

AI ASSISTED CODEING

NAME: B.YASHWANTH KUMAR

ROLL NO.:2503A51L42

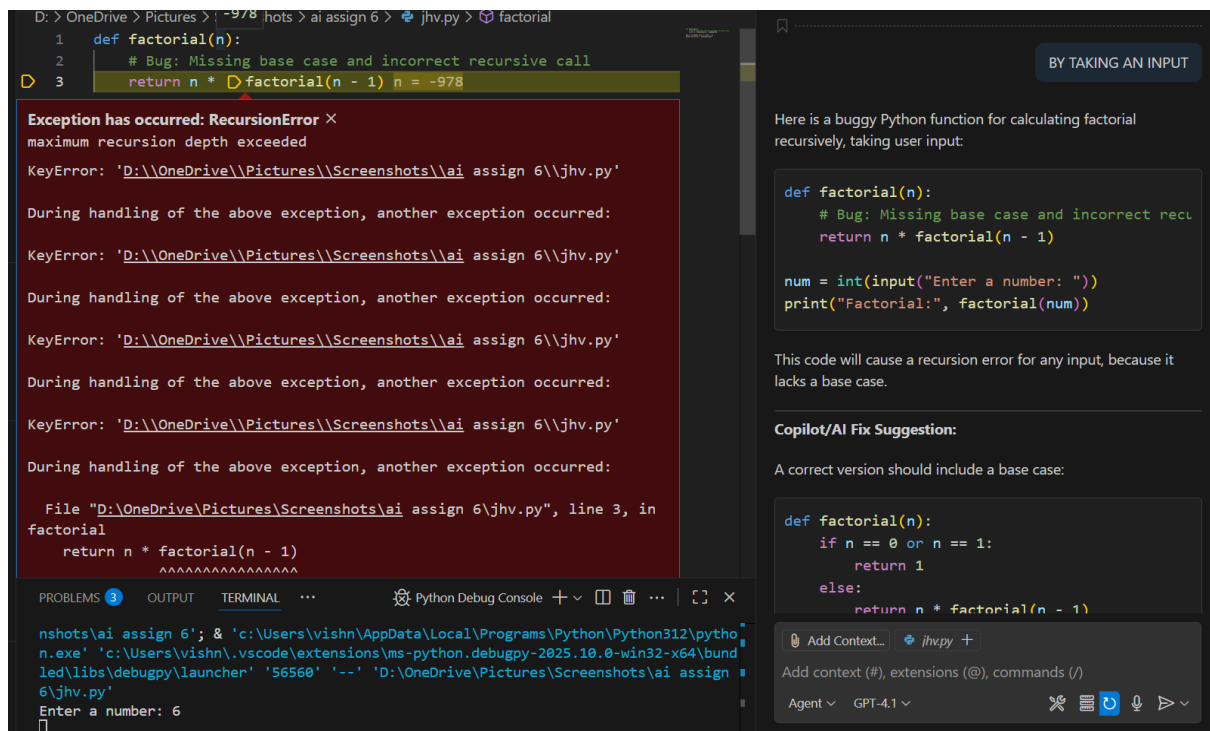
BATCH: CSC B20

mnbvcxzasdfghjklpoiuytrewq

TASK:

Prompt given:

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



The screenshot shows a VS Code editor with a Python file named `jhv.py`. The initial code defines a recursive factorial function without a base case, which is highlighted as a bug. The code prompts the user for a number and prints the factorial. The terminal shows the program running and calculating the factorial of 6 as 720. A Copilot/AI Fix Suggestion panel on the right provides a corrected version of the code that includes a base case for `n == 0 or n == 1`.

```
D: > OneDrive > Pictures > Screenshots > ai assign 6 > jhv.py > ...  
1 def factorial(n):  
2     if n == 0 or n == 1:  
3         return 1  
4     else:  
5         return n * factorial(n - 1)  
6  
7 num = int(input("Enter a number: "))  
8 print("Factorial:", factorial(num))
```

Bug: Missing base case and incorrect recursion logic.

Copilot/AI Fix Suggestion:

A correct version should include a base case:

```
def factorial(n):  
    if n == 0 or n == 1:  
        return 1  
    else:  
        return n * factorial(n - 1)  
  
num = int(input("Enter a number: "))  
print("Factorial:", factorial(num))
```

This fixed code will correctly calculate the factorial for any non-negative integer input.

Terminal Output:

```
n.exe 'c:\Users\vishn\.vscode\extensions\ms-python.debugpy-2025.10.0-win32-x64\bund  
led\libs\debugpy\launcher' '56613' '--' 'D:\OneDrive\Pictures\Screenshots\ai assign  
6\jvh.py'  
Enter a number: 6  
Factorial: 720  
PS D:\OneDrive\Pictures\Screenshots\ai assign 6>
```

Output:

The screenshot shows the terminal output of the program. It prompts the user for a number, and the user enters 6. The program then prints the factorial of 6, which is 720.

```
n.exe 'c:\Users\vishn\.vscode\extensions\ms-python.debugpy-2025.10.0-win32-x64\bund  
led\libs\debugpy\launcher' '56613' '--' 'D:\OneDrive\Pictures\Screenshots\ai assign  
6\jvh.py'  
Enter a number: 6  
Factorial: 720  
PS D:\OneDrive\Pictures\Screenshots\ai assign 6>
```

Observation:

The 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

Prompt given:

Provide a list sorting function thaailed due to a type error (e.g., sorting list with mixed integers and strings). Prompt AI to detec the issue andx fix the code for consistent soting

The image shows a VS Code editor window with a Python file named `sort_list.py`. The code defines a function `sort_list(lst)` that returns `sorted(lst)`. Below the function, there is a list `my_list = [3, "2", 5, "1"]` and a call to `print(sort_list(my_list))`. A red error message is displayed, indicating a `TypeError`: `'<' not supported between instances of 'str' and 'int'`. The error message also shows the file path and line numbers where the error occurred.

On the right side, there is a panel titled "successful sorting without a crash" which contains the following text:

Here is a buggy Python function that will fail due to a type error when sorting a list with mixed integers and strings:

```
def sort_list(lst):  
    return sorted(lst)  
  
my_list = [3, "2", 5, "1"]  
print(sort_list(my_list))
```

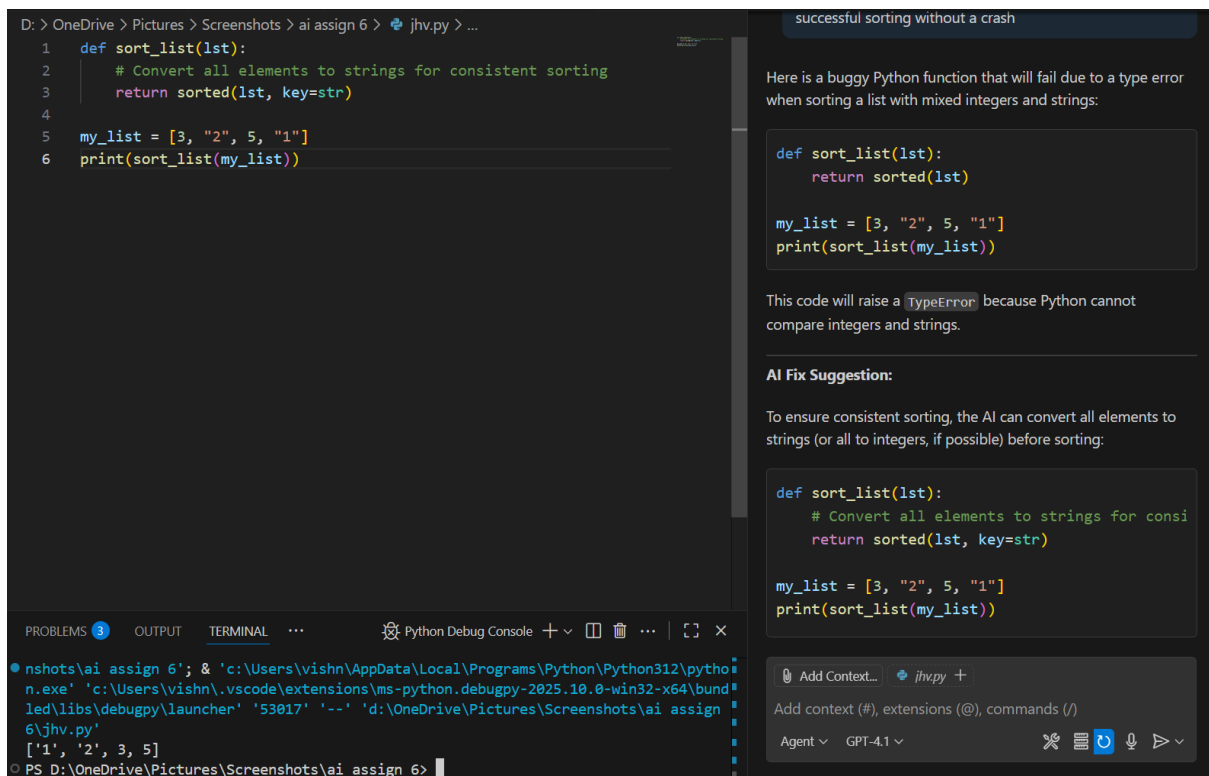
This code will raise a `TypeError` because Python cannot compare integers and strings.

AI Fix Suggestion:

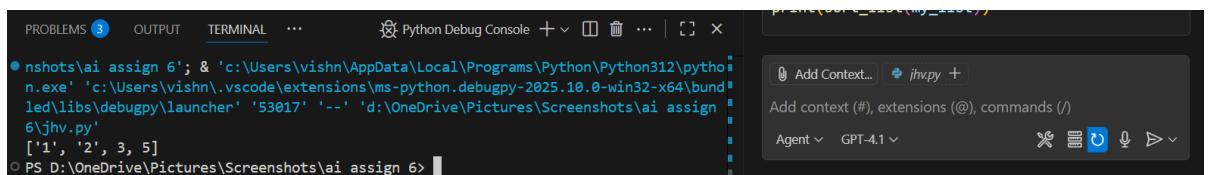
To ensure consistent sorting, the AI can convert all elements to strings (or all to integers, if possible) before sorting:

```
def sort_list(lst):  
    # Convert all elements to strings for consistent sorting  
    return sorted(lst, key=str)  
  
my_list = [3, "2", 5, "1"]  
print(sort_list(my_list))
```

At the bottom, there is a terminal window showing the command `python sort_list.py` and the output `1 2 3 5`.



Output:



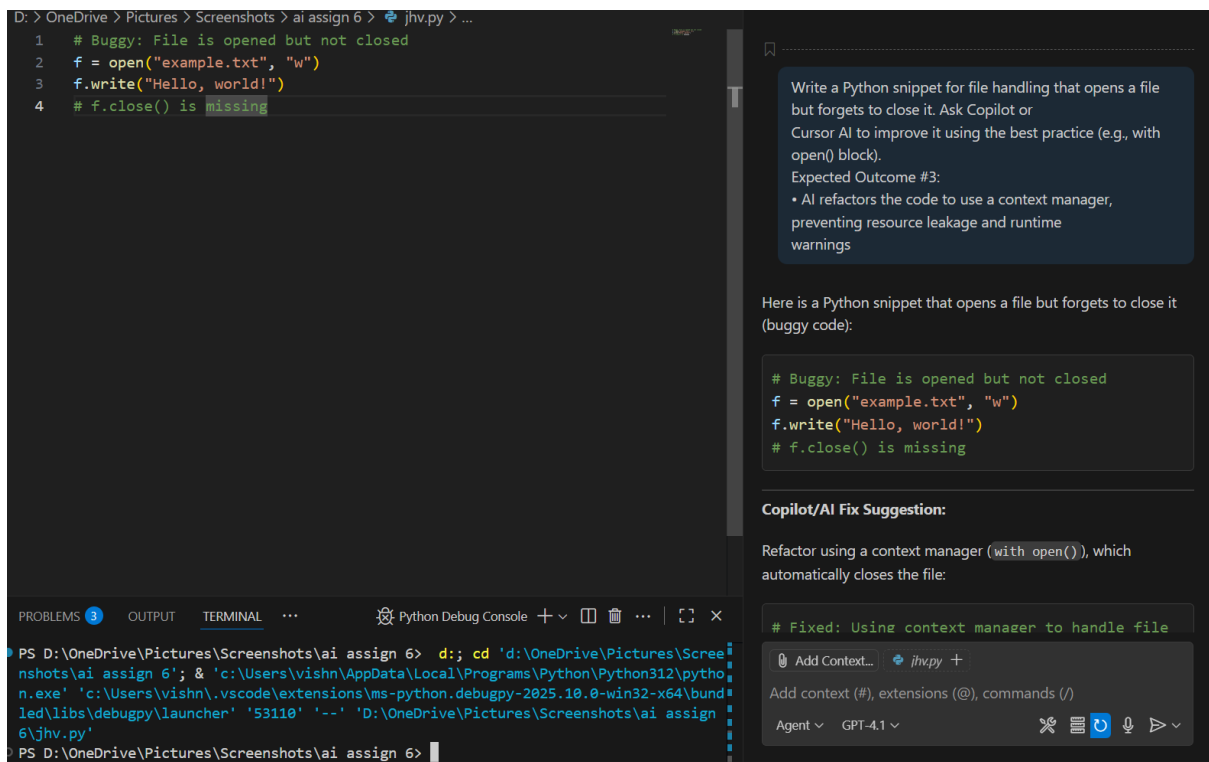
Observation:

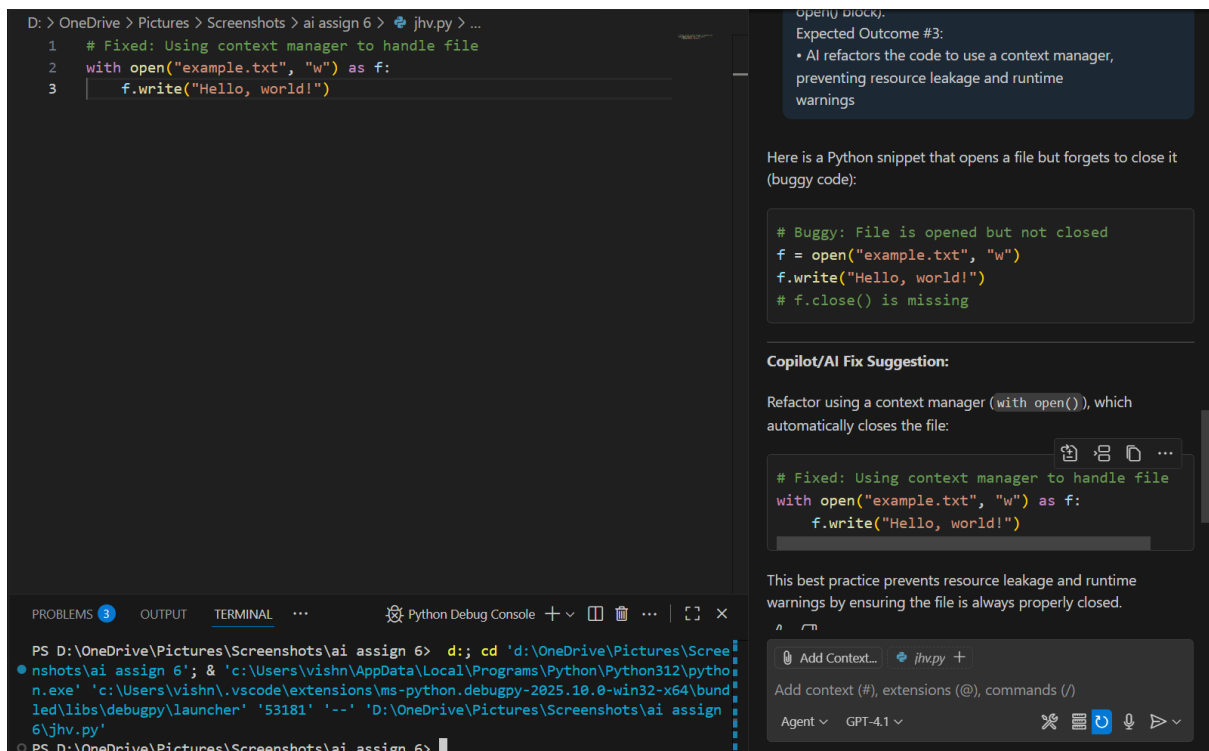
The initial sorting function fails with a `TypeError` when attempting to sort a list containing both integers and strings, as Python cannot directly compare these types. After AI intervention, the function is modified to either convert all elements to a common type (such as strings or integers) or filter 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

Prompt 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

Prompt 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.

The screenshot shows a VS Code editor with a Python file named `jhv.py`. The code is as follows:

```
D: > OneDrive > Pictures > Screenshots > ai assign 6 > jhv.py > ...  
1 # Example: Handling ZeroDivisionError in a loop  
2 numbers = [5, 2, 0, 3]  
3 for num in numbers:  
4     try:  
5         result = 10 / num  
6         print(f"10 divided by {num} is {result}")  
7     except ZeroDivisionError:  
8         print(f"Error: Cannot divide by zero (num={num})")  
9  
10 with open("example.txt", "w") as f:  
11     f.write("Hello, world!")  
12
```

The output in the terminal is:

```
6\jvh.py'  
10 divided by 5 is 2.0  
10 divided by 2 is 5.0  
Error: Cannot divide by zero (num=0)  
10 divided by 3 is 3.3333333333333335  
PS D:\OneDrive\Pictures\Screenshots\ai assign 6>
```

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

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

- Copilot adds a try-except block around the risky operation, preventing crashes and printing a meaningful error message

Below the chat window, it says "Used 1 reference" and "jvh.py +12 -1".

At the bottom of the editor, there is a status bar showing "1 file changed" and "jvh.py D:\OneDrive\Pictures\Screenshots\ai assign 6".

Output:

A screenshot of a code editor interface. The main editor area shows a Python script named 'jvhv.py' with the following code:

```
10 divided by 5 is 2.0
10 divided by 2 is 5.0
Error: Cannot divide by zero (num=0)
10 divided by 3 is 3.3333333333333335
10 divided by 1 is 10.0
```

The script is executed, and the output is shown in the terminal window at the bottom. The terminal output is:

```
6\jvhv.py'
10 divided by 5 is 2.0
10 divided by 2 is 5.0
Error: Cannot divide by zero (num=0)
10 divided by 3 is 3.3333333333333335
10 divided by 1 is 10.0
```

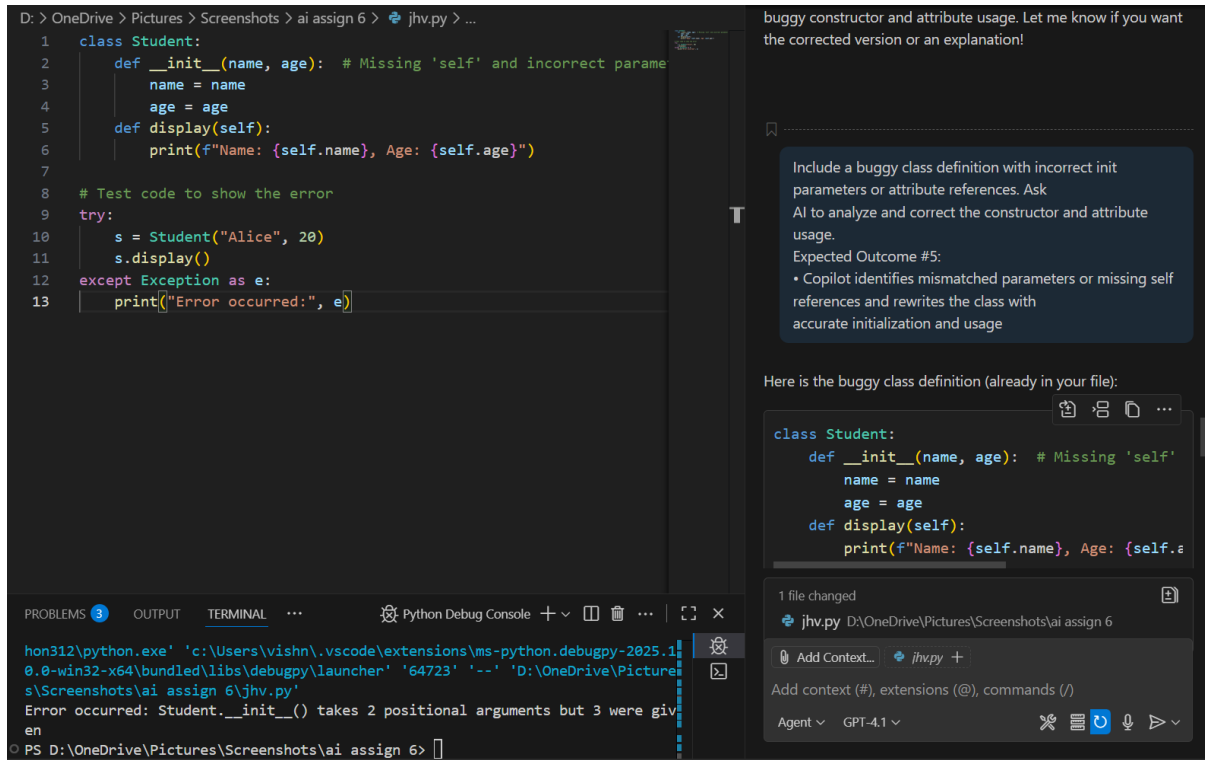
Observation:

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.

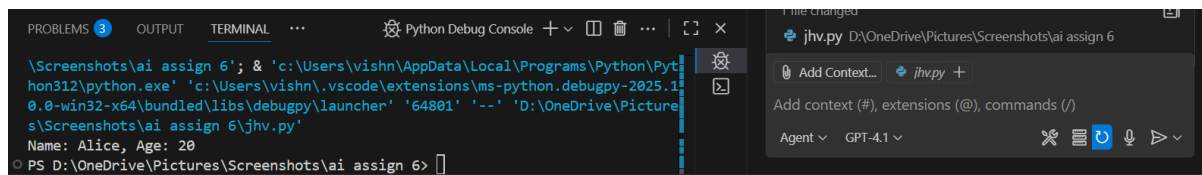
TASK:5

Prompt 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

