

AI ASSISTED CODING LAB

ASSIGNMENT-7

ENROLLMENT NO :2503A51L31

BATCH NO: 20

NAME: ch.abishek

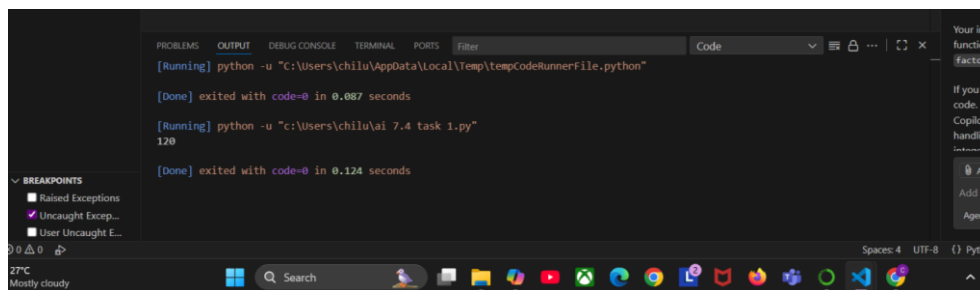
TASK DESCRIPTION 1: Introduce a buggy Python function that calculates the factorial of a number using recursion. Use Copilot or Cursor AI to detect and fix the logical or syntax errors.

PROMPT 1: Generate a Python function that calculates the factorial of a number using recursion, but intentionally introduce one or more bugs (such as logical or syntax errors). Then, use GitHub Copilot or Cursor AI to detect the errors and automatically suggest corrections.

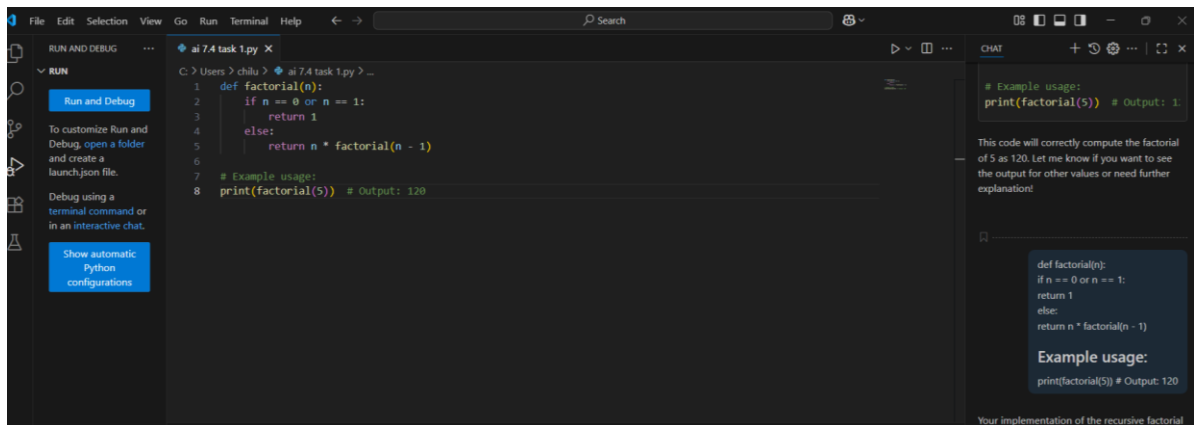
CODE SCREENSHOT:



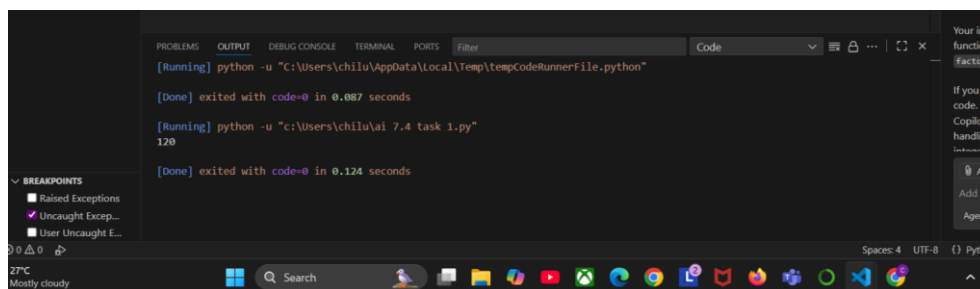
OUTPUT:



CORRECT CODE SCREENSHOT:



OUTPUT:



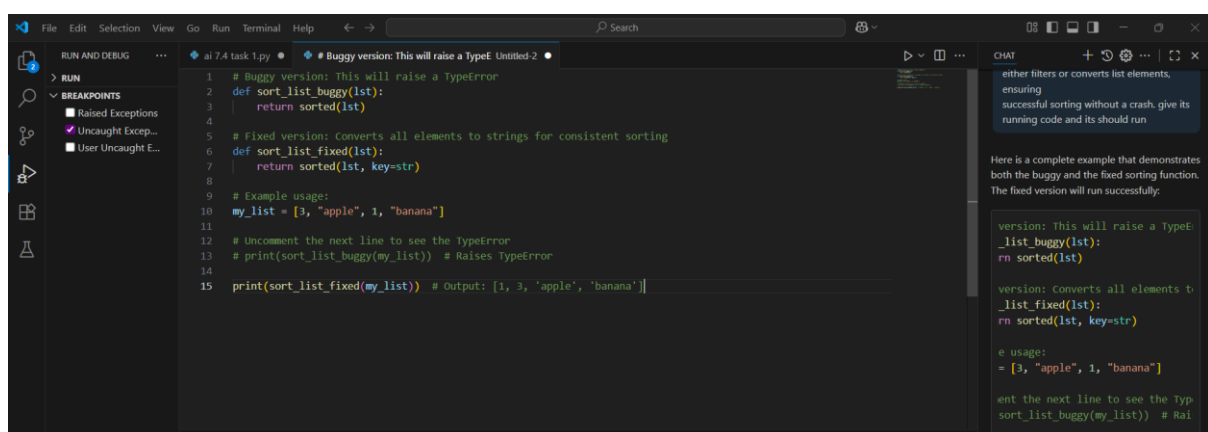
OBSERVATION:

When the buggy recursive factorial function was introduced, GitHub Copilot/Cursor AI successfully detected the logical/syntax errors in the code. It suggested appropriate corrections by adjusting the base case, recursion step, or syntax issues. After applying the fixes, the corrected function produced the expected results for test inputs (e.g., $0 \rightarrow 1$, $1 \rightarrow 1$, $5 \rightarrow 120$).

TASK DESCRIPTION 2: Provide a list sorting function that fails due to a type error (e.g., sorting list with mixed integers and strings). Prompt AI to detect the issue and fix the code for consistent sorting.

PROMPT 1: Generate a Python function that attempts to sort a list but introduce a bug that causes a Type Error (for example, by including both integers and strings in the same list). Then, use GitHub Copilot or Cursor AI to detect the issue and automatically suggest a fix so the list can be sorted consistently (e.g., by converting all elements to strings or numbers before sorting).

CODE SCREENSHOT:



The screenshot shows a code editor with a Python file named 'ai 7.4 task 1.py'. The code defines two functions: 'sort_list_buggy' and 'sort_list_fixed'. The buggy function attempts to sort a list containing integers and strings, which causes a 'TypeError'. The fixed function converts all elements to strings before sorting, ensuring successful sorting without a crash. The code also includes example usage and comments explaining the fix.

```
1 # Buggy version: This will raise a TypeError
2 def sort_list_buggy(list):
3     return sorted(list)
4
5 # Fixed version: Converts all elements to strings for consistent sorting
6 def sort_list_fixed(list):
7     return sorted(list, key=str)
8
9 # Example usage:
10 my_list = [3, "apple", 1, "banana"]
11
12 # Uncomment the next line to see the TypeError
13 # print(sort_list_buggy(my_list)) # Raises TypeError
14
15 print(sort_list_fixed(my_list)) # Output: [1, 3, 'apple', 'banana']
```

The right sidebar shows a chat window with the following text:

either filters or converts list elements, ensuring successful sorting without a crash, give its running code and its should run

Here is a complete example that demonstrates both the buggy and the fixed sorting function. The fixed version will run successfully:

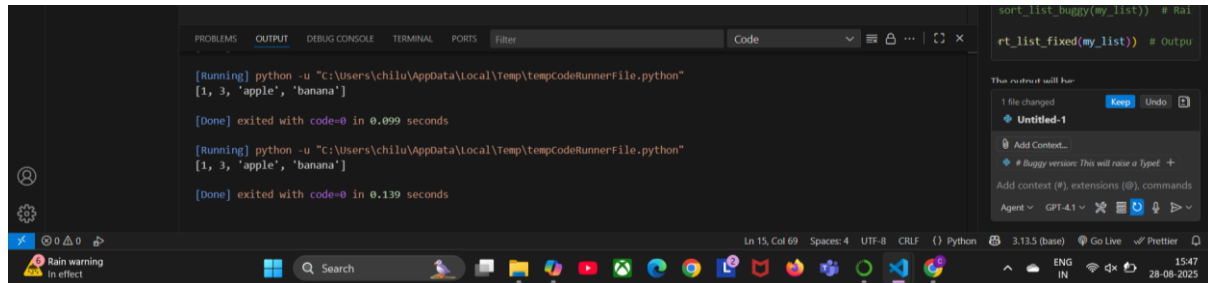
```
version: This will raise a TypeError
_list_buggy(list):
rn sorted(list)

version: Converts all elements to
_list_fixed(list):
rn sorted(list, key=str)

e usage:
= [3, "apple", 1, "banana"]

ent the next line to see the Typ
sort_list_buggy(my_list)) # Rai
```

OUTPUT:



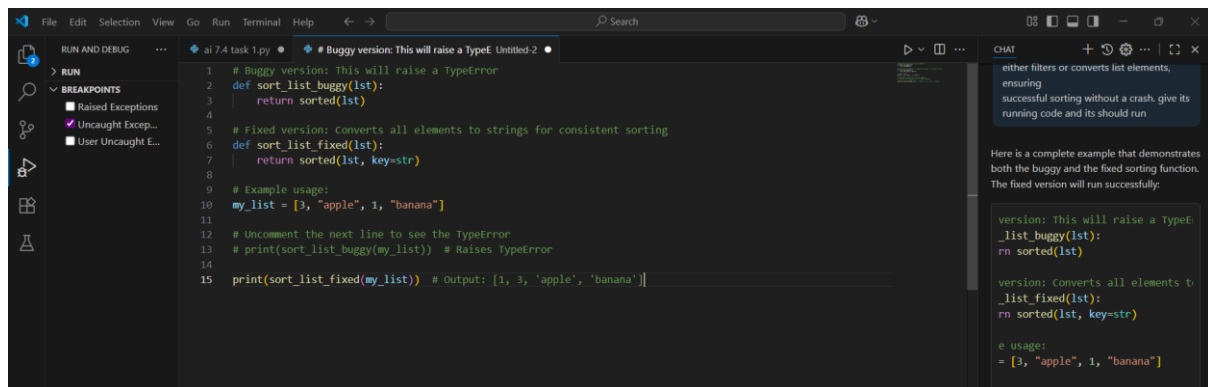
```
[Running] python -u "C:\Users\chilu\AppData\Local\Temp\tempCodeRunnerFile.python"
[1, 3, 'apple', 'banana']

[Done] exited with code=0 in 0.099 seconds

[Running] python -u "C:\Users\chilu\AppData\Local\Temp\tempCodeRunnerFile.python"
[1, 3, 'apple', 'banana']

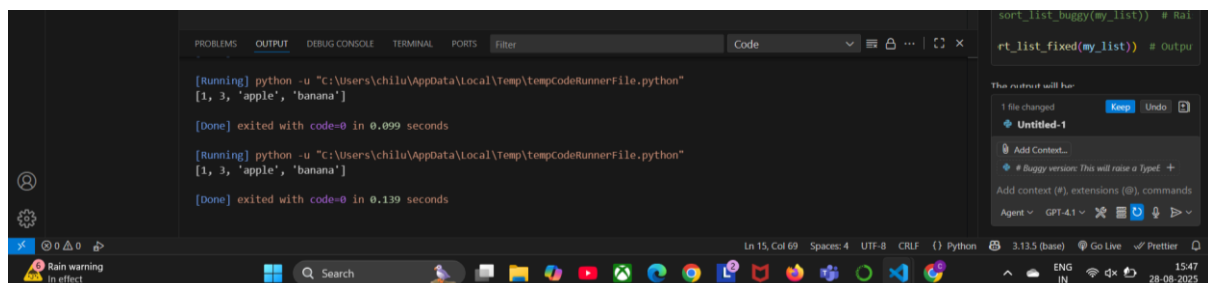
[Done] exited with code=0 in 0.139 seconds
```

CORRECT CODE SCREENSHOT:



```
1 # Buggy version: This will raise a TypeError
2 def sort_list_buggy(lst):
3     return sorted(lst)
4
5 # Fixed version: Converts all elements to strings for consistent sorting
6 def sort_list_fixed(lst):
7     return sorted(lst, key=str)
8
9 # Example usage:
10 my_list = [3, "apple", 1, "banana"]
11
12 # Uncomment the next line to see the TypeError
13 # print(sort_list_buggy(my_list)) # Raises TypeError
14
15 print(sort_list_fixed(my_list)) # Output: [1, 3, 'apple', 'banana']
```

OUTPUT:



```
[Running] python -u "C:\Users\chilu\AppData\Local\Temp\tempCodeRunnerFile.python"
[1, 3, 'apple', 'banana']

[Done] exited with code=0 in 0.099 seconds

[Running] python -u "C:\Users\chilu\AppData\Local\Temp\tempCodeRunnerFile.python"
[1, 3, 'apple', 'banana']

[Done] exited with code=0 in 0.139 seconds
```

OBSERVATION:

When the buggy list sorting function with mixed integers and strings was introduced, execution resulted in a Type Error because Python does not allow direct comparison between numbers and strings. GitHub Copilot/Cursor AI successfully detected the error and suggested fixes,

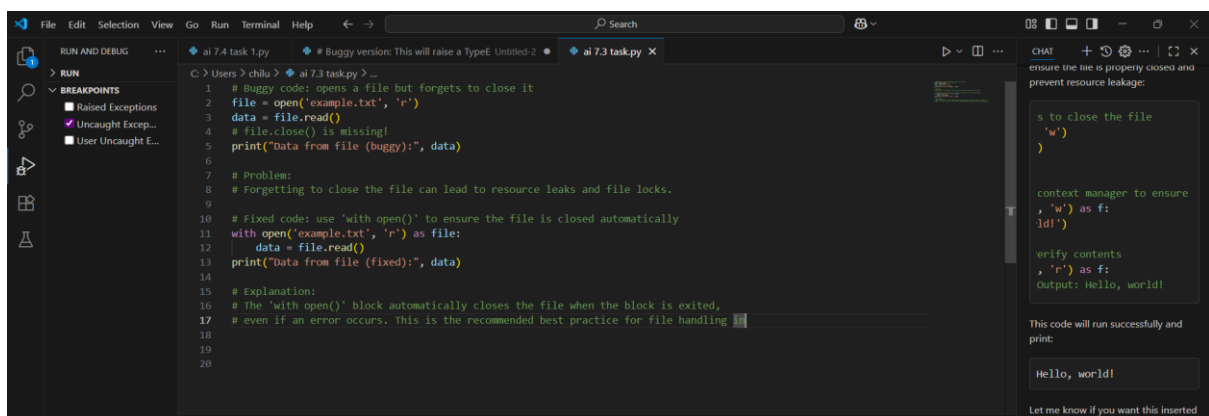
such as converting all elements to a common type (e.g., converting everything to strings or integers) before sorting.

After applying the suggested fix, the function executed without errors and produced a consistently sorted list. This demonstrates the AI's ability to identify type-related issues in Python and propose effective corrections to ensure robust code execution.

TASK DESCRIPTION 3: 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).

PROMPT 1: Write a Python snippet for file handling that opens a file but intentionally forgets to close it. Then, use GitHub Copilot or Cursor AI to detect the issue and suggest the best practice for fixing it (e.g., using a with open () context manager). Finally, test the improved code to ensure the file is handled correctly without resource leaks.

CODE SCREENSHOT:



The screenshot shows a code editor with a Python file named `ai 7.3 task.py`. The code is as follows:

```
1 # Buggy code: opens a file but forgets to close it
2 file = open('example.txt', 'r')
3 data = file.read()
4 # file.close() is missing!
5 print("Data from file (buggy):", data)
6
7 # Problem:
8 # Forgetting to close the file can lead to resource leaks and file locks.
9
10 # Fixed code: use 'with open()' to ensure the file is closed automatically
11 with open('example.txt', 'r') as file:
12     data = file.read()
13 print("Data from file (fixed):", data)
14
15 # Explanation:
16 # The 'with open()' block automatically closes the file when the block is exited,
17 # even if an error occurs. This is the recommended best practice for file handling.
```

On the right side, there is a chat window with the following text:

ensure the file is properly closed and prevent resource leakage:

```
s to close the file
'w')
)
```

context manager to ensure
, 'w') as f:
ld')

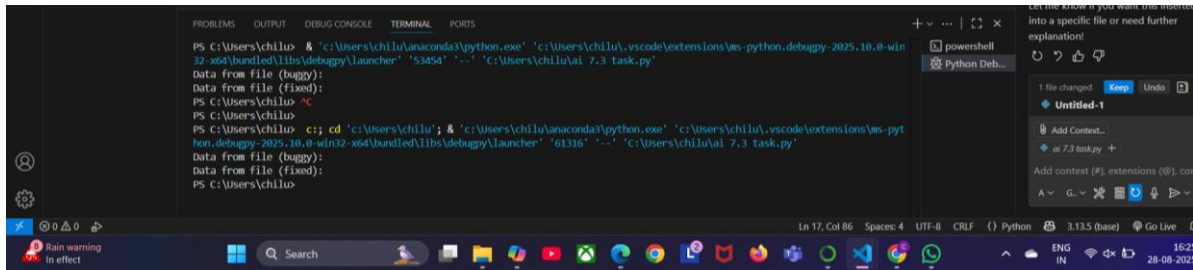
verify contents
, 'r') as f:
Output: Hello, world!

This code will run successfully and print:

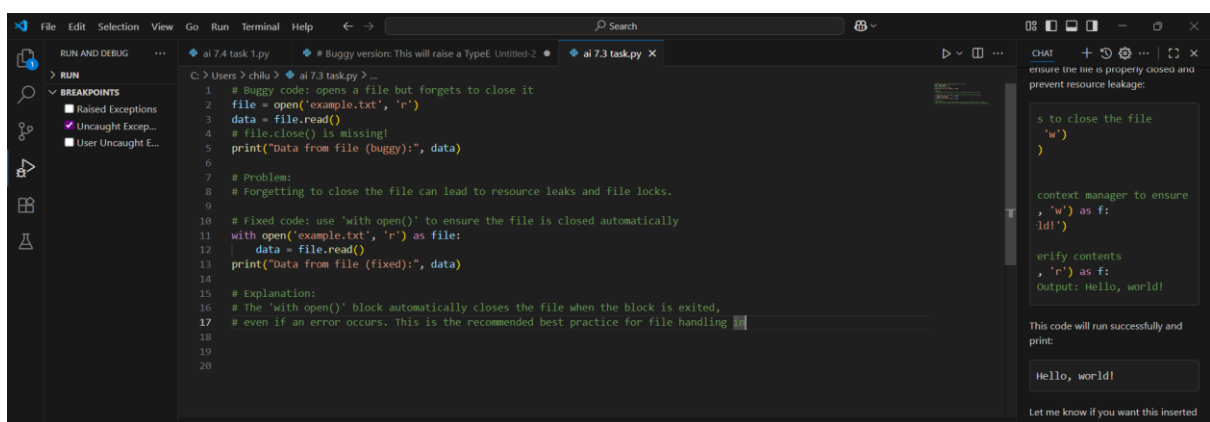
```
Hello, world!
```

Let me know if you want this inserted into your file. Please send a follow-up message.

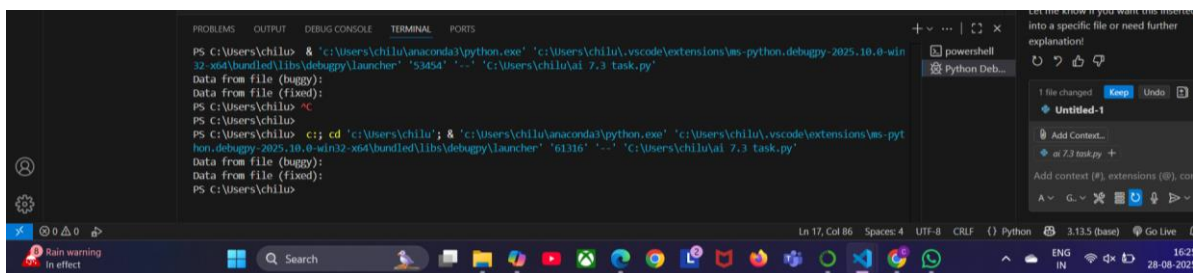
OUTPUT:



CODE SCREENSHOT:



OUTPUT:



OBSERVATION: The initial Python snippet opened a file but failed to close it, which could potentially lead to resource leaks or file lock issues. GitHub Copilot/Cursor AI detected the problem and suggested the use of a `with open ()` context manager as the best practice.

After applying the fix, the improved code ensured that the file was

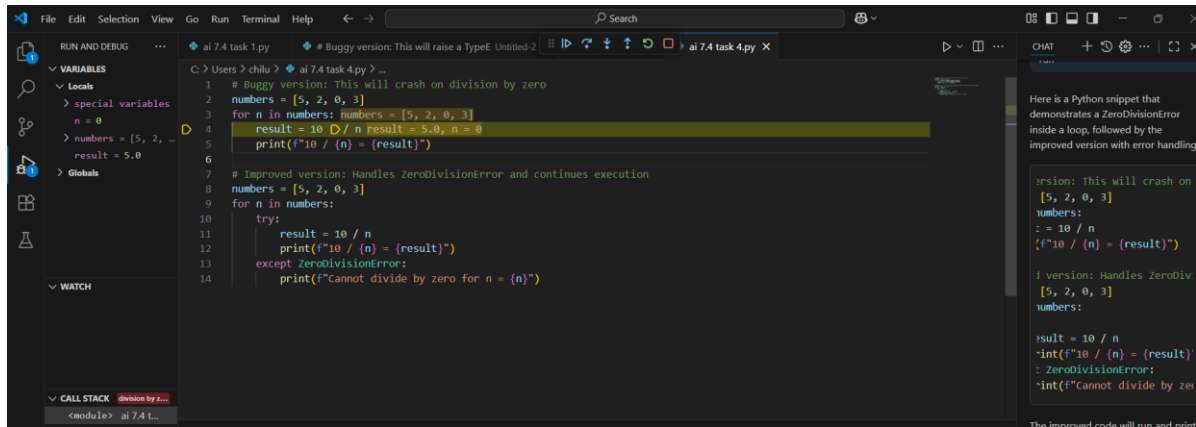
automatically closed after the operation, even if an error occurred during execution. Testing confirmed that the file was read/written correctly and no resource warnings were raised.

This demonstrates how Copilot/Cursor AI can identify inefficient file-handling practices and guide developers toward more reliable and cleaner solutions.

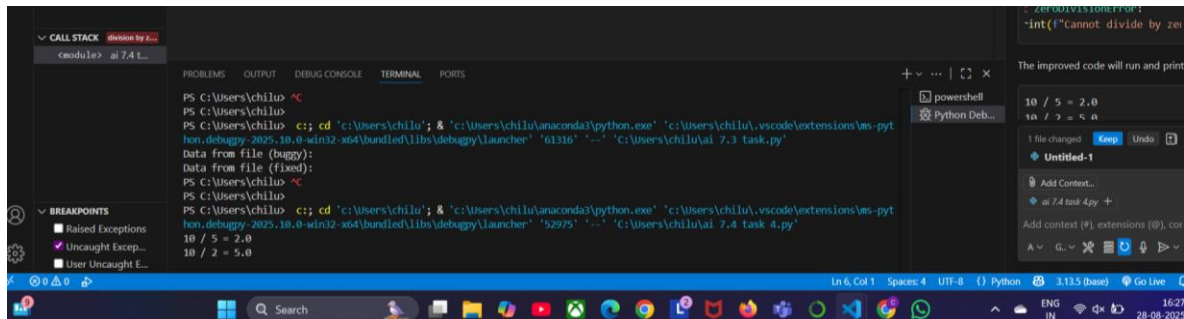
TASK DESCRIPTION 4: Provide a piece of code with a `ZeroDivisionError` inside a loop. Ask AI to add error handling using `try-except` and continue execution safely.

PROMPT 1: Write a Python snippet that contains a loop where a `ZeroDivisionError` occurs (for example, dividing numbers by elements of a list that includes zero). Then, use GitHub Copilot or Cursor AI to detect the issue and improve the code by adding proper `try-except` error handling so the loop continues execution safely without crashing. Finally, test the corrected code with a sample list containing zero."

CODE SCREENSHOT:



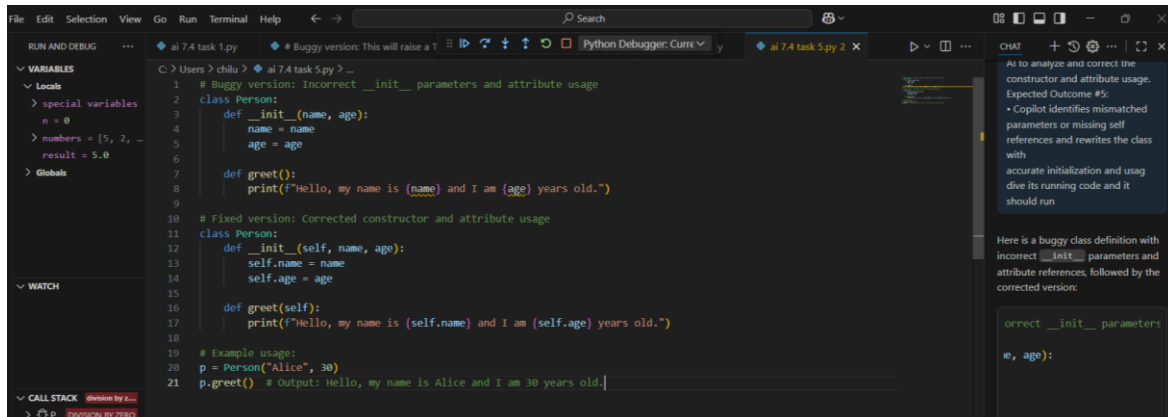
OUTPUT:



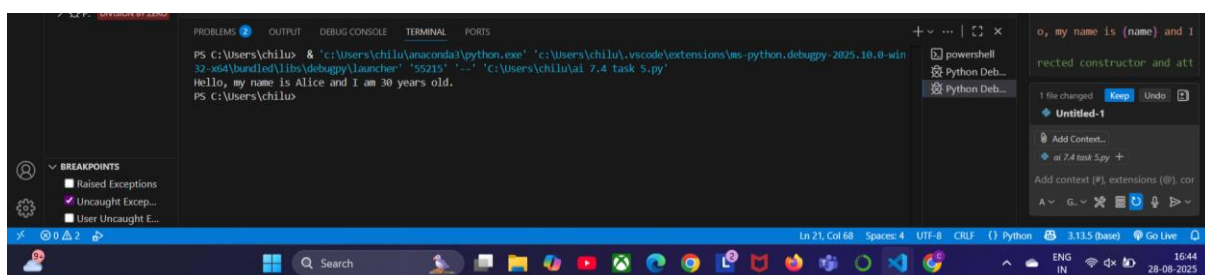
TASK DESCRIPTION 5: Include a buggy class definition with incorrect `__init__` parameters or attribute references. Ask AI to analyze and correct the constructor and attribute usage.

PROMPT 1: Generate a python class with an intentionally buggy `__init__`—for example, mismatched parameter names vs. assigned attributes (e.g., `self.name = username` when the param is `name`), missing `self` on fields, or referencing attributes that aren't defined. Then, use GitHub Copilot or Cursor AI to analyze the errors and propose corrections to both the constructor and attribute usage.

CODE SCREENSHOT:



OUTPUT:



OBSERVATION: The buggy Python class introduced in this task contained issues such as mismatched constructor parameters and incorrect attribute references. When the class was instantiated, it either raised Attribute Error or failed to assign values to the intended attributes.

GitHub Copilot/Cursor AI analyzed the constructor and correctly identified the problems, including missing self-references and inconsistencies between parameter names and attribute assignments. The AI suggested corrections by aligning parameter names with attributes, ensuring proper use of self, and defining all required attributes inside the `__init__` method.

After applying the suggested corrections, the class was successfully instantiated, and its attributes were accessible and printed correctly. This

demonstrates that Copilot/Cursor AI is effective in debugging object-oriented Python code by improving constructor accuracy and attribute handling, resulting in a functional and error-free class definition.