Assignment-7.4

Name: G.Sudeeksha

Roll-No: 2403A510E0

Batch No: 05

Course: AI Assisted Coding

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.

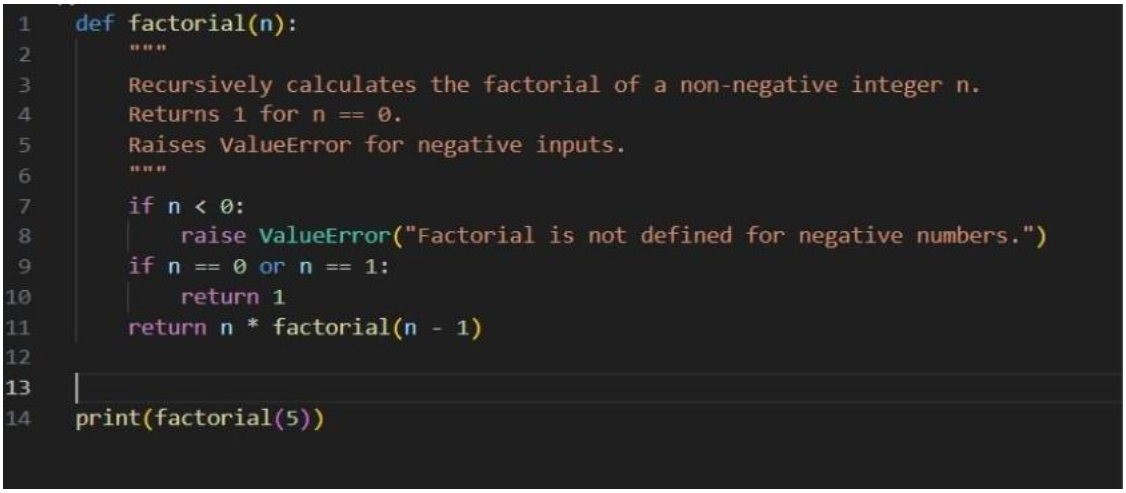
Outcome #1:

• Copilot or Cursor AI correctly identifies missing base condition or incorrect recursive call and

suggests a functional factorial implementation

#Prompt: Calculate the factorial of a number using recursion.

Code:



```python
def factorial(n):
    """
    Recursively calculates the factorial of a non-negative integer n.
    Returns 1 for n == 0.
    Raises ValueError for negative inputs.
    """
    if n < 0:
        raise ValueError("Factorial is not defined for negative numbers.")
    if n == 0 or n == 1:
        return 1
    return n * factorial(n - 1)

print(factorial(5))
```

Output:

```
PS C:\Users\Reshm\Desktop\AIAC> & "C:/Program Files/Python313/python.exe" c:/Users/Reshm/Desktop/AIAC/7.3.py
120
PS C:\Users\Reshm\Desktop\AIAC>
```

Observation :

- The buggy version lacks a base case, causing infinite recursion and a stack overflow error.
- The corrected version adds a base case (n == 0 or n == 1) and input validation for negative numbers.
- The recursive call is properly structured, ensuring correct calculation for all non-negative integers.
- This demonstrates the importance of base conditions in recursive functions to prevent runtime errors.

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.

Outcome #2:

AI detects the type inconsistency and either filters or

converts list elements, ensuring successful sorting without a crash.

#Prompt: Detect the issue and fix the code for consistent sorting

Code:

```python
def fixed_sort(lst):
    """
    Sorts a list with mixed integers and strings by converting all elements to integers
    Non-numeric strings are ignored.
    """
    int_list = []
    for x in lst:
        try:
            int_list.append(int(x))
        except ValueError:
            continue  # Skip non-numeric strings
    return sorted(int_list)


mixed_list = [3, "2", 1, "4", "abc"]
print(fixed_sort(mixed_list))
```

Output:

```
PS C:\Users\Reshm\Desktop\AIAC> & "C:/Program Files/Python313/python.exe" c:/Users/Reshm/Desktop/AIAC/7.3.py
[1, 2, 3, 4]
PS C:\Users\Reshm\Desktop\AIAC>
```

Obseravation:

- The buggy version fails with a TypeError when trying to sort a list containing both integers and strings, as Python cannot compare these types directly.
- The fixed version converts all elements to integers before sorting, skipping non-numeric strings, which ensures consistent and error-free sorting.
- This demonstrates the importance of handling type consistency when performing operations like sorting on mixed-type lists.

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).

Outcome #3:

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

#Prompt: To write a Python snippet for file handling that opens a file

Code:

```
with open("example.txt", "w") as f:
    f.write("Hello, world!")

with open("example.txt", "r") as f:
    content = f.read()
    print(content)
```

Output:

```
PS C:\Users\Reshm\Desktop\AIAC> & "C:/Program Files/Python313/python.exe" c:/Users/Reshm/Desktop/AIAC/7.3.py
Hello, world!
PS C:\Users\Reshm\Desktop\AIAC>
```

Observation:

The code uses a context manager (with open(...) as f:), which ensures the file is automatically closed after writing, preventing resource leaks.The string "Hello, world!" is written to example.txt, but nothing is printed to the console unless you explicitly read and print the file contents.This approach is considered best practice for file operations in Python, as it handles exceptions and resource management cleanly. To verify the output, you should read the file and print its contents after writing.

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: Write a loop that causes a ZeroDivisionError, then use AI to add try-except so the loop continues safely

Code:

```python
numbers = [5, 2, 0, 8]
for n in numbers:
    try:
        result = 10 / n
        print(result)
    except ZeroDivisionError:
        print(f"Error: Division by zero for n={n}. Skipping.")
```

Output:

```
PS C:\Users\Reshm\Desktop\AIAC> & "C:/Program Files/Python313/python.exe" c:/Users/Reshm/Desktop/AIAC/7.3.py
2.0
5.0
Error: Division by zero for n=0. Skipping.
1.25
PS C:\Users\Reshm\Desktop\AIAC>
```

Observation:

The code safely handles division by zero using a try-except block inside the loop.

When a zero is encountered, it prints a clear error message and continues processing the remaining numbers.

Valid divisions are printed as expected.

This approach prevents the program from crashing and demonstrates robust error handling in Python loops.

Task Description #5:

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

Outcome #5:

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

#Prompt: Write a buggy Python class with incorrect init parameters or attribute references, then use AI to fix the constructor and attribute usage.

Code:

```python
class Player:
    def __init__(self, name, score):
        self.name = name
        self.score = score

    def display(self):
        print(f"Player: {self.name}, Score: {self.score}")

# Usage example
p = Player("Alice", 42)
p.display()
```

Output :

```
PS C:\Users\Reshm\Desktop\AIAC> & "C:/Program Files/Python313/python.exe" c:/Users/Reshm/Desktop/AIAC/7.3.py
Player: Alice, Score: 42
PS C:\Users\Reshm\Desktop\AIAC>
```

Observation:

The buggy version omits the self parameter in _init and does not use self to reference attributes, causing errors and preventing proper object initialization.The corrected version includes self in both the constructor and attribute references, ensuring that instance variables are set and accessed correctly.This highlights the importance of using self in Python class methods to maintain object state and avoid runtime errors