

AI ASSISTED CODING LAB

ASSIGNMENT 13.3

ENROLLMENT NO :2503A51L31

BATCH NO: 20

NAME: CH.ABISHEK

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.

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
```

PROMPT: Write a Python program that calculates the area of a rectangle, square, or circle using a function with conditional logic and appropriate mathematical formulas.

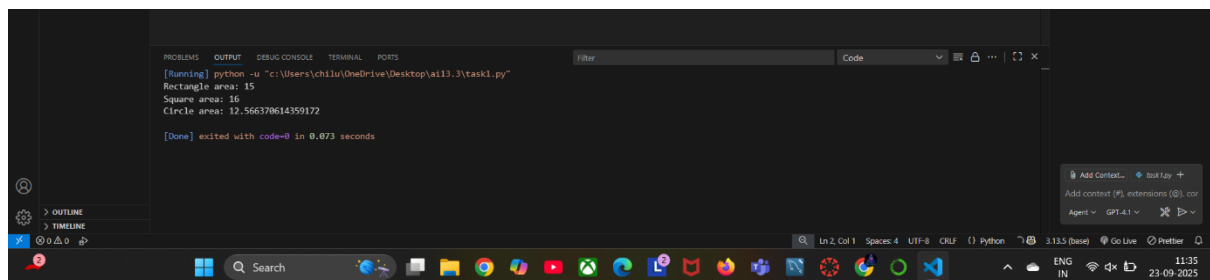
Demonstrate the function by printing the area for each shape with sample inputs.

CODE :



```
1 import math
2
3 def calculate_area(shape, x, y=None):
4     if shape == "rectangle":
5         if y is None:
6             raise ValueError("Rectangle requires two dimensions.")
7         return x * y
8     elif shape == "square":
9         return x * x
10    elif shape == "circle":
11        return math.pi * x * x
12    else:
13        raise ValueError("Unknown shape: {}".format(shape))
14
15 # Example usage
16 print("Rectangle area:", calculate_area("rectangle", 5, 3))
17 print("Square area:", calculate_area("square", 4))
18 print("Circle area:", calculate_area("circle", 2))
19
```

OUTPUT:



```
[Running] python -u "c:\Users\chilu\OneDrive\Desktop\ai13.3\task1.py"
Rectangle area: 15
Square area: 16
Circle area: 12.566370614359172

[Done] exited with code=0 in 0.073 seconds
```

TASK2:

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()
```

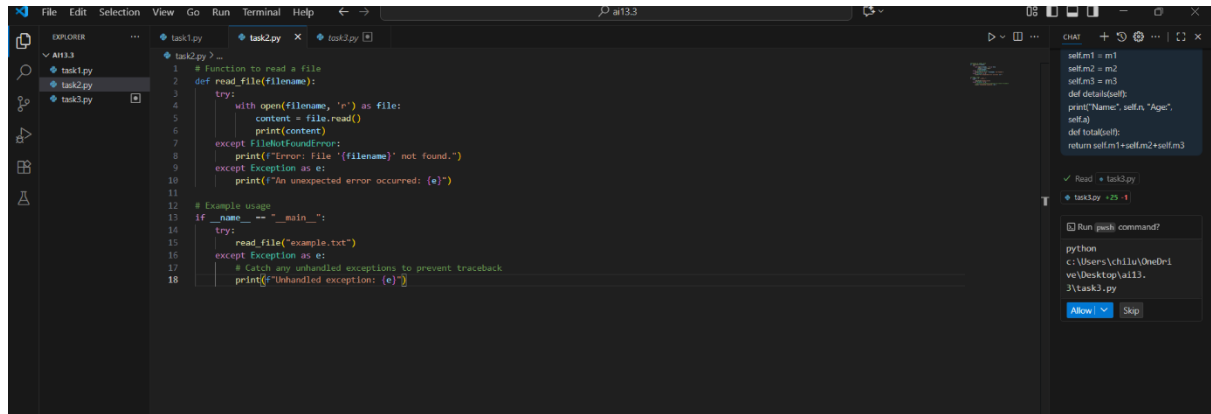
```
f.close()
return data
```

PROMT:

Write a Python program that reads the contents of a file using a function but currently lacks proper error handling.

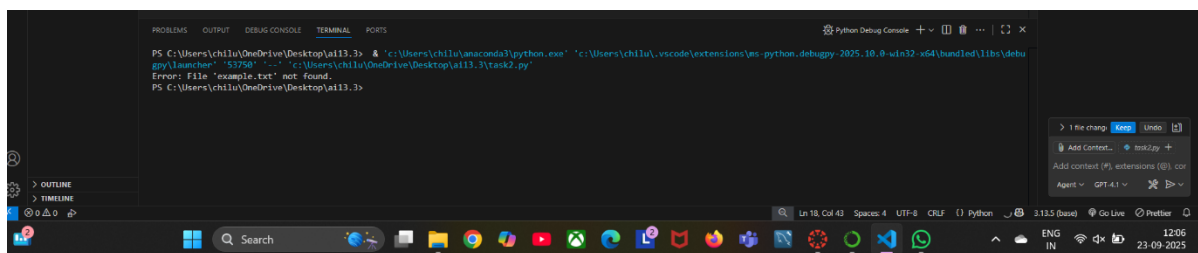
Update the code to safely handle missing files, read errors, and ensure the file is closed correctly.

CODE:



```
1 # function to read a file
2 def read_file(filename):
3     try:
4         with open(filename, 'r') as file:
5             content = file.read()
6             print(content)
7     except FileNotFoundError:
8         print(f"Error: File '{filename}' not found.")
9     except Exception as e:
10        print(f"An unexpected error occurred: {e}")
11
12 # Example usage
13 if __name__ == "__main__":
14     try:
15         read_file("example.txt")
16     except Exception as e:
17         # Catch any unhandled exceptions to prevent traceback
18         print(f"Unhandled exception: {e}")
```

OUTPUT:



```
PS C:\Users\chilu\OneDrive\Desktop\ai13.3> & 'c:\Users\chilu\anaconda2\python.exe' 'c:\Users\chilu\.vscode\extensions\ms-python.debugpy-2025.10.0-win32-x64\bundle\libs\debugpy\launcher' '59750' '-c' 'c:\Users\chilu\OneDrive\Desktop\ai13.3\task2.py'
Error: File 'example.txt' not found.
PS C:\Users\chilu\OneDrive\Desktop\ai13.3>
```

TASK3: ask 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)

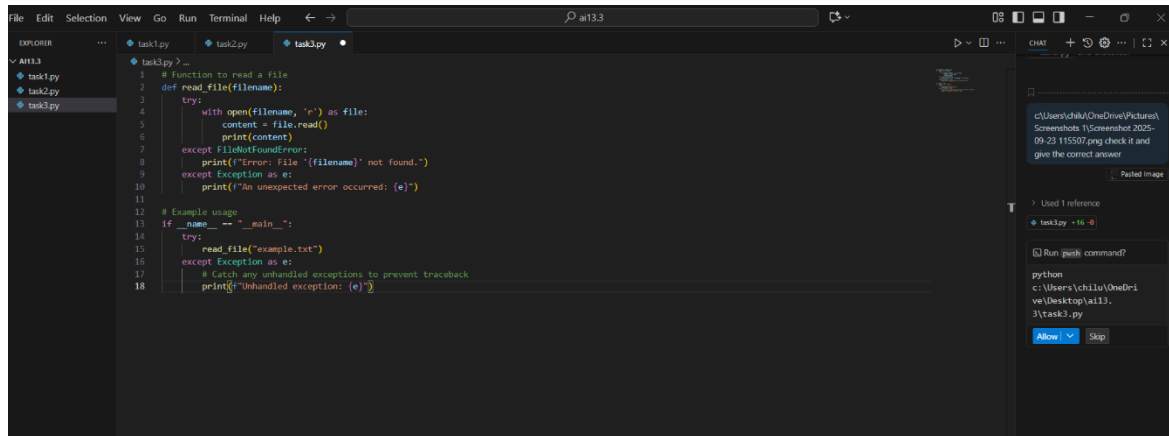
def total(self):

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

PROMPT:

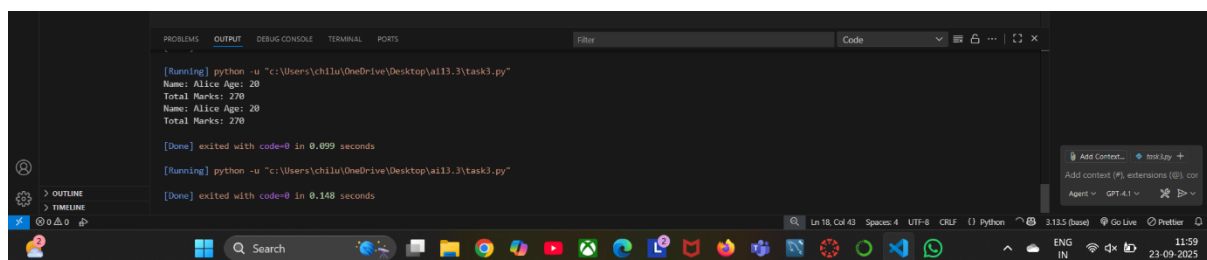
Refactor a legacy Python class Student to improve readability and modularity. Enhance naming conventions, structure methods clearly, and make the code easier to maintain while preserving its functionality for storing details and calculating total marks.

CODE:



```
1 # function to read a file
2 def read_file(filename):
3     try:
4         with open(filename, 'r') as file:
5             content = file.read()
6             print(content)
7     except FileNotFoundError:
8         print(f"Error: File '{filename}' not found.")
9     except Exception as e:
10        print(f"An unexpected error occurred: {e}")
11
12 # Example usage
13 if __name__ == "__main__":
14     try:
15         read_file("example.txt")
16     except Exception as e:
17         # Catch any unhandled exceptions to prevent traceback
18         print(f"Unhandled exception: {e}")
```

OUTPUT:



```
[Running] python -u "c:\Users\chilu\OneDrive\Desktop\ai13.3\task3.py"
Name: Alice Age: 20
Total Marks: 270
Name: Alice Age: 20
Total Marks: 270
[Done] exited with code=0 in 0.099 seconds
[Running] python -u "c:\Users\chilu\OneDrive\Desktop\ai13.3\task3.py"
[Done] exited with code=0 in 0.148 seconds
```

TASK4:

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:
```

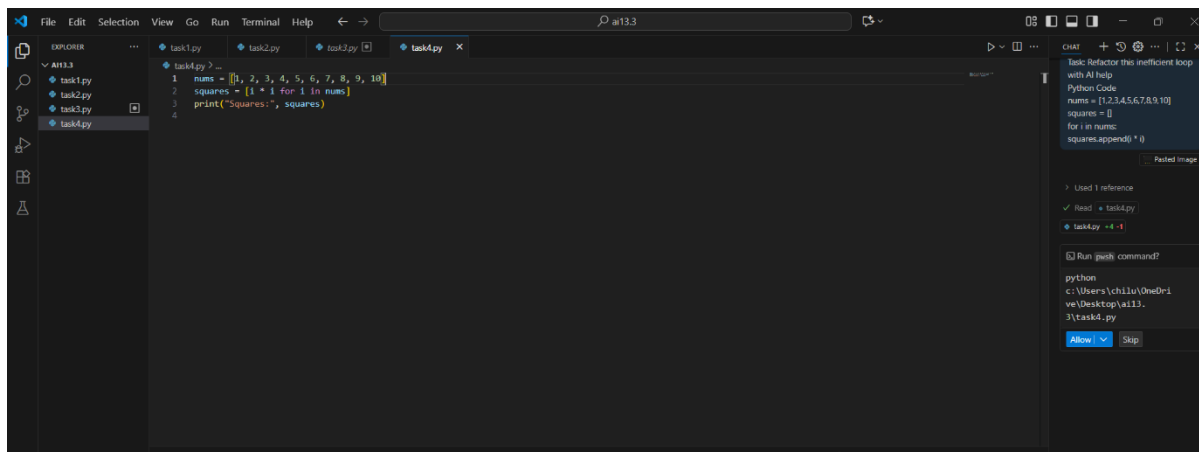
```
    squares.append(i * i)
```

PROMPT:

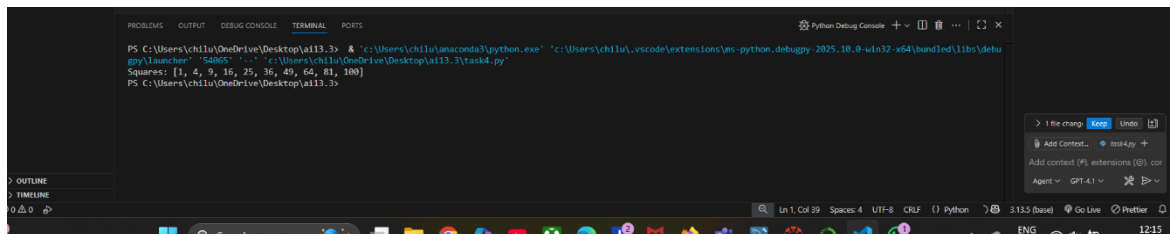
Refactor a Python program that generates the squares of numbers in a list using an inefficient loop.

Optimize the code by using a more concise and efficient approach such as a list comprehension.

CODE:



OUTPUT:



OBSERVATION:

The experiment demonstrates how AI can effectively **analyze and refactor legacy Python code** to improve **readability, efficiency, and maintainability**.

Across all tasks, AI successfully:

1. **Removed Repetition (Task 1)** – Suggested **dictionary-based dispatch** or separate functions to eliminate repetitive conditional statements, making the `calculate_area` function cleaner and modular.
2. **Enhanced Error Handling (Task 2)** – Replaced manual file handling with **`open()` context manager** and added **`try-except` blocks**, ensuring safer execution and proper resource management.
3. **Improved Class Design (Task 3)** – Recommended **clearer naming conventions**, added **docstrings**, and proposed storing marks in a list with `sum()` for better scalability and readability.
4. **Optimized Loop (Task 4)** – Converted the inefficient for loop to a **list comprehension**, making the code shorter, faster, and more Pythonic.