

# AI ASSISTED CODING

## LAB-7.4

NAME: Sumanth  
ENROLL NO: 2403A52025  
BATCH:04

### TASK :01

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:

```
lab7.4.1.py > ...
1  def buggy_recursive_factorial(n):
2      """
3      This function attempts to calculate the factorial of a number using recursion,
4      but it has a bug that causes infinite recursion.
5
6      For any integer n > 0, it will cause a RecursionError.
7
8      Args:
9      |   n: A non-negative integer.
10
11     Returns:
12     |   The calculated factorial of n.
13     """
14     if not isinstance(n, int) or n < 0:
15         raise ValueError("Input must be a non-negative integer")
16
17     # The base case is correct.
18     if n == 0:
19         return 1
20
21     # The bug is in the recursive step below.
22     # It calls itself with the same number 'n' instead of 'n - 1',
23     # so it never reaches the base case.
24     return n * buggy_recursive_factorial(n)
25
26 # Example of the bug in action:
27 # try:
28 #     # This will raise a RecursionError
29 #     print(f"buggy_recursive_factorial(3) = {buggy_recursive_factorial(3)}")
30 # except RecursionError as e:
31 #     print(f"Calling buggy_recursive_factorial(3) caused an error: {e}")
32 #
33 # Expected output for factorial(3): 6
34 # Actual output: RecursionError: maximum recursion depth exceeded in comparison
35 buggy_recursive_factorial(3)
36
```

Detect the bugs in the code and correct it .

## CODE:

```
lab7.4.1.py > ...
1 def recursive_factorial(n):
2     """
3     This function calculates the factorial of a number using recursion.
4
5     Args:
6     |     n: A non-negative integer.
7
8     Returns:
9     |     The calculated factorial of n.
10    """
11    if not isinstance(n, int) or n < 0:
12        raise ValueError("Input must be a non-negative integer")
13
14    # Base case: factorial of 0 is 1.
15    if n == 0:
16        return 1
17
18    # Recursive step: n * factorial(n-1)
19    return n * recursive_factorial(n - 1)
20
21 # Example of the corrected function in action:
22 try:
23     result = recursive_factorial(int(input("Enter a non-negative integer: ")))
24     print(f"recursive_factorial(5) = {result}")
25 except ValueError as e:
26     print(f"Calling recursive_factorial(3) caused an error: {e}")
27
```

## OUTPUT:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS
PS C:\Users\ramch\OneDrive\Desktop\ai> & C:/Users/ramch/AppData/Local/Programs/Python/Python313/python.exe c:/Users/ramch/OneDrive/Desktop/ai/lab7.4.1.py
Enter a non-negative integer: 5
recursive_factorial(5) = 120
PS C:\Users\ramch\OneDrive\Desktop\ai>
```

## OBSERVATION:

The given prompt code has bugs which are resolved by the AI and rewrite the code.

## TASK :02

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:

```

lab/4.2.py / 🐞 buggy_type_error_sort
1 def buggy_type_error_sort(items):
2     """
3     Attempts to sort a list using a bubble sort algorithm, but has a bug.
4
5     This function will raise a TypeError if the list contains non-numeric
6     types (like strings), as it incorrectly tries to perform an arithmetic
7     operation on the elements before comparing them.
8
9     Args:
10        items: A list of items to sort.
11
12    Returns:
13        A new list with the items sorted.
14    """
15    # Create a copy to avoid modifying the original list in place
16    sorted_items = list(items)
17    n = len(sorted_items)
18
19    for i in range(n):
20        swapped = False
21        for j in range(0, n - i - 1):
22            # THE BUG: This comparison assumes elements are numeric.
23            # Adding an integer (0) to a string will raise a TypeError.
24            if sorted_items[j] + 0 > sorted_items[j + 1] + 0:
25                sorted_items[j], sorted_items[j + 1] = sorted_items[j + 1], sorted_items[j]
26                swapped = True
27
28        if not swapped:
29            break
30
31    return sorted_items
32
33 # --- Example of the bug in action ---
34
35 # This will work because the elements are numbers
36 numbers = [10, 2, 88, 1, -5]
37 print(f"Sorting numbers: {buggy_type_error_sort(numbers)}")
38 # Expected and Actual Output: Sorting numbers: [-5, 1, 2, 10, 88]
39
40 # This will fail with a TypeError
41 strings = ["cherry", "apple", "banana"]
42 try:
43     buggy_type_error_sort(strings)
44 except TypeError as e:
45     print(f"\nSorting strings failed as expected: {e}")
46 # Expected Output: TypeError: can only concatenate str (not "int") to str

```

Detect the issue in the code and correct it.

**CODE:**

```

lab7.4.2.py > bubble_sort
1 def bubble_sort(items):
2     """
3     Sorts a list using a bubble sort algorithm.
4
5     Args:
6     items: A list of comparable items to sort (e.g., all numbers or
7           all strings).
8
9     Returns:
10    A new list with the items sorted.
11    """
12    # Create a copy to avoid modifying the original list in place
13    sorted_items = list(items)
14    n = len(sorted_items)
15
16    for i in range(n):
17        swapped = False
18        for j in range(0, n - i - 1):
19            # The comparison now works for any comparable types.
20            if sorted_items[j] > sorted_items[j + 1]:
21                sorted_items[j], sorted_items[j + 1] = sorted_items[j + 1], sorted_items[j]
22                swapped = True
23
24        if not swapped:
25            break
26
27    return sorted_items
28
29 # --- Example of the corrected function ---
30
31 # This will work because the elements are numbers
32 numbers = [10, 2, 88, 1, -5]
33 print(f"Sorting numbers: {bubble_sort(numbers)}")
34 # Expected Output: Sorting numbers: [-5, 1, 2, 10, 88]
35
36 # This now works correctly with strings as well
37 strings = ["cherry", "apple", "banana"]
38 print(f"Sorting strings: {bubble_sort(strings)}")
39 # Expected Output: Sorting strings: ['apple', 'banana', 'cherry']
40

```

## OUTPUT:

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\ramch\OneDrive\Desktop\ai> & C:/Users/ramch/AppData/Local/Programs/Python/Python313/python.exe c:/Users/ramch/OneDrive/Desktop/ai/lab7.4.2.py
Sorting numbers: [-5, 1, 2, 10, 88]
Sorting strings: ['apple', 'banana', 'cherry']
PS C:\Users\ramch\OneDrive\Desktop\ai>

```

## OBSERVATION:

The code has a bug of not sorting the list of names lexicographically it only sorts the list of numbers in ascending order whereas in the code given by gemini has both the ability.

## TASK :03

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:**

```
def log_message(filename, message):  
  
    file_handle = open(filename, 'a')  
    file_handle.write(f"{message}\n")  
  
    print(f"Wrote to {filename}, but forgot to close it.")  
log_file = "app.log"  
log_message(log_file, "User logged in.")  
log_message(log_file, "Data processing started.")  
log_message(log_file, "Error encountered during processing.")
```

Detect the bug and correct the code.

**CODE:**

```
lab7.4.3 Welcome  
1 def log_message(filename, message):  
2     """  
3     Appends a message to a log file, ensuring the file is properly closed.  
4  
5     Args:  
6         filename (str): The path to the log file.  
7         message (str): The message to write to the file.  
8     """  
9     # Using a 'with' statement is the standard, safe way to handle files.  
10    # It automatically closes the file handle when the block is exited,  
11    # even if an error occurs, preventing resource leaks.  
12    with open(filename, 'a') as file_handle:  
13        file_handle.write(f"{message}\n")  
14    print(f"Successfully logged message to {filename}.")  
15  
16    log_file = "app.log"  
17    log_message(log_file, "User logged in.")  
18    log_message(log_file, "Data processing started.")  
19    log_message(log_file, "Error encountered during processing.")  
20
```

**OUTPUT:**

```
Sorting strings: ['apple', 'banana', 'cherry']  
PS C:\Users\ramch\OneDrive\Desktop\ai> & C:/Users/ramch/AppData/Local/Programs/Python/Python313/python.exe c:/Users/ramch/OneDrive/Desktop/ai/  
lab7.4.3.py  
Successfully logged message to app.log.  
Successfully logged message to app.log.  
Successfully logged message to app.log.  
PS C:\Users\ramch\OneDrive\Desktop\ai>
```

**OBSERVATION:**

The bug in the code is it opens a file to store some data but forgot to close it. The bug is resolved by the AI

#### TASK :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:

Generate a python code to perform the ZeroDivisionError and include try-except methods to improve the efficiency of the code.

#### CODE:

```
def safe_division_loop(numerator, divisors):
    """
    Demonstrates handling a ZeroDivisionError within a loop.

    This function iterates through a list of divisors and attempts to divide
    the numerator by each one. It uses a try-except block to catch the
    ZeroDivisionError and continue the loop without crashing.

    Args:
        numerator (int or float): The number to be divided.
        divisors (list): A list of numbers to divide the numerator by.
    """
    print(f"Starting division loop with numerator: {numerator}\n")

    for divisor in divisors:
        try:
            # Attempt the division operation which might cause an error
            result = numerator / divisor
            print(f"Successfully calculated: {numerator} / {divisor} = {result}")
        except ZeroDivisionError:
            # This block executes ONLY if a ZeroDivisionError occurs
            print(f"Error: Cannot divide by zero. Skipping divisor '{divisor}'.")
            # The loop will automatically continue to the next item

    print("\nLoop finished. The program continued execution safely.")

# --- Example Usage ---

# A list of numbers to use as divisors, including 0 to trigger the error.
numbers_to_divide_by = [10, 5, 2, 0, 8, 4]
main_number = 100

safe_division_loop(main_number, numbers_to_divide_by)
```

#### OUTPUT:

```

PS C:\Users\ramch\OneDrive\Desktop\ai> & C:/Users/ramch/AppData/Local/Programs/Python/Python313/py
lab7.4.4.py
Starting division loop with numerator: 100

Successfully calculated: 100 / 10 = 10.0
Successfully calculated: 100 / 5 = 20.0
Successfully calculated: 100 / 2 = 50.0
Error: Cannot divide by zero. Skipping divisor '0'.
Successfully calculated: 100 / 8 = 12.5
Successfully calculated: 100 / 4 = 25.0

Loop finished. The program continued execution safely.
PS C:\Users\ramch\OneDrive\Desktop\ai>

```

## OBSERVATION:

The code generated by AI has the ability to perform the operations like ZeroDivisionError and giving an efficient output.

## TASK :05

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

## PROMPT:

```

lab7.4.5.py > BuggyUser > get_greeting
1 class BuggyUser:
2     def __init__(self, username, email):
3
4         print(f"Initializing user with username: {username}")
5         username = username
6         self.email = email
7         self.is_active = True
8
9     def get_greeting(self):
10        return f"Welcome back, {self.username}!"
11 print("Creating a BuggyUser instance...")
12 user = BuggyUser("alex_123", "alex@example.com")
13
14 print("\nAttempting to call a method on the instance...")
15 try:
16     # This call will fail.
17     greeting = user.get_greeting()
18     print(greeting)
19 except AttributeError as e:
20     print(f"Caught an expected error: {e}")
21     print("This happened because 'self.username' was not set in __init__.")
22
23

```

Detect the bug in the code and rewrite the correct code with the correct class.

## CODE:

```

class User:
    """A simple class to represent a user."""
    def __init__(self, username, email):
        """
        Initializes a User object.

        Args:
            username (str): The user's username.
            email (str): The user's email address.
        """
        # THE FIX: Assign the 'username' parameter to an instance attribute.
        self.username = username
        self.email = email
        self.is_active = True

    def get_greeting(self):
        """Returns a welcome message for the user."""
        return f"Welcome back, {self.username}!"

print("Creating a User instance...")
user = User("alex_123", "alex@example.com")

print("\nCalling the get_greeting method...")
greeting = user.get_greeting()
print(greeting)

```

## OUTPUT:

```

PS C:\Users\ramch\OneDrive\Desktop\ai> & C:/Users/ramch/AppData/Local/Programs/Python/Python313/python.exe c:/Users/ramch/OneDrive/Desktop/ai/lab7.4.5.py
Creating a User instance...

Calling the get_greeting method...
Welcome back, alex_123!
PS C:\Users\ramch\OneDrive\Desktop\ai>

```

## OBSERVATION:

The AI has efficiently corrected the bugs in the code provided by the user and gave the debugged code.