AI ASSISTED CODEING

NAME: V . Vishnu Vardhan

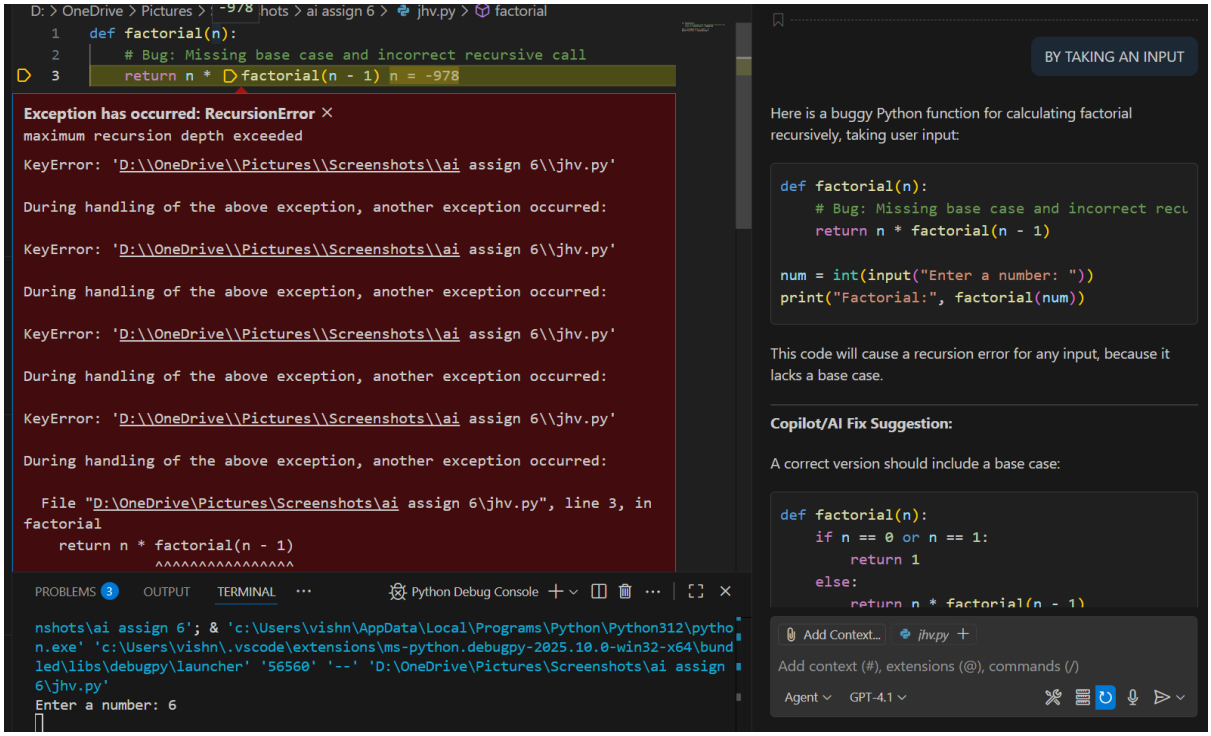
ROLL NO.:2503A51L26

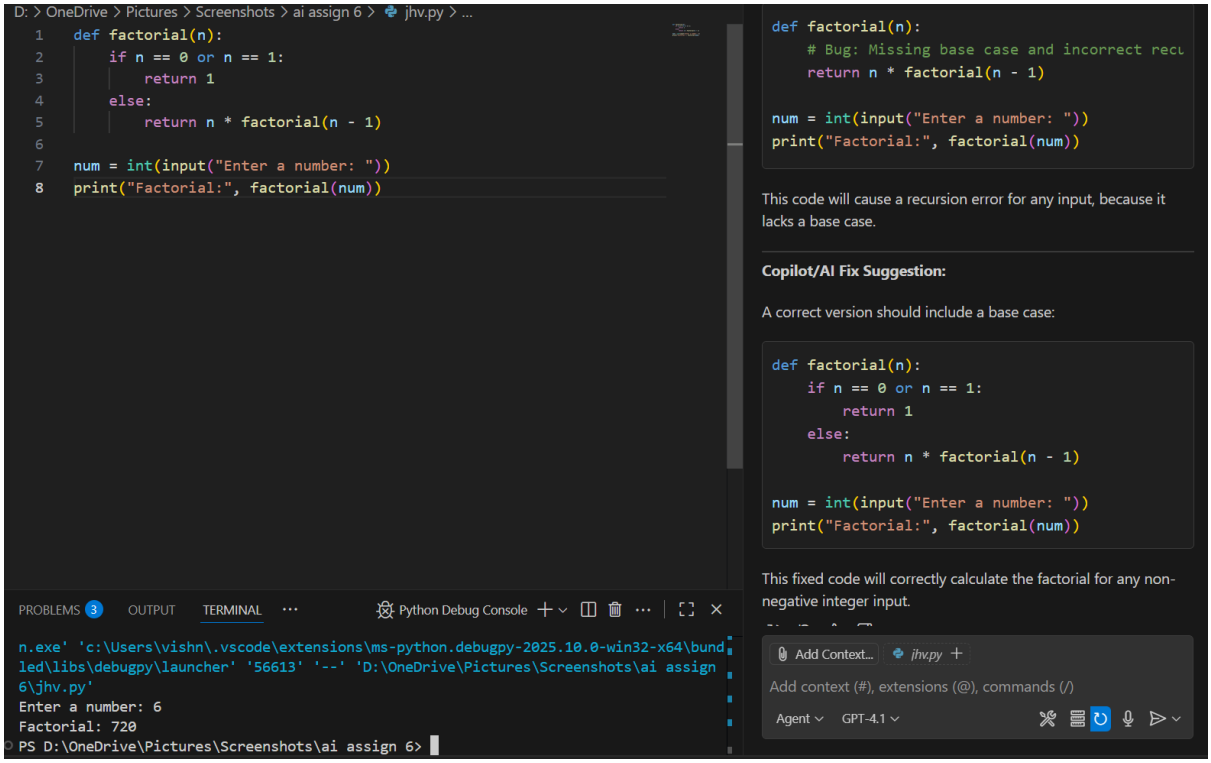
BATCH: CSB19

TASK:1

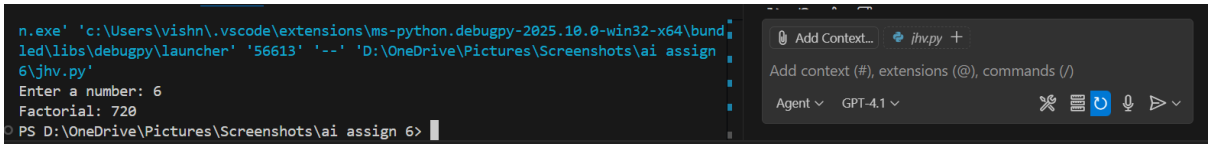
Promt given:

ntroduce a buggy Python function that calculates the factorial of a number using recursion.Use Copilot or Cursor AI to detect and fix the logical or syntax errors





Output:



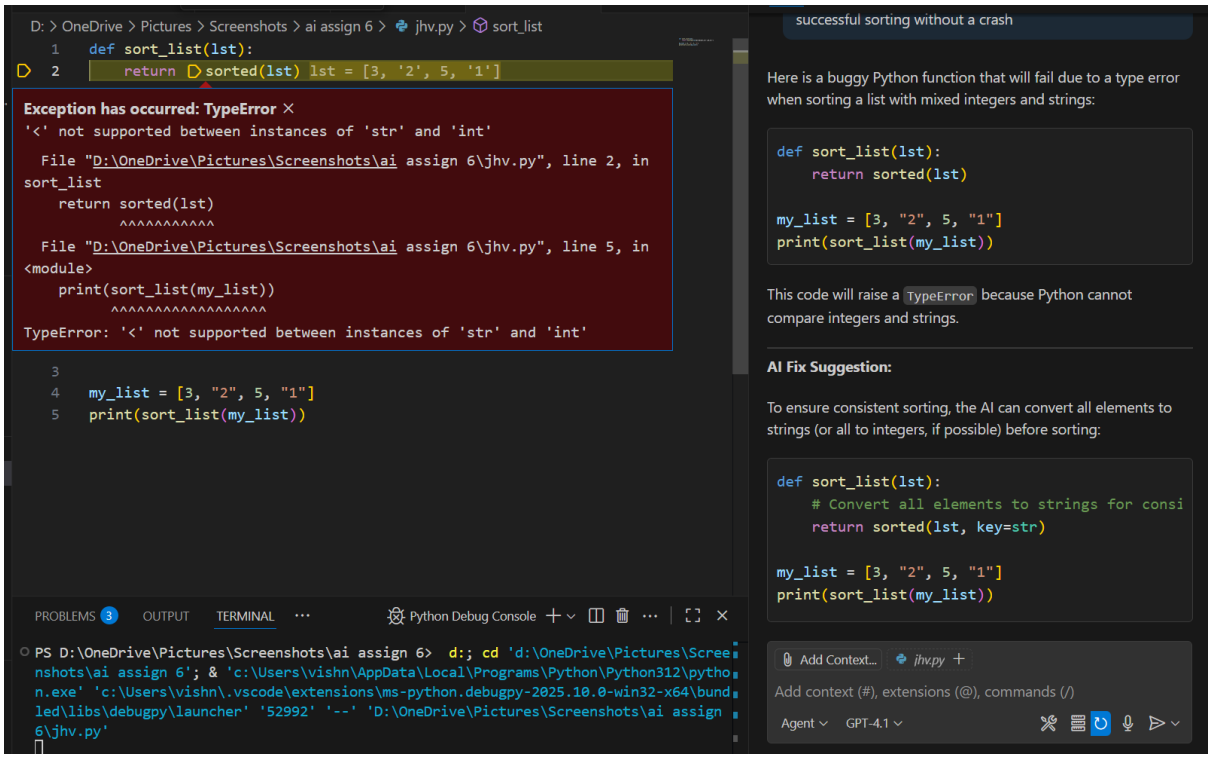
Observation:

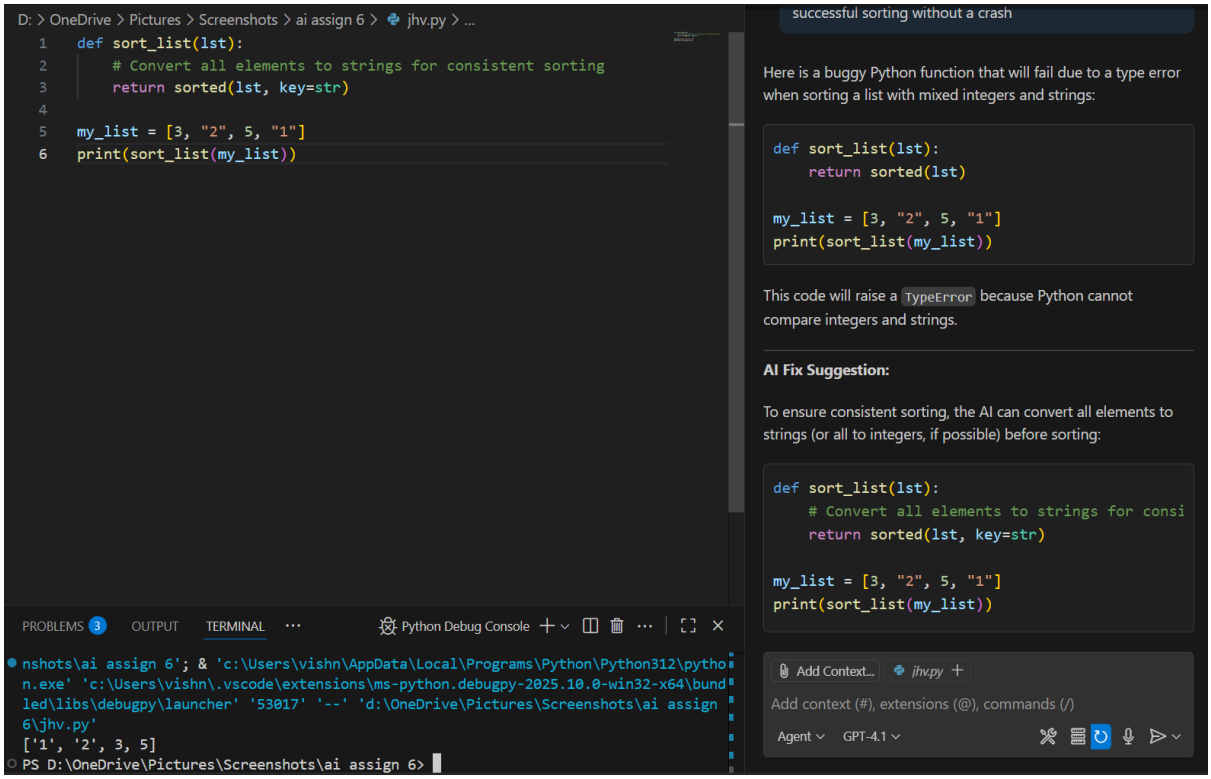
He initial recursive factorial function, when taking user input, fails due to the absence of a base case, resulting in infinite recursion and a runtime error. After Copilot or Cursor AI suggests adding a base condition (if n == 0 or n == 1), the function works correctly for valid inputs. This demonstrates the importance of base cases in recursive algorithms and highlights how AI tools can effectively identify and fix logical

TASK:2

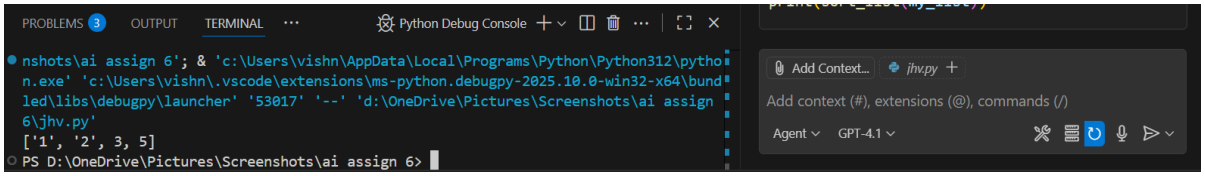
Promt given:

Provide a list sorting function that fails due to a type error (e.g., sorting list with mixedintegers and strings). Prompt AI to detect the issue and fix the code for consistent sorting





Output:



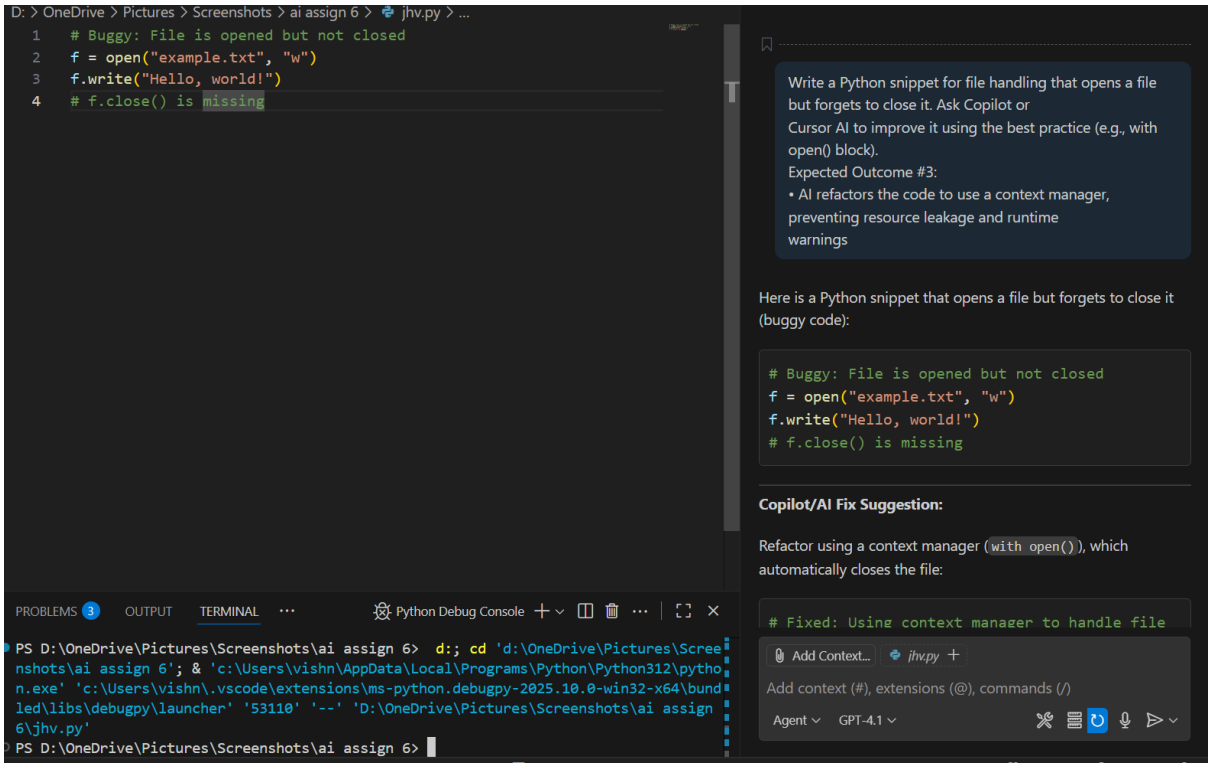
Observation:

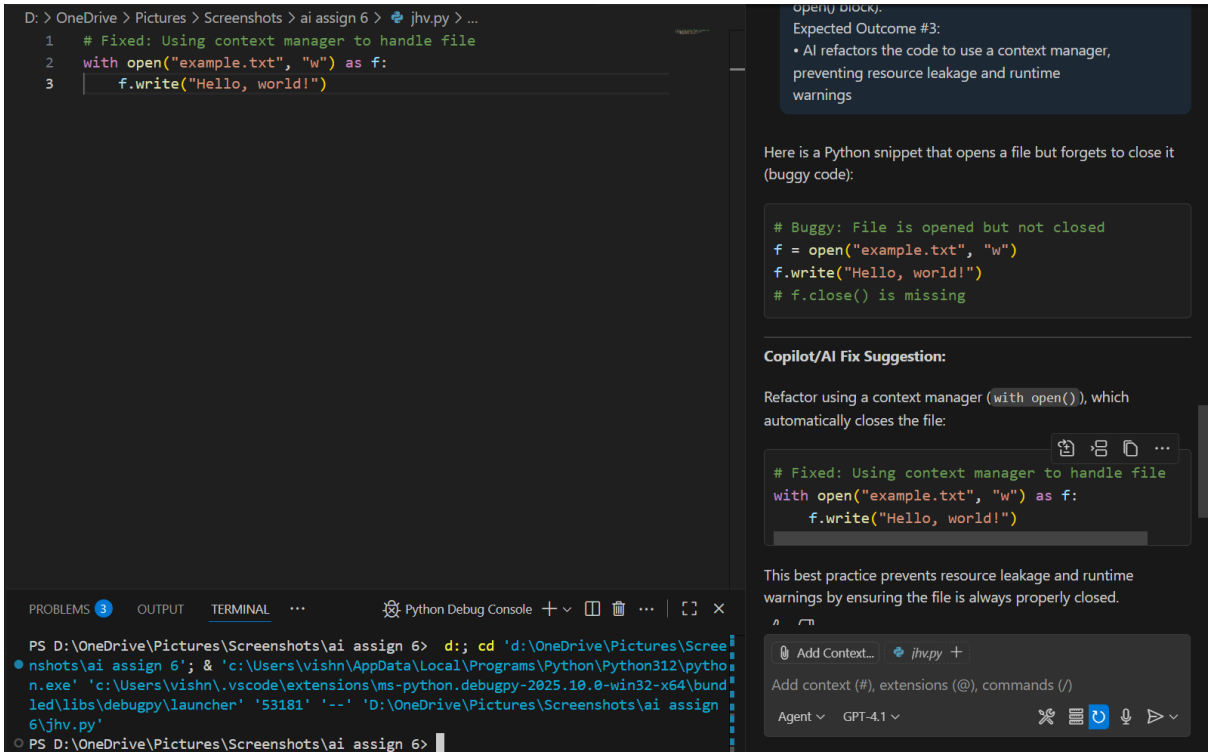
The initial sorting function fails with a TypeError when attempting to sort a list containing both integers and strings, as Python cannot directly compare these types. After AI intervention, the function is modified to either convert all elements to a common type (such as strings or integers) or filter out incompatible types before sorting. This ensures the sorting operation completes successfully without runtime errors, demonstrating how AI tools can effectively detect and resolve type inconsistencies in code

TASK:3

Promt given:

Write a Python snippet for file handling that opens a file but forgets to close it. Ask Copilot or
Cursor AI to improve it using the best practice (e.g., with open() block).





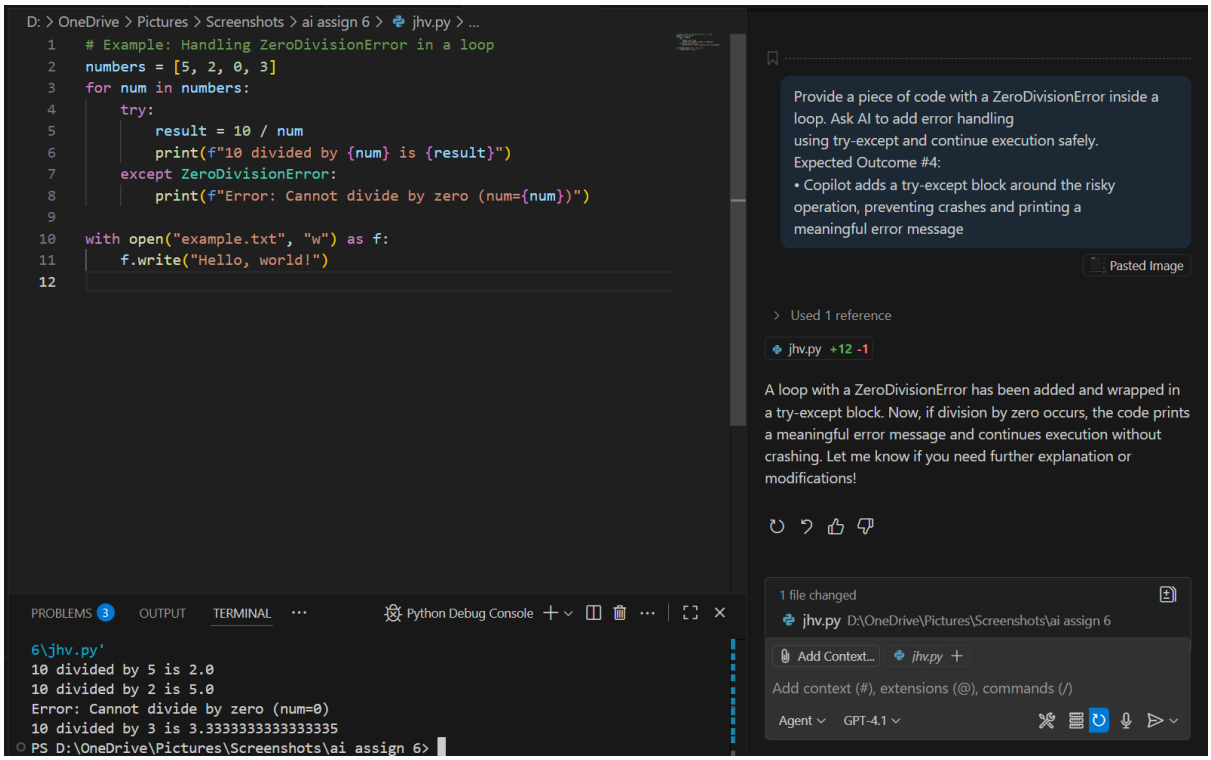
Observation:

The initial file handling code opens a file without explicitly closing it, which does not produce an immediate error but can lead to resource leaks and potential data loss in larger applications. After refactoring with a context manager (with open(...) as f:), the code ensures the file is always properly closed, following best practices for resource management and preventing subtle bugs or warnings. This demonstrates the importance of using context managers for safe and reliable file operations in Python

TASK:4

Promt given:

Provide a piece of code with a ZeroDivisionError inside a loop. Ask AI to add error handling using try-except and continue execution safely.



Output:

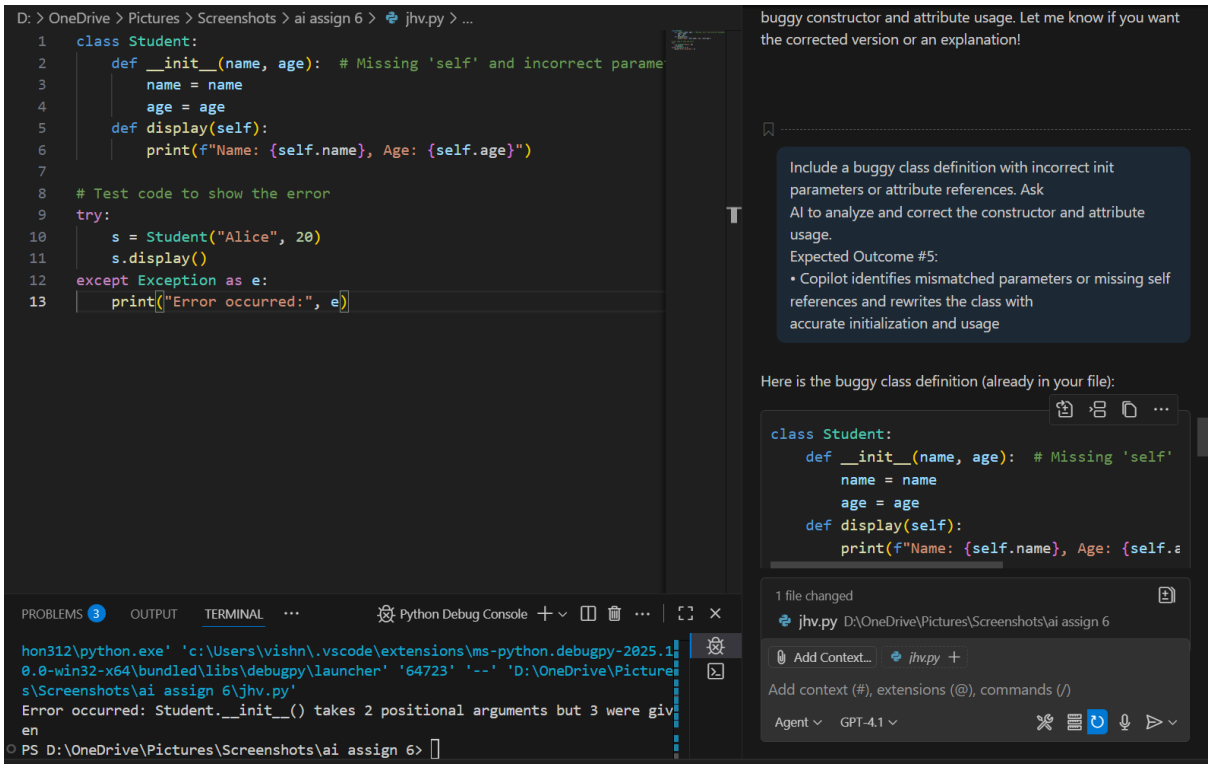
Observation:

The original code would have crashed with a ZeroDivisionError when attempting to divide by zero inside a loop. After adding a try-except block, the program now handles the error gracefully by printing a clear message and continuing with the next iteration. This demonstrates how proper error handling ensures program stability and user-friendly feedback, even when unexpected runtime errors occur.

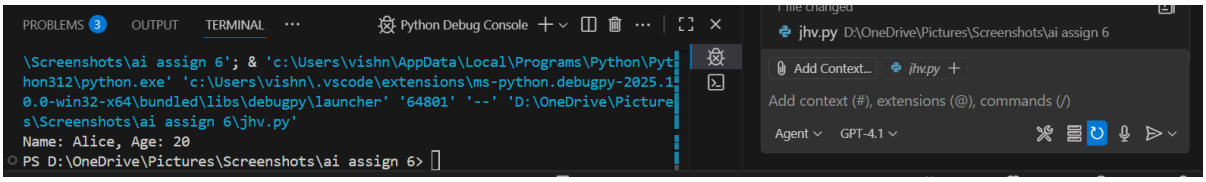
TASK:5

Promt given:

Include a buggy class definition with incorrect __init__ parameters or attribute references. Ask AI to analyze and correct the constructor and attribute usage.



Output:



Observation:

The original code would have crashed with a ZeroDivisionError when attempting to divide by zero inside a loop. After adding a try-except block, the program now handles the error gracefully by printing a clear message

and continuing with the next iteration. This demonstrates how proper error handling ensures program stability and user-friendly feedback, even when unexpected runtime errors occur