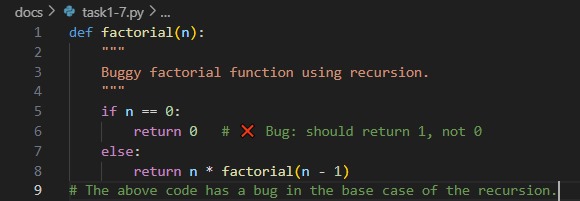
**AI ASSITED CODING **

***Lab 7: Error Debugging with AI – Systematic Approaches to Finding and Fixing Bugs***

**#Task-1:**

**Prompt**: 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.

**A computer screen with text on it

AI-generated content may be incorrect.Code Generated:**

**Observation:**

The first code contains a bug in the base case of the recursive factorial function. It incorrectly returns 0 when n == 0, which causes all recursive results to evaluate to zero. In contrast, the corrected version fixes the base case by returning 1 when n == 0, which is mathematically correct since 0! = 1. Apart from this change, both functions use the same recursive logic where n \* factorial(n-1) computes the factorial. Thus, the main difference lies in handling the base case: the buggy code gives wrong outputs, while the corrected code produces accurate factorial values.

**#Task-2:**

A computer screen with text

AI-generated content may be incorrect.**Prompt**: Provide a list sorting function that fails due to a type error (e.g., sorting list with mixed  
integers and strings).

A computer screen with text

AI-generated content may be incorrect.Prompt AI to detect the issue and fix the code for consistent sorting.

**Observation:**

The first code (buggy\_sort) tries to sort a list directly using sorted(lst), but it fails with a TypeError when the list contains mixed data types like integers and strings, since Python cannot compare them directly. The corrected code (fixed\_sort) resolves this by using sorted(lst, key=str), which converts all elements to strings before comparison, ensuring consistent sorting without errors. While the buggy version highlights the limitation of sorting mixed types, the fixed version guarantees that sorting works smoothly, though it treats numbers as strings, meaning numeric order is lost in favour of lexicographic order.

**#Task-3:**

A screen shot of a computer program

AI-generated content may be incorrect.**Prompt:**Provide a piece of code with a ZeroDivisionError inside a loop

A screen shot of a computer program

AI-generated content may be incorrect.Ask AI to add error handling  
using try-except and continue execution safely.

**Observation:**

In the first code, the loop attempts to divide numbers in a list by 10, but when it encounters zero, a ZeroDivisionError occurs and the program crashes immediately, preventing further execution. In the corrected version, the risky division is wrapped in a try-except block, which catches the error when division by zero happens. Instead of stopping the program, it prints a warning message and safely continues with the remaining iterations. This makes the corrected code more robust and reliable, as it gracefully handles errors without interrupting execution.

#**Task-4:**

**Prompt:**

Write a Python snippet for file handling that opens a file but forgets

A screenshot of a computer program

AI-generated content may be incorrect.to close it

**A screenshot of a computer program

AI-generated content may be incorrect.**Ask Copilot or Cursor AI to improve it using the best practice (e.g., with open() block).

**Observation:**

In the first code, the file is opened using open("sample.txt", "r") but never explicitly closed, which can cause issues like memory leaks or file locks if used repeatedly. The improved version suggested by Copilot uses the with open() context manager, which ensures that the file is automatically closed once the block is exited, even if an error occurs. This makes the code more reliable, safer, and a best practice in Python file handling compared to the manual open-and-forget approach of the first version.

**#Task-5:**

A computer screen shot of code

AI-generated content may be incorrect.**Prompt:** Include a buggy class definition with incorrect \_\_init\_\_ parameters or attribute references

**A computer screen shot of a program code

AI-generated content may be incorrect.**Ask AI to analyze and correct the constructor and attribute usage.

**Observation:**

In the first code, the class definition is buggy because the constructor \_\_init\_\_ is missing the required self-parameter, and the attributes are referenced using self even though it was never defined, which causes errors when creating objects. Additionally, the display method does not include self as a parameter, so it cannot access instance attributes. In the corrected version, Copilot adds self to both the constructor and the method, ensuring that instance variables name and age are properly initialized and accessed. This makes the class functional, allowing objects to be created and their details displayed without errors.