

ASSIGNMENT-7.4

NAME:K.manjula

H.NO:2303A51857

B-13

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:

```
def factorial(n):  
    return n * factorial(n - 1)  
print(factorial(5))
```

OUTPUT:

```
⊙ Traceback (most recent call last):
  File "c:\Users\shash\Downloads\AI AC\ass 7.4.py", line 3, in <module>
    print(factorial(5))
    ~~~~~~^
  File "c:\Users\shash\Downloads\AI AC\ass 7.4.py", line 2, in factorial
    return n * factorial(n - 1)
    ~~~~~~^
  File "c:\Users\shash\Downloads\AI AC\ass 7.4.py", line 2, in factorial
    return n * factorial(n - 1)
    ~~~~~~^
  File "c:\Users\shash\Downloads\AI AC\ass 7.4.py", line 2, in factorial
    return n * factorial(n - 1)
    ~~~~~~^
[Previous line repeated 996 more times]
RecursionError: maximum recursion depth exceeded
```

CORRECTED CODE:

```
def factorial(n):
    if n == 0 or n == 1: # Base case
        return 1
    else:
        return n * factorial(n - 1) # Recursive case

print(factorial(5))
```

OUTPUT:

```
PS C:\Users\shash\Downloads\AI AC> & C:\Users\shash\AppData\Local\Programs\Python\Python314\python.exe "c:\Users\shash\Downloads\AI AC\ass 7.4.py"
120
PS C:\Users\shash\Downloads\AI AC>
```

CHANGES IN CODE:

- **Added base condition:**

Python

```
if n == 0 or n == 1:
    return 1
```

- **Kept recursive call intact:**

Python

```
return n * factorial(n - 1)
```

JUSTIFICATION:

The factorial function was giving errors because the base condition was missing or incorrect. Due to this, the function kept calling itself and caused crashes. Using AI, the issue was identified and the correct base case was added. After fixing the recursive logic, the function returned correct factorial values.

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 CODE:

```
def sort_list(lst):  
    return sorted(lst)  
print(sort_list([3, "2", 5, "10"]))
```

OUTPUT:

```
Traceback (most recent call last):
  File "c:\Users\shash\Downloads\AI AC\ass 7.4.py", line 3, in <module>
    print(sort_list([3, "2", 5, "10"]))
    ~~~~~~
  File "c:\Users\shash\Downloads\AI AC\ass 7.4.py", line 2, in sort_list
    return sorted(lst)
TypeError: '<' not supported between instances of 'str' and 'int'
PS C:\Users\shash\Downloads\AI AC>
```

CORRECTED CODE:

```
def sort_list(lst):
    return sorted(lst, key=lambda x: str(x))
print(sort_list([3, "2", 5, "10"]))
```

OUTPUT:

```
TypeError: '<' not supported between instances of 'str' and 'int'
● PS C:\Users\shash\Downloads\AI AC> & C:\Users\shash\AppData\Local\Programs\Python\Python314\python.exe "c:/User
.4.py"
  ['10', '2', 3, 5]
○ PS C:\Users\shash\Downloads\AI AC>
```

CHANGES IN CODE:

```
def sort_list(lst):
    # Change 1: Added a key function (lambda x: str(x))

    # Reason: Prevents TypeError when list contains mixed data types (int + s

    # Explanation: By converting every element to a string for comparison,

    # Python can sort consistently without crashing.

    return sorted(lst, key=lambda x: str(x))
```

JUSTIFICATION:

The sorting function failed because the list contained mixed data types like numbers and strings. Python cannot compare these directly, which caused a runtime error. AI helped detect this problem and fixed it by converting all values to the same type. This made the sorting process safe and error-free.

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 CODE:

```
f = open("ass 7.4.py", "r")
data = f.read()
print(data)
```

OUTPUT:

```
FileNotFoundError: [Errno 2] No such file or directory: 'ass 7.4.txt'
PS C:\Users\shash\Downloads\AI AC> & C:\Users\shash\AppData\Local\Programs\Python\Python314\python.exe "c:/Users/shash/Downloads/AI AC/ass 7
.4.py"

f = open("ass 7.4.py", "r")
data = f.read()
print(data)

PS C:\Users\shash\Downloads\AI AC>
```

CORRECTED CODE:

```
with open("ass 7.4.py", "r") as f:
    data = f.read()
    print(data)
```

OUTPUT:

```
PS C:\Users\shash\Downloads\AI AC> & C:\Users\shash\AppData\Local\Programs\Python\Python314\python.exe "c:/Users/shash/Downloads/AI AC/ass 7
.4.py"

with open("ass 7.4.py", "r") as f:
    data = f.read()
    print(data)

PS C:\Users\shash\Downloads\AI AC>
```

CHANGES IN CODE:

```
# Corrected file-handling code with documentation of changes

# Change 1: Replaced manual open() with context manager (with open(...))
# Reason: Ensures the file is automatically closed after use, even if an error occurs
# Benefit: Prevents resource leaks and makes code cleaner and safer.

with open("ass 7.4.txt", "r") as f:
    data = f.read()
    print(data)

# No need for f.close() - handled automatically by the context manager
```

JUSTIFICATION:

The program opened a file but did not close it properly, which can lead to memory leaks. AI suggested using the `with open()` statement to manage the file automatically. This approach ensures the file is closed after use and improves performance and reliability.

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:

```
numbers = [10, 5, 0, 2]
for n in numbers:
    result = 10 / n    # ✖ Error occurs here when n = 0
    print(f"10 / {n} = {result}")
```

OUTPUT:

```
PS C:\Users\shash\Downloads\AI AC> & C:\Users\shash\AppData\Local\Programs\Python\Python314\python.exe "c:/Users/shash/Downloads/AI AC/ass 7
.4.py"
10 / 10 = 1.0
10 / 5 = 2.0
Traceback (most recent call last):
  File "c:/Users/shash/Downloads/AI AC/ass 7.4.py", line 4, in <module>
    result = 10 / n # ✖ Error occurs here when n = 0
    ~~~~~
ZeroDivisionError: division by zero
PS C:\Users\shash\Downloads\AI AC> |
```

CORRECTED CODE:

```
numbers = [10, 5, 0, 2]
for n in numbers:
    try:
        result = 10 / n
        print(f"10 / {n} = {result}")
    except ZeroDivisionError:
        print(f"Error: Cannot divide by zero (n = {n})")
```

OUTPUT:

```
PS C:\Users\shash\Downloads\AI AC> & C:\Users\shash\AppData\Local\Programs\Python\Python314\python.exe "c:/Users/shash/Downloads/AI AC/ass 7
.4.py"
10 / 10 = 1.0
10 / 5 = 2.0
Error: Cannot divide by zero (n = 0)
10 / 2 = 5.0
PS C:\Users\shash\Downloads\AI AC> |
```

JUSTIFICATION:

The program stopped when a division by zero error occurred inside the loop. AI detected the exact error location and added exception handling using try-except. Now the program displays an error message and continues processing remaining values.

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 __init__(name, age):
        name = name
        age = age

    def display(self):
        print("Name:", name)
        print("Age:", age)

s1 = Student("Shashi", 21)
s1.display()
```

OUTPUT:

```
PS C:\Users\shash\Downloads\AI AC> & C:\Users\shash\AppData\Local\Programs\Python\Python314\python.exe "c:/Users/shash/Downloads/AI AC/ass 7
.4.py"
Traceback (most recent call last):
  File "c:/Users/shash/Downloads/AI AC/ass 7.4.py", line 11, in <module>
    s1 = Student("Shashi", 21)
TypeError: Student.__init__() takes 2 positional arguments but 3 were given
PS C:\Users\shash\Downloads\AI AC> |
```

CORRECTED CODE:

```
class Student:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def display(self):
        print("Name:", self.name)
        print("Age:", self.age)

s1 = Student("Shashi", 21)
s1.display()
```

OUTPUT:

```
PS C:\Users\shash\Downloads\AI AC> & C:\Users\shash\AppData\Local\Programs\Python\Python314\python.exe "c:/Users/shash/Downloads/AI AC/ass 7
.4.py"
Name: Shashi
Age: 21
PS C:\Users\shash\Downloads\AI AC> |
```

CHANGES IN CODE:

1. **Added self in __init__**
→ Needed so Python knows which object we are working with.
2. **Used self.name and self.age**
→ Stores the values inside the object instead of losing them.
3. **Changed print("Name:", name) to print("Name:", self.name)**
→ Correct way to access the stored value.
4. **Changed print("Age:", age) to print("Age:", self.age)**
→ Same reason, now it prints the object's age correctly.

JUSTIFICATION:

The class was not working correctly because self was missing in the constructor and attributes were not accessed properly. AI corrected the constructor and fixed attribute references. After these changes, object creation and method execution worked correctly.