AI ASSISTED CODING

Lab Assignment-7.4

Name : T.Shylasri

H.T.NO:2303A51876

Batch-14(LAB-6)

Date:05-02-2026

ASSIGNMENT-7.4

Task 1: Debugging a Recursive Calculation Module

Scenario

You are maintaining a utility module in a software project that performs mathematical computations. One function is meant to calculate the factorial of a number, but users are reporting crashes or incorrect outputs.

Task Description

You are given a Python function intended to calculate the factorial of a number using recursion, but it contains logical or syntactical errors (such as a missing base condition or incorrect recursive call).

Use GitHub Copilot or Cursor AI to:

• Analyze the faulty code

• Identify the exact cause of the error

• Suggest and apply corrections to make the function work correctly

Document how the AI detected the issue and what changes were made.

Expected Outcome

• A corrected recursive factorial function

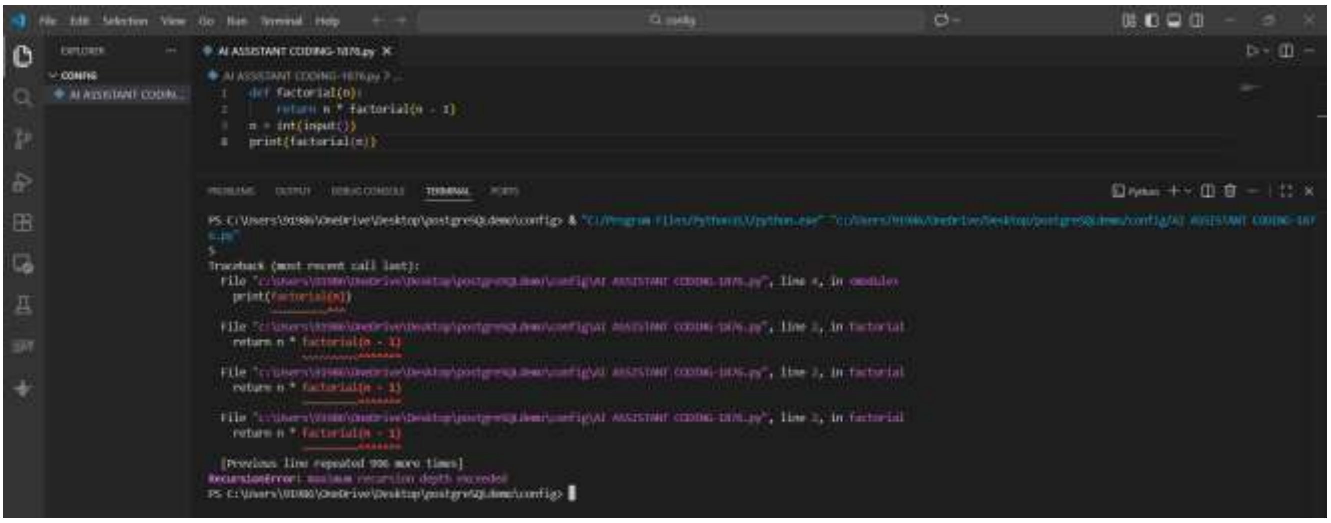
• AI-generated explanation identifying:

o The missing or incorrect base case

o The corrected recursive logic

• Sample input/output demonstrating correct execution

Error Code



```python
def factorial(n):
    return n * factorial(n - 1)
n = int(input())
print(factorial(n))
```

```
PS C:\Users\91986\OneDrive\Desktop\postgreSQLdemo\config> & "C:/Program Files/Python311/python.exe" "c:/Users/91986/OneDrive/Desktop/postgreSQLdemo/config/AI ASSISTANT CODING-1876.py"
5
Traceback (most recent call last):
  File "c:\Users\91986\OneDrive\Desktop\postgreSQLdemo\config\AI ASSISTANT CODING-1876.py", line 4, in <module>
    print(factorial(n))
  File "c:\Users\91986\OneDrive\Desktop\postgreSQLdemo\config\AI ASSISTANT CODING-1876.py", line 2, in factorial
    return n * factorial(n - 1)
  File "c:\Users\91986\OneDrive\Desktop\postgreSQLdemo\config\AI ASSISTANT CODING-1876.py", line 2, in factorial
    return n * factorial(n - 1)
  File "c:\Users\91986\OneDrive\Desktop\postgreSQLdemo\config\AI ASSISTANT CODING-1876.py", line 2, in factorial
    return n * factorial(n - 1)
  [Previous line repeated 996 more times]
RecursionError: maximum recursion depth exceeded
PS C:\Users\91986\OneDrive\Desktop\postgreSQLdemo\config>
```

Error Explanation (Identified by AI)

The above code causes a runtime error due to infinite recursion.
The function factorial() calls itself repeatedly without a base condition to stop the recursion.

Explanation of the Corrected Code

- A **base case** is added for n == 0 and n == 1
- This prevents infinite recursive calls
- Each recursive call reduces n by 1, ensuring termination
- The factorial value is calculated correctly using recursion

AI Analysis and Error Identification

After analyzing the code, the AI identified the following issues:

- The function does not contain a base case
- Recursive calls continue indefinitely
- This results in a RecursionError (maximum recursion depth exceeded)

Exact Cause of the Error

- Recursive functions must have a **termination condition**
- Without a base case, the function keeps calling itself
- Factorial computation requires stopping when n == 0 or n == 1

AI-Suggested Corrections

AI suggested:

- Adding a proper **base condition**
- Ensuring the recursive call reduces the problem size

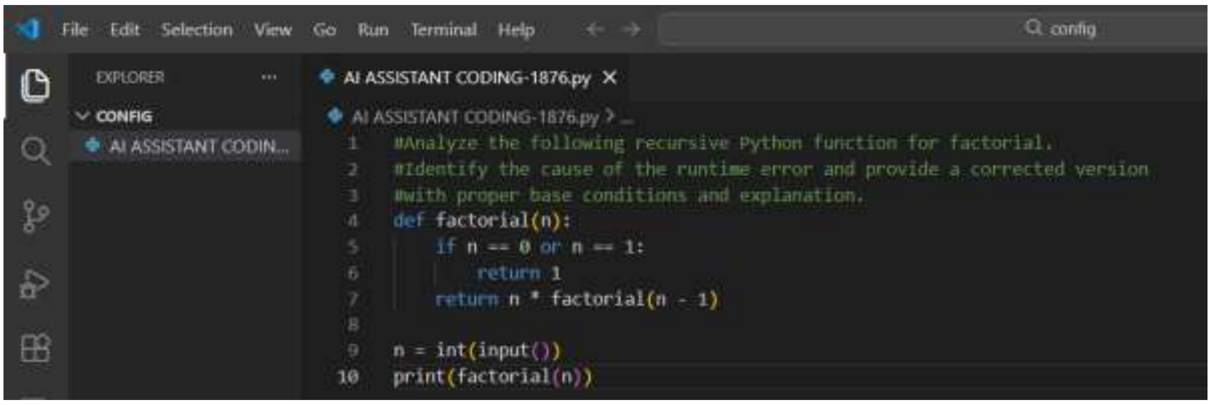
Prompt:

#Analyze the following recursive Python function for factorial.

#Identify the cause of the runtime error and provide a corrected version

#with proper base conditions and explanation.

Corrected Code:

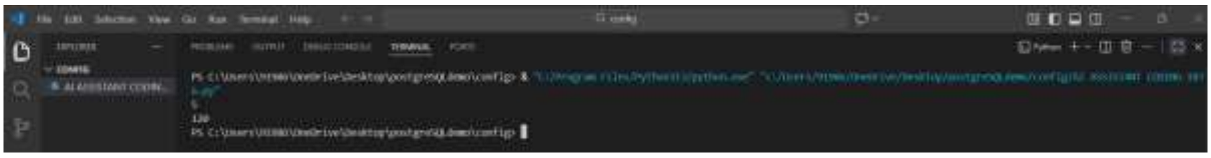


File Edit Selection View Go Run Terminal Help
config
EXPLORER
CONFIG
AI ASSISTANT CODIN...
AI ASSISTANT CODING-1876.py
AI ASSISTANT CODING-1876.py > ...

```python
#Analyze the following recursive Python function for factorial.
#Identify the cause of the runtime error and provide a corrected version
#with proper base conditions and explanation.
def factorial(n):
    if n == 0 or n == 1:
        return 1
    return n * factorial(n - 1)

n = int(input())
print(factorial(n))
```

Output:



```
PS C:\Users\[illegible]\OneDrive\Desktop\postgreSQL demo\config> & [illegible]
[illegible]
5
120
PS C:\Users\[illegible]\OneDrive\Desktop\postgreSQL demo\config>
```

Justification

This correction is necessary because:

- Recursive functions must always have a **base condition**
- Without a base case, recursion never stops
- The corrected logic follows the mathematical definition of factorial
- AI tools effectively identified the missing base case and suggested a reliable fix

Conclusion

Using AI tools like GitHub Copilot or Cursor AI helps in quickly identifying recursion errors, understanding their cause, and applying correct fixes. This improves debugging efficiency and code reliability.

Task 2: Fixing Data Type Errors in a Sorting Utility

Scenario

You are developing a data processing script that sorts user input values.

The program crashes when users enter mixed data types.

Task Description

You are provided with a list-sorting function that fails due to a

TypeError caused by mixed data types (e.g., integers and strings).

Use GitHub Copilot or Cursor AI to:

• Detect the root cause of the runtime error

• Modify the code to ensure consistent sorting (by filtering or type

conversion)

• Prevent the program from crashing

Explain the debugging steps followed by the AI.

Expected Outcome

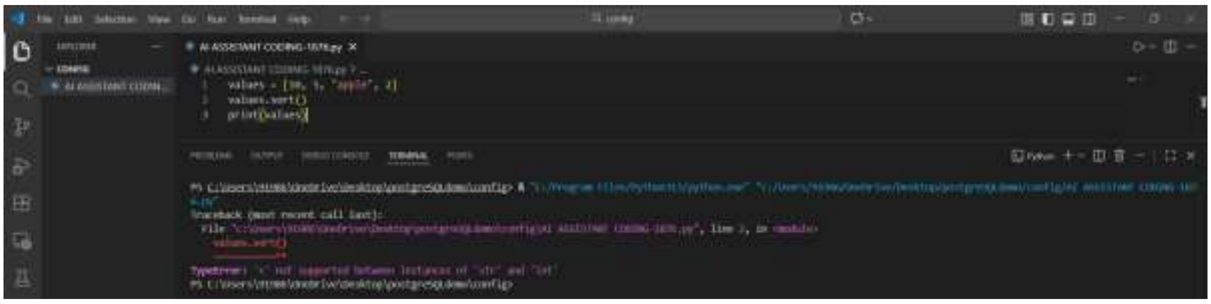
• A corrected sorting function

• AI-generated solution handling type inconsistencies

• Successful sorting without runtime errors

• Explanation of how the fix improves robustness

Error-Generated Code:



```
values = [10, 5, "apple", 2]
values.sort()
print(values)
```

```
PS [illegible]> & [illegible]
Traceback (most recent call last):
  File "[illegible]", line 2, in <module>
    values.sort()
TypeError: '<' not supported between instances of 'str' and 'int'
PS [illegible]>
```

Explanation of the Error (Root Cause)

- The list contains mixed data types (int and str)
- Python cannot compare integers and strings during sorting
- Sorting requires all elements to be of compatible data types
- This causes a runtime TypeError

AI Debugging Steps Followed

1. AI detected a **TypeError during sorting**
2. Identified mixed data types as the cause
3. Suggested separating values based on type
4. Recommended sorting each type independently
5. Prevented invalid comparisons

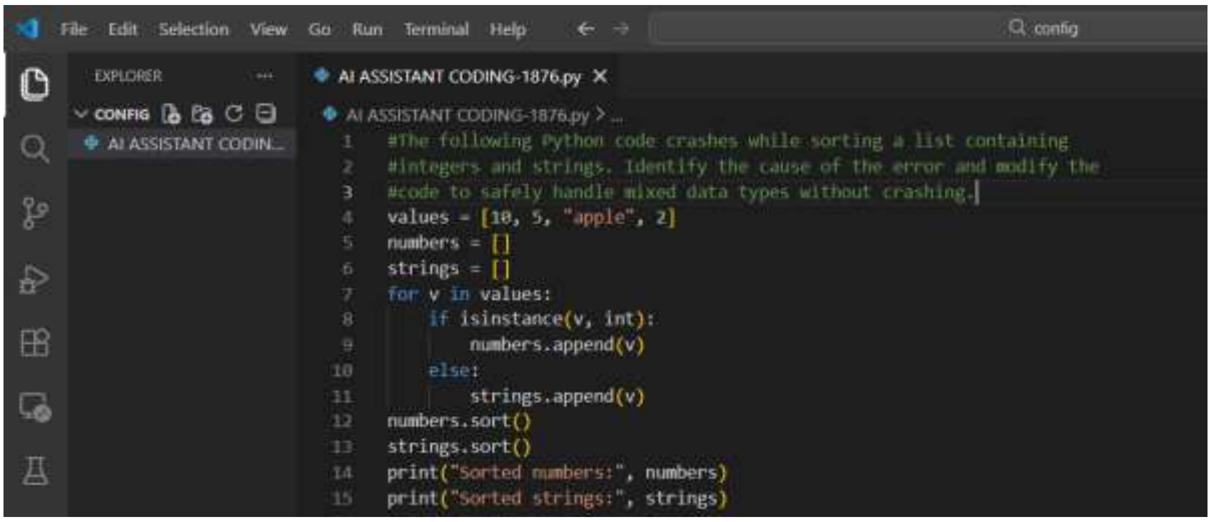
Prompt:

#The following Python code crashes while sorting a list containing

#integers and strings. Identify the cause of the error and modify the

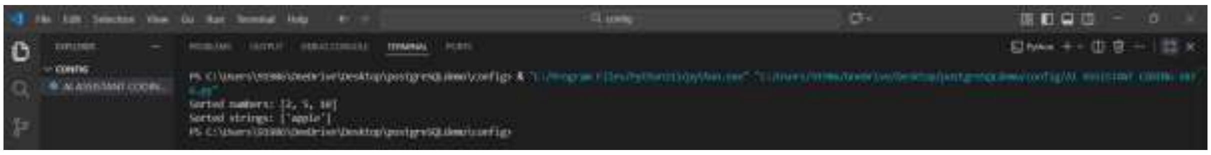
#code to safely handle mixed data types without crashing.

Corrected Code:



```python
#The following Python code crashes while sorting a list containing
#integers and strings. Identify the cause of the error and modify the
#code to safely handle mixed data types without crashing.
values = [10, 5, "apple", 2]
numbers = []
strings = []
for v in values:
    if isinstance(v, int):
        numbers.append(v)
    else:
        strings.append(v)
numbers.sort()
strings.sort()
print("Sorted numbers:", numbers)
print("Sorted strings:", strings)
```

Output:



```
PS C:\Users\[illegible]\OneDrive\Desktop\postgreSQLdemo\config> & [illegible]
Sorted numbers: [2, 5, 10]
Sorted strings: ['apple']
PS C:\Users\[illegible]\OneDrive\Desktop\postgreSQLdemo\config>
```

Justification:

- Prevents runtime crashes caused by invalid comparisons
- Handles mixed data types safely
- Ensures consistent and predictable sorting
- Improves program robustness for real-world user input
- AI-assisted debugging helps identify and resolve errors efficiently

Task 3: Improving File Handling Reliability

Scenario

A backend script reads data from files regularly. Over time, the system shows performance issues due to improper resource management.

Task Description

You are given a Python file-handling snippet that opens a file but does not explicitly close it.

Use GitHub Copilot or Cursor AI to:

• Identify the potential problem in the code

• Refactor it using best practices (such as a context manager)

• Ensure safe and reliable file handling

Briefly describe why the revised approach is better.

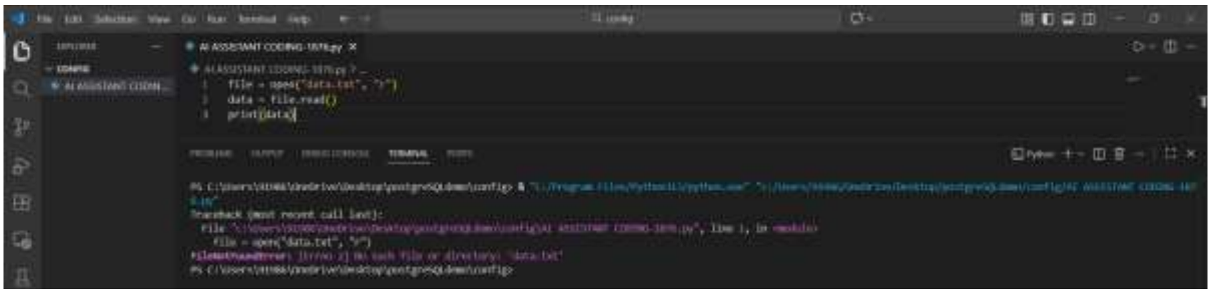
Expected Outcome

• Refactored code using the with open() statement

• AI explanation highlighting prevention of resource leaks

• Clean execution without warnings or errors

Error-Prone Code (Improper File Handling):



```
file = open("data.txt", "r")
data = file.read()
print(data)
```

```
PS C:\Users\[illegible]\OneDrive\Desktop\postgreSQLdemo\config> & "C:/Program Files/Python311/python.exe" "c:/Users/[illegible]/OneDrive/Desktop/postgreSQLdemo/config/AI ASSISTANT CODING-1876.py"
Traceback (most recent call last):
  File "c:\Users\[illegible]\OneDrive\Desktop\postgreSQLdemo\config\AI ASSISTANT CODING-1876.py", line 1, in <module>
    file = open("data.txt", "r")
FileNotFoundError: [Errno 2] No such file or directory: 'data.txt'
PS C:\Users\[illegible]\OneDrive\Desktop\postgreSQLdemo\config>
```

Problem Identified by AI

- The file is opened but **never explicitly closed**
- Open file descriptors accumulate over time
- This can cause:
  - Memory leaks
  - File locking issues
  - Performance degradation
  - Resource exhaustion warnings

AI Debugging & Explanation

AI identified that:

- The file handle remains open if an exception occurs
- Manual closing is error-prone

- A **context manager (with open)** automatically handles cleanup

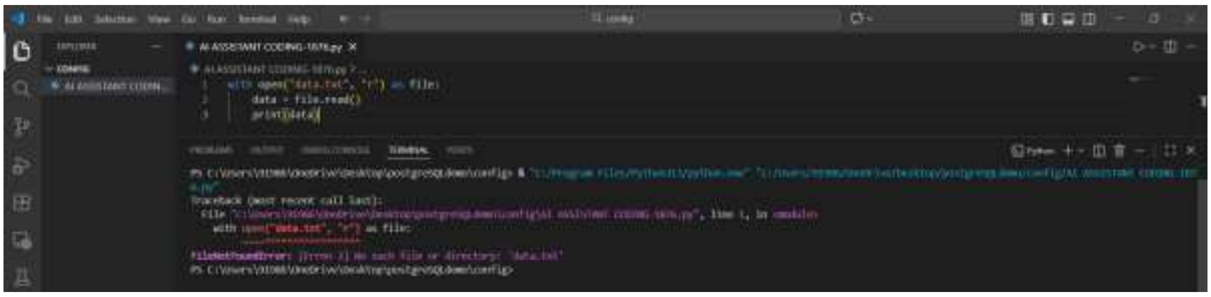
Prompt:

#The following Python code opens a file but does not close it.

#Identify the potential problem and refactor the code using best

#practices to ensure safe and reliable file handling.

Refactored Code (Best Practice Solution)



```
with open("data.txt", "r") as file:
    data = file.read()
    print(data)
```

```
PS C:\Users\91986\OneDrive\Desktop\postgreSQLdemo\config> & "C:/Program Files/Python311/python.exe" [illegible]
Traceback (most recent call last):
  File "[illegible]", line 1, in <module>
    with open("data.txt", "r") as file:
FileNotFoundError: [Errno 2] No such file or directory: 'data.txt'
PS C:\Users\91986\OneDrive\Desktop\postgreSQLdemo\config>
```

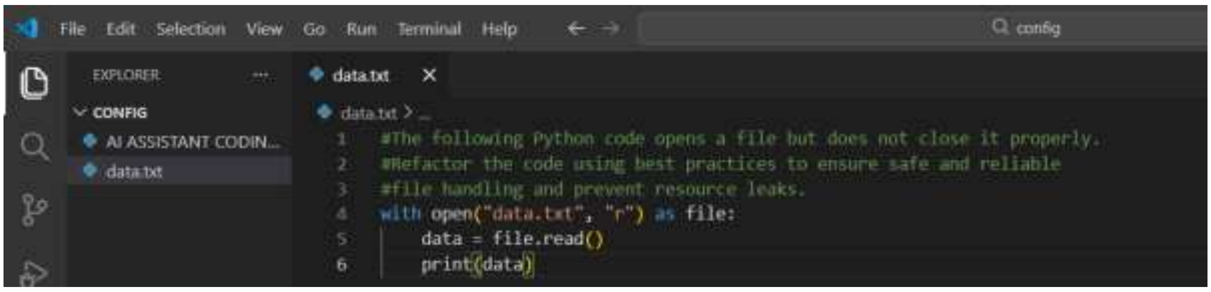
Prompt:

#The following Python code opens a file but does not close it properly.

#Refactor the code using best practices to ensure safe and reliable

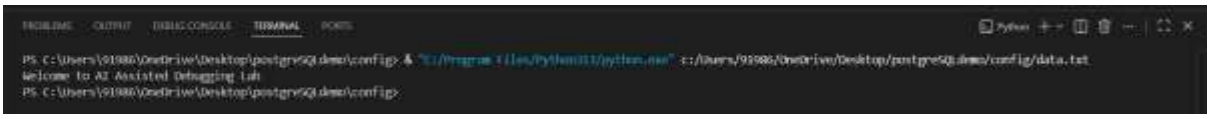
#file handling and prevent resource leaks.

Corrected Code:



```
#The following Python code opens a file but does not close it properly.
#Refactor the code using best practices to ensure safe and reliable
#file handling and prevent resource leaks.
with open("data.txt", "r") as file:
    data = file.read()
    print(data)
```

Output:



```
PS C:\Users\91986\OneDrive\Desktop\postgreSQLdemo\config> & "C:/Program Files/Python311/python.exe" c:/Users/91986/OneDrive/Desktop/postgreSQLdemo/config/data.txt
Welcome to AI Assisted Debugging Lab
PS C:\Users\91986\OneDrive\Desktop\postgreSQLdemo\config>
```

Justification:

- Automatically closes the file after use
- Prevents resource leaks
- Handles exceptions safely

- Improves reliability and performance
- Follows Python best practices
- Suitable for long-running backend scripts

Task 4: Handling Runtime Errors Gracefully in Loops

Scenario

You are working on a data analysis script that processes a list of values. Some values cause runtime errors, but the program should continue processing remaining data.

Task Description

You are provided with a code snippet containing a ZeroDivisionError inside a loop.

Use GitHub Copilot or Cursor AI to:

• Detect the exact location of the error

• Add appropriate exception handling using try-except

• Ensure the loop continues executing safely

Document how AI improved the fault tolerance of the program.

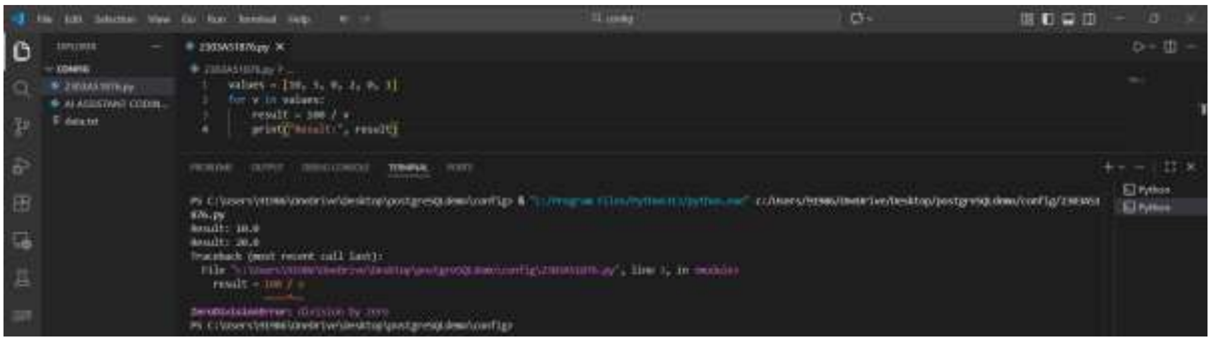
Expected Outcome

• Updated code with proper exception handling

• Meaningful error messages instead of program crashes

• Successful execution for all valid inputs

Error Code:



```python
values = [10, 5, 0, 2, 0, 3]
for v in values:
    result = 100 / v
    print("Result:", result)
```

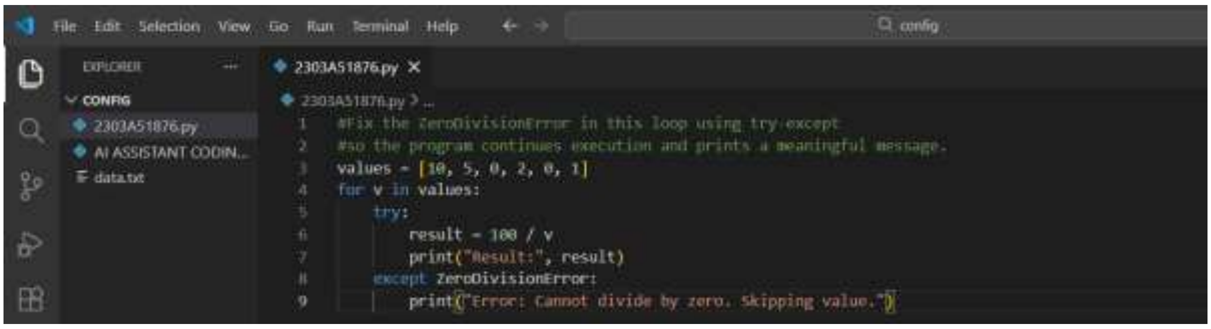
```
PS C:\Users\[illegible]\OneDrive\Desktop\postgresql.demo\config> & "C:/Program Files/Python313/python.exe" c:/Users/[illegible]/OneDrive/Desktop/postgresql.demo/config/2303A51
876.py
Result: 10.0
Result: 20.0
Traceback (most recent call last):
  File "c:\Users\[illegible]\OneDrive\Desktop\postgresql.demo\config\2303A51876.py", line 3, in <module>
    result = 100 / v
             ~~~~^~~
ZeroDivisionError: division by zero
PS C:\Users\[illegible]\OneDrive\Desktop\postgresql.demo\config>
```

Prompt:

#Fix the ZeroDivisionError in this loop using try-except

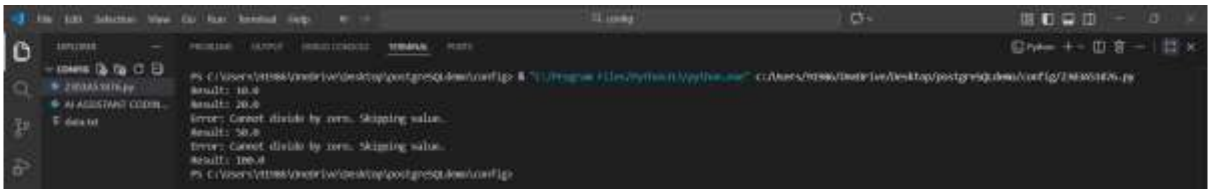
#so the program continues execution and prints a meaningful message.

Corrected Code:



```python
#Fix the ZeroDivisionError in this loop using try-except
#so the program continues execution and prints a meaningful message.
values = [10, 5, 0, 2, 0, 1]
for v in values:
    try:
        result = 100 / v
        print("Result:", result)
    except ZeroDivisionError:
        print("Error: Cannot divide by zero. Skipping value.")
```

Output:



```
PS C:\Users\[illegible]\OneDrive\Desktop\postgresql-demo\config> & "C:/Program Files/Python313/python.exe" c:/Users/[illegible]/OneDrive/Desktop/postgresql-demo/config/2303A51876.py
Result: 10.0
Result: 20.0
Error: Cannot divide by zero. Skipping value.
Result: 50.0
Error: Cannot divide by zero. Skipping value.
Result: 100.0
PS C:\Users\[illegible]\OneDrive\Desktop\postgresql-demo\config>
```

Observation

When the program was executed without exception handling, it terminated with a ZeroDivisionError. After adding the try–except block, the program handled the error gracefully by displaying an error message and continued processing the remaining values without interruption.

Justification

In data processing applications, input values may be unpredictable and can cause runtime errors such as division by zero. Without proper error handling, a single invalid value can terminate the entire program. Therefore, exception handling using try–except is necessary to make the program fault-tolerant and ensure continuous execution.

Conclusion

Exception handling significantly improves the reliability and robustness of programs. By using try–except, runtime errors such as division by zero can be managed effectively, preventing program crashes and allowing safe execution for all valid inputs. AI tools helped quickly identify the error and implement an efficient solution.

Task 5: Debugging Class Initialization Errors

Scenario

A class written by a junior developer is throwing unexpected errors when

objects are created or attributes are accessed.

Task Description

You are given a Python class with:

• Incorrect __init__ parameters

• Missing or incorrect attribute references (e.g., missing self)

Use GitHub Copilot or Cursor AI to:

• Analyze the class definition

• Identify constructor and attribute issues

• Correct the class so objects initialize and behave correctly

Explain the corrections suggested by the AI.

Expected Outcome

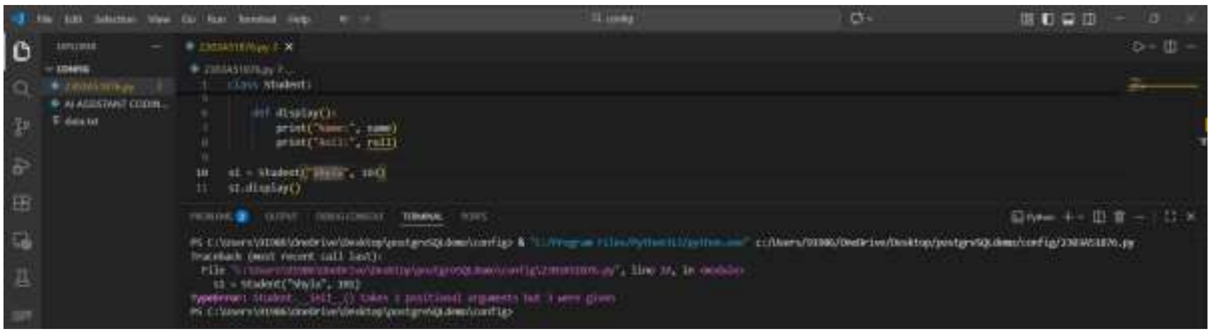
• A corrected class definition

• Proper use of self and constructor parameters

• AI-assisted explanation of the original errors and fixes

• Sample object creation and method usage

Error Code:



```
class Student:

    def display():
        print("Name:", name)
        print("Roll:", roll)

s1 = Student("Shyla", 101)
s1.display()
```

```
PS C:\Users\[illegible]\OneDrive\Desktop\postgreSQL.demo\config> & "C:/Program Files/Python311/python.exe" c:/Users/[illegible]/OneDrive/Desktop/postgreSQL.demo/config/[illegible].py
Traceback (most recent call last):
  File "C:\Users\[illegible]\OneDrive\Desktop\postgreSQL.demo\config\[illegible].py", line 10, in <module>
    s1 = Student("Shyla", 101)
TypeError: Student.__init__() takes 1 positional argument but 3 were given
PS C:\Users\[illegible]\OneDrive\Desktop\postgreSQL.demo\config>
```

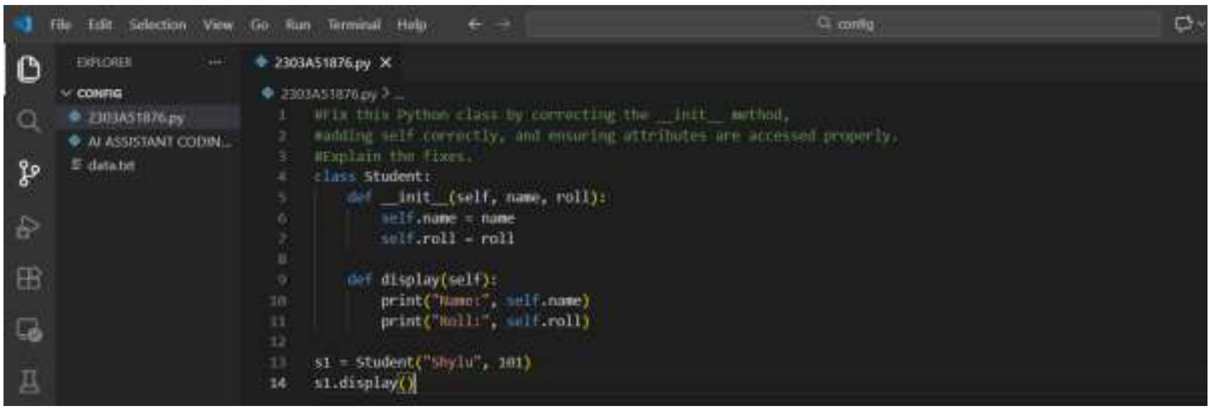
Prompt:

#Fix this Python class by correcting the __init__ method,

#adding self correctly, and ensuring attributes are accessed properly.

#Explain the fixes.

Corrected Code:

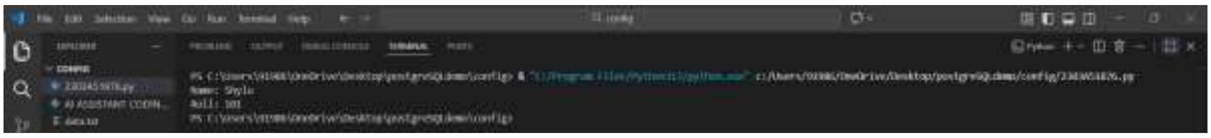


```python
#Fix this Python class by correcting the __init__ method,
#adding self correctly, and ensuring attributes are accessed properly.
#Explain the fixes.
class Student:
    def __init__(self, name, roll):
        self.name = name
        self.roll = roll

    def display(self):
        print("Name:", self.name)
        print("Roll:", self.roll)

s1 = Student("Shylu", 101)
s1.display()
```

Output:



```
PS C:\Users\91986\OneDrive\Desktop\postgreSQLdemo\config> & "C:/Program Files/Python313/python.exe" c:/Users/91986/OneDrive/Desktop/postgreSQLdemo/config/2303A51876.py
Name: Shylu
Roll: 101
PS C:\Users\91986\OneDrive\Desktop\postgreSQLdemo\config>
```

Observation

The original program failed during object creation and method calls due to incorrect constructor parameters and missing attribute references. After applying AI-suggested corrections, the class initialized properly and methods accessed object attributes without errors.

Justification

Proper object-oriented programming in Python requires correct use of the self keyword to bind data to class objects. Incorrect constructor definitions and attribute references lead to runtime errors and unexpected behavior. Debugging these issues ensures correct object initialization and access.

Conclusion

Correct usage of self is essential for class-based programming in Python. AI tools like GitHub Copilot efficiently identified constructor and attribute errors and suggested accurate fixes, resulting in a fully functional and reliable class implementation.