

ASSIGNMENT-7.4

NAME : P.Srikar

Roll No : 2403a52188

Batch : 07

Task Description #1:

- 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.

Expected Outcome #1:

- Copilot or Cursor AI correctly identifies missing base condition or incorrect recursive call and suggests a functional factorial implementation

PROMPT:

Give me a buggy python code that calculates the factorial of a number using recursions and i need to identify the error.

◆ Gemini

0s



```
def factorial(n):
    if n == 0:
        return 1
    else:
        return n * factorial(n-1)

print(factorial(5)) # This should work fine

# Now let's try with a negative number
print(factorial(-5)) # This will cause an error
```



120

```
-----
RecursionError                                Traceback (most recent call last)
/tmp/ipython-input-4147717046.py in <cell line: 0>()
      8
      9 # Now let's try with a negative number
--> 10 print(factorial(-5)) # This will cause an error
```

⬆ 1 frames

... last 1 frames repeated, from the frame below ...

```
/tmp/ipython-input-4147717046.py in factorial(n)
      3     return 1
      4     else:
----> 5         return n * factorial(n-1)
      6
      7 print(factorial(5)) # This should work fine
```

RecursionError: maximum recursion depth exceeded



```
def factorial(n):
    if n < 0:
        raise ValueError("Factorial is not defined for negative numbers")
    elif n == 0:
        return 1
    else:
        return n * factorial(n - 1)

# Example usage
print(factorial(5))    # ✓ Works fine → 120
print(factorial(0))    # ✓ Works fine → 1
print(factorial(-5))   # ✗ Raises ValueError
```



120
1

Task Description #2:

- 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.

Expected Outcome #2:

- AI detects the type inconsistency and either filters or converts list elements, ensuring successful sorting without a crash.

PROMPT:

Wrote a function to sort a list, but it fails with a TypeError because the list has both integers and strings. Can you explain the issue and rewrite the function so it sorts the list consistently.

```
✓ [18] def sort_mixed_list_with_error(mixed_list):  
0s  
    # This function is intended to sort a list with mixed data types  
    # It will cause a TypeError  
    mixed_list.sort()  
    return mixed_list  
  
    # Example usage (will cause a TypeError)  
    mixed_data = [5, 'apple', 2, 'banana', 1, 'cherry']  
    # print(sort_mixed_list_with_error(mixed_data))
```

```
✓ [19] # Uncomment the line below to see the TypeError  
0s  
    # sort_mixed_list_with_error(mixed_data)
```

```
◆ Gemini  
✓ [19] def sort_mixed_list_corrected(mixed_list):  
0s  
    # This function sorts a list with mixed data types by converting all elements to strings  
    return sorted(mixed_list, key=str)  
  
    # Example usage  
    mixed_data = [5, 'apple', 2, 'banana', 1, 'cherry']  
    sorted_data = sort_mixed_list_corrected(mixed_data)  
    print(sorted_data)
```

```
🔗 [1, 2, 5, 'apple', 'banana', 'cherry']
```

Task Description #3:

- 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).

Expected Outcome #3:

- AI refactors the code to use a context manager, preventing resource leakage and runtime warnings.

PROMPT:

I wrote a Python snippet that opens a file but I forgot to close it. Can you refactor it using best practices like a context manager (with open) to avoid resource leaks

```
[1] # Create a dummy file for demonstration
with open("my_file.txt", "w") as f:
    f.write("This is some sample content.")

# Incorrect way to open and read a file (without closing)
file_handle = open("my_file.txt", "r")
content = file_handle.read()
print("Content read from file (without explicit close):")
print(content)
# File handle is not explicitly closed here, which can lead to resource leaks
```

↔ Content read from file (without explicit close):
This is some sample content.

★ Gemini

```
▶ # Correct way to open and read a file using a context manager (with open)
with open("my_file.txt", "r") as file_handle:
    content = file_handle.read()
    print("\nContent read from file (using with open):")
    print(content)
# The file is automatically closed when exiting the 'with' block
```

↔ Content read from file (using with open):
This is some sample content.

Task Description #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.

Expected Outcome #4:

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

PROMPT:

I wrote a loop in Python that causes a ZeroDivisionError. Can you add try-except so it handles the error and keeps running safely

```
[ ] # Uncomment the line below to see the TypeError
    # sort_mixed_list_with_error(mixed_data)
```

✓
0s

```
[6] # Example loop that will cause a ZeroDivisionError
    numbers = [10, 5, 0, 2, 8]
    divisor = 0

    # This loop will raise a ZeroDivisionError when it tries to divide by zero
    # for number in numbers:
    #     result = number / divisor
    #     print(f"Result: {result}")
```

✓
0s



```
# Corrected loop with try-except to handle ZeroDivisionError
numbers = [10, 5, 0, 2, 8]
divisor = 0
```

```
for number in numbers:
    try:
        result = number / divisor
        print(f"Result: {result}")
    except ZeroDivisionError:
        print("Error: Division by zero occurred. Skipping this iteration.")
```



```
Error: Division by zero occurred. Skipping this iteration.
Error: Division by zero occurred. Skipping this iteration.
Error: Division by zero occurred. Skipping this iteration.
Error: Division by zero occurred. Skipping this iteration.
Error: Division by zero occurred. Skipping this iteration.
```

Task Description #5:

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

Expected Outcome #5:

- Copilot identifies mismatched parameters or missing self references and rewrites the class with accurate initialization and usage

PROMPT:

I wrote a Python class with mistakes in the `__init__` method and attribute references. Can you fix the constructor and correct the attribute usage

```
✓ [19] class MyClassWithError:
0s   def __init__(self, value):
      # Mistake: Using a different variable name instead of self for attribute
      val = value

      def display_value(self):
          # Mistake: Trying to access attribute using the wrong variable name
          print(f"The value is: {val}")

      # Example usage (will cause an AttributeError)
      # my_object = MyClassWithError(10)
      # my_object.display_value()
```

```
✓ [20] # Uncomment the lines below to see the AttributeError
0s     # my_object = MyClassWithError(10)
     # my_object.display_value()
```

```
[18] class MyClassCorrected:
      def __init__(self, value):
          # Correct: Using self to create an instance attribute
          self.value = value

      def display_value(self):
          # Correct: Accessing the instance attribute using self
          print(f"The value is: {self.value}")

# Example usage of the corrected class
my_object_corrected = MyClassCorrected(10)
my_object_corrected.display_value()
```



The value is: 10