\ai lab 13>  c:; cd 'c:\Users\HP\Desktop\ai lab 13'; & 'c:\Users\HP\AppData\Local\Programs\Python\Python313\python.exe' 'c:\Users\HP\.vscode\extensions\ms-python.debugpy-2025.10.0-win32-x64\bundled\libs\debugpy\launcher' '53453' '--' 'C:\Users\HP\Desktop\ai lab 13\task 2.py'
Error: The file 'example.txt' was not found.
PS C:\Users\HP\Desktop\ai lab 13> ^C
PS C:\Users\HP\Desktop\ai lab 13>
PS C:\Users\HP\Desktop\ai lab 13>  c:; cd 'c:\Users\HP\Desktop\ai lab 13'; & 'c:\Users\HP\AppData\Local\Programs\Python\Python313\python.exe' 'c:\Users\HP\.vscode\extensions\ms-python.debugpy-2025.10.0-win32-x64\bundled\libs\debugpy\launcher' '53457' '--' 'C:\Users\HP\Desktop\ai lab 13\task 2.py'
Error: The file 'example.txt' was not found.
PS C:\Users\HP\Desktop\ai lab 13>
```

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

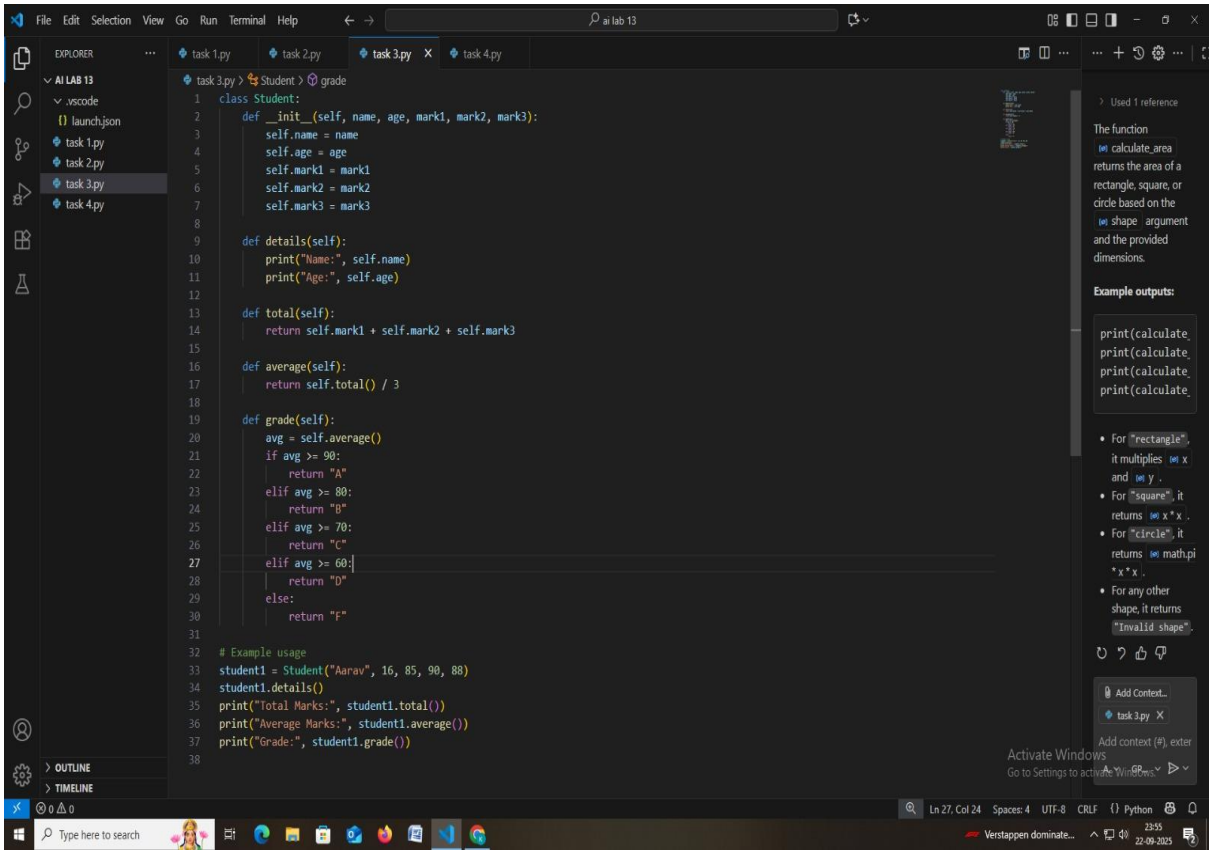
Expected Output:

- AI improves naming (name, age, marks).
- Adds docstrings.
- Improves print readability.
- Possibly uses sum(self.marks) if marks stored in a list

PROMPT:

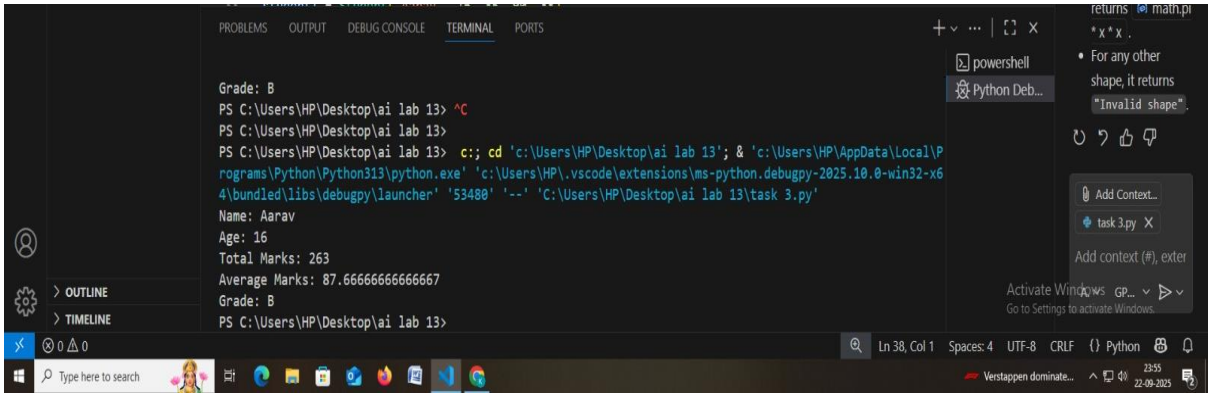
Refactor the legacy Student class to improve readability and modularity. Store marks in a list, use descriptive variable names (name, age, marks), add docstrings, and provide methods to display student details and calculate total marks.

CODE GENERATED:



```python
class Student:
    def __init__(self, name, age, mark1, mark2, mark3):
        self.name = name
        self.age = age
        self.mark1 = mark1
        self.mark2 = mark2
        self.mark3 = mark3

    def details(self):
        print("Name:", self.name)
        print("Age:", self.age)

    def total(self):
        return self.mark1 + self.mark2 + self.mark3

    def average(self):
        return self.total() / 3

    def grade(self):
        avg = self.average()
        if avg >= 90:
            return "A"
        elif avg >= 80:
            return "B"
        elif avg >= 70:
            return "C"
        elif avg >= 60:
            return "D"
        else:
            return "F"

# Example usage
student1 = Student("Aarav", 16, 85, 90, 88)
student1.details()
print("Total Marks:", student1.total())
print("Average Marks:", student1.average())
print("Grade:", student1.grade())
```

OUTPUT:



```
Grade: B
PS C:\Users\HP\Desktop\ai lab 13> ^C
PS C:\Users\HP\Desktop\ai lab 13>
PS C:\Users\HP\Desktop\ai lab 13>  c:; cd 'c:\Users\HP\Desktop\ai lab 13'; & 'c:\Users\HP\AppData\Local\Programs\Python\Python313\python.exe' 'c:\Users\HP\.vscode\extensions\ms-python.debugpy-2025.10.0-win32-x64\bundled\libs\debugpy\launcher' '53480' '--' 'C:\Users\HP\Desktop\ai lab 13\task 3.py'
Name: Aarav
Age: 16
Total Marks: 263
Average Marks: 87.66666666666667
Grade: B
PS C:\Users\HP\Desktop\ai lab 13>
```

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

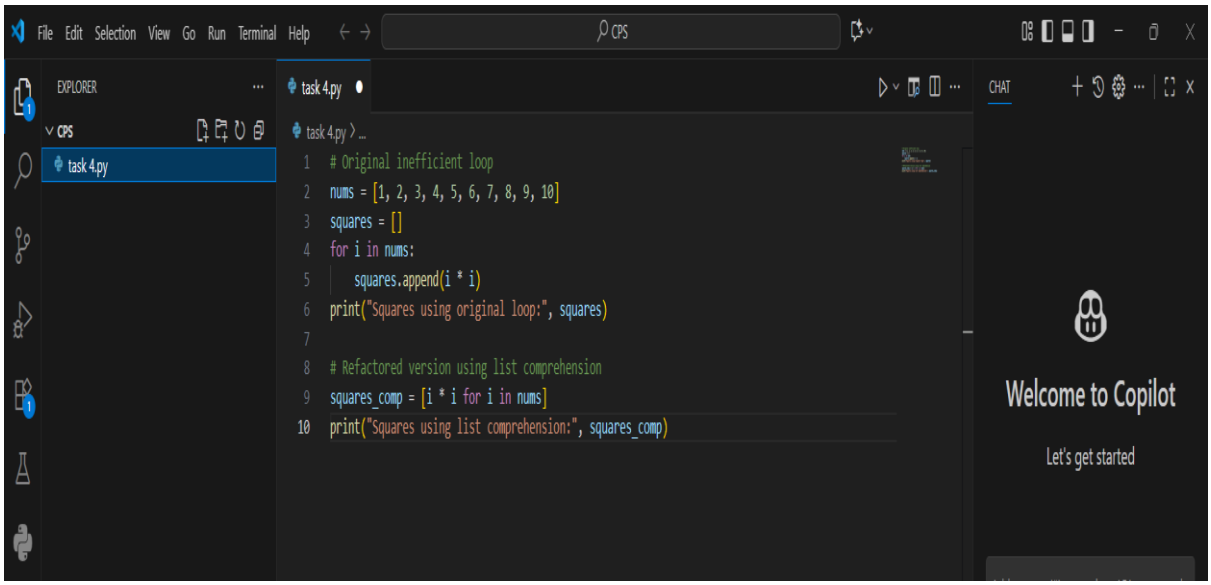
Python Code

```
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]