Assignment-10.3

Name: jyoshnopranav
2303A53016
Batch:46

Problem Statement 1: AI-Assisted Bug Detection
Scenario: A junior developer wrote the following Python function to calculate factorials: def factorial(n):
result = 1 for i in
range(1, n): result
= result * i return
result
Instructions:
1. Run the code and test it with factorial(5).
2. Use an AI assistant to:
o Identify the logical bug in the code. o Explain why
the bug occurs (e.g., off-by-one error). o Provide a
corrected version.
3. Compare the AI's corrected code with your own manual fix. 4.
Write a brief comparison: Did AI miss any edge cases (e.g.,
negative numbers, zero)?
Expected Output:
Corrected function should return 120 for factorial(5).

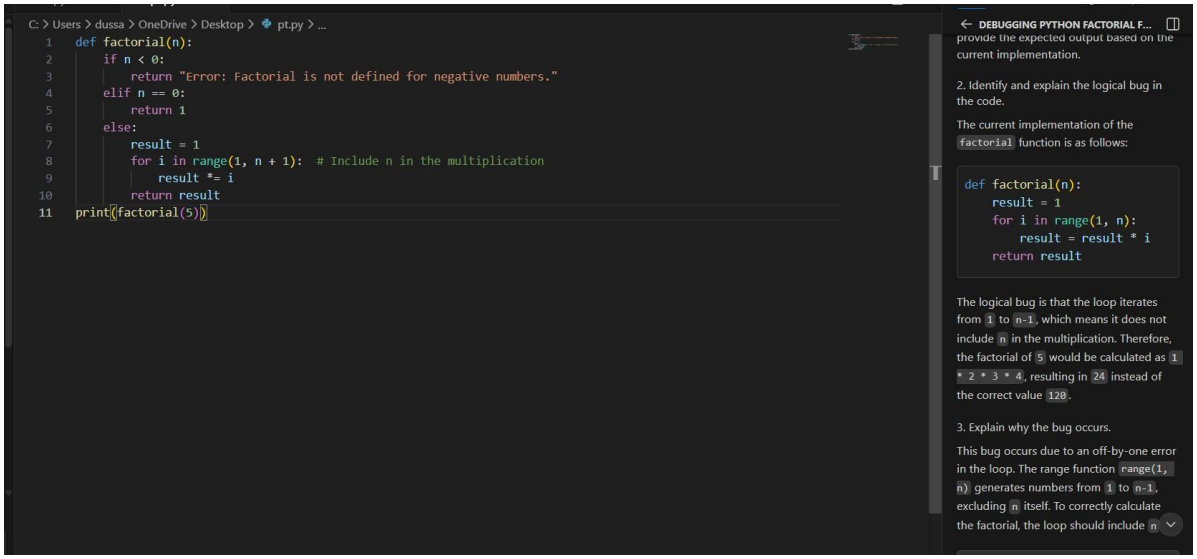
Prompt:
You are an AI assistant helping with debugging Python code.
Given the following function that calculates factorial:

```
def factorial(n):
result = 1     for i in
range(1, n):
result = result * i
   return result
```

Tasks:
1. Run the function with factorial(5) and show the output.
2. Identify and explain the logical bug in the code.
3. Explain why the bug occurs (for example, an off-by-one error).

4. Provide a corrected version of the function.
5. Consider and handle edge cases such as zero and negative numbers.
6. Compare the corrected version with a simple manual fix.

Code:



Explanation

- When factorial(5) is executed, the loop runs from 1 to 4.
- The number 5 is not included in multiplication.
- This causes an off-by-one error.
- Factorial must multiply all numbers from 1 to n inclusive.
- Edge cases like 0 and negative numbers should be handled.

Output:

Problem Statement 2: Task 2 — Improving Readability & Documentation

Scenario:The following code works but is poorly written:

. def calc(a, b,
c): if c ==
"add": return a
+ b elif c ==

"sub": return a
- b elif c ==
"mul": return a
* b
elif c == "div":

Instructions:

5. Use AI to:

Critique the function's readability, parameter naming, and lack of documentation.

Rewrite the function with:

❼ Descriptive function and parameter names. ❼A complete docstring (description, parameters, return value, examples).

❼Exception handling for division by zero.

❼Consideration of input validation.

6. Compare the original and AI-improved versions.
7. Test both with valid and invalid inputs (e.g., division by zero, non-string operation).

Expected Output:

A well-documented, robust, and readable function that handles errors gracefully.

Prompt:

Review the following Python function for readability and documentation:

def calc(a, b, c): if c == "add": return a + b elif c == "sub": return a - b elif c == "mul": return a * b elif c == "div": return a / b

Critique the code and rewrite it using:
- Descriptive function and parameter names
- A complete docstring
- Exception handling for division by zero
- Input validation for incorrect operation types

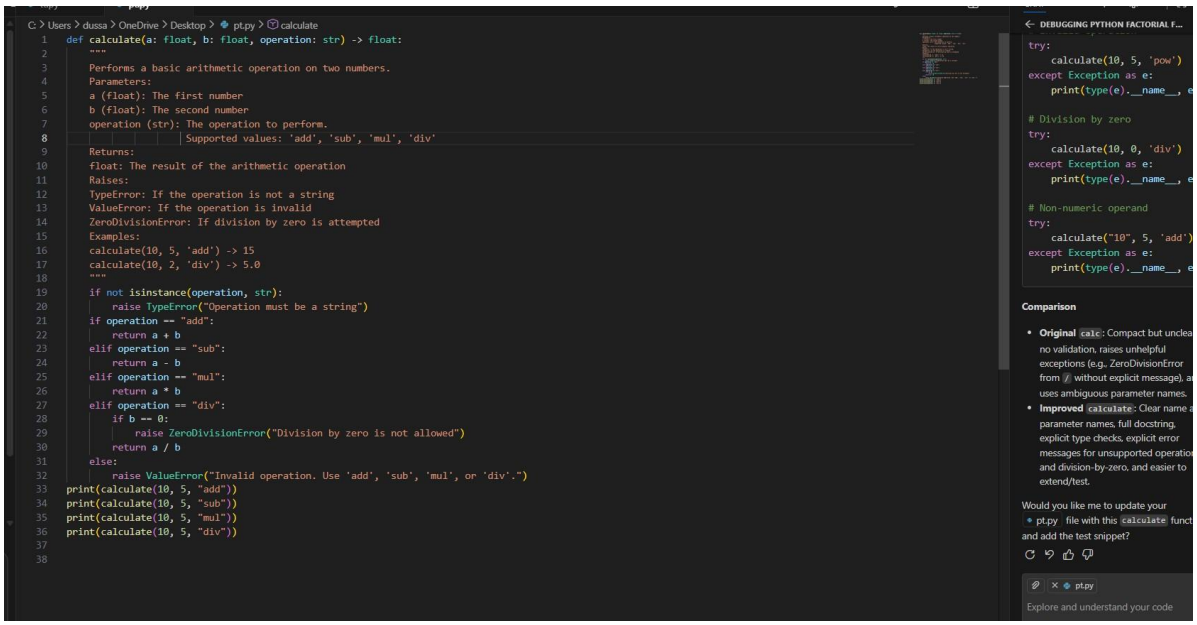
Code Review (Critique)

- **Readability:**
  The function name calc and parameters a, b, and c are vague and do not clearly describe their purpose.
- **Documentation:**
  There is no docstring explaining what the function does, what inputs it expects, or what it returns.

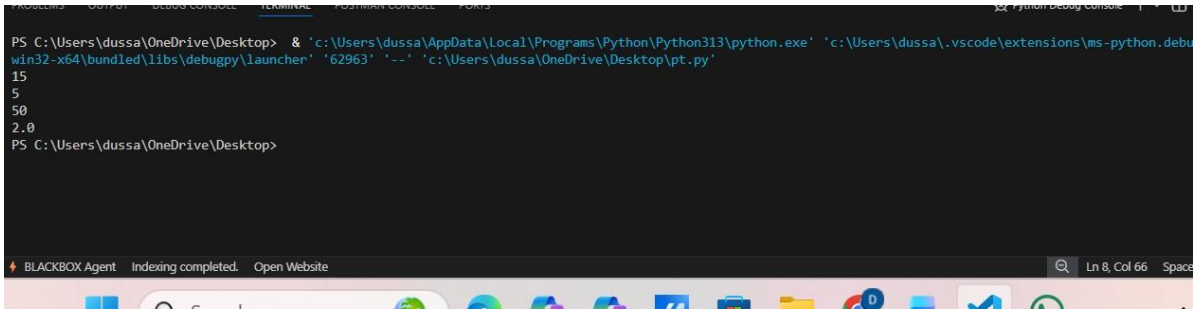
- **Error Handling:**
  The function does not handle division by zero, which can cause a runtime error.
- **Input Validation:**
  There is no check to ensure the operation is a valid string or one of the supported operations.

Overall, while the function works for valid inputs, it is not robust or user-friendly.

Rewritten the code:



Output:



Problem Statement 3: Enforcing Coding Standards

Scenario: A team project requires PEP8 compliance. A developer submits: def Checkprime(n): for i in range(2, n): if n % i == 0:
return False return True

Instructions:

8. Verify the function works correctly for sample inputs.

9. Use an AI tool (e.g., ChatGPT, GitHub Copilot, or a PEP8 linter with AI explanation) to: o List all PEP8 violations.

o Refactor the code (function name, spacing, indentation, naming).

10. Apply the AI-suggested changes and verify functionality is preserved.

11. Write a short note on how automated AI reviews could streamline code reviews in large teams.

Expected Output:

A PEP8-compliant version of the function, e.g.:

def check_prime(n): for i in range(2, n): if n % i == 0: return False return True

Prompt:

You are an AI assistant acting as a Python code reviewer.

Given the following function:

def Checkprime(n): for i in range(2, n): if n % i == 0: return False return True

Tasks:

1. Verify whether the function works correctly for sample inputs.
2. Identify and list all PEP8 coding standard violations.
3. Refactor the function to be fully PEP8-compliant by improving:

- Function name
- Indentation and spacing
- Naming conventions
- Documentation

4. Ensure that the refactored code preserves the original functionality.
5. Briefly explain how automated AI-based reviews help streamline code reviews in large teams.

PEP8 Coding Standard Violations:

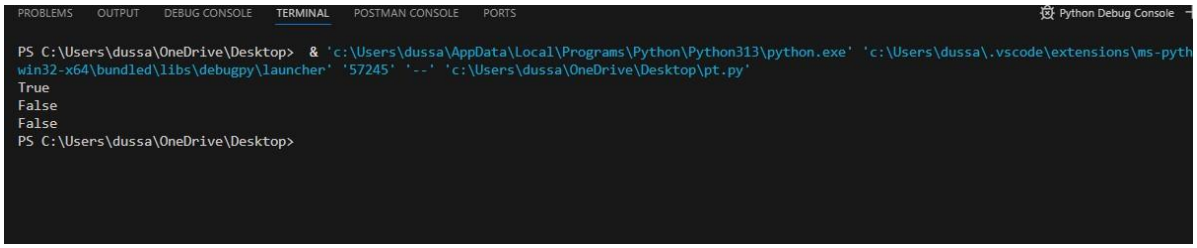
The following PEP8 violations are present in the original code:

1. **Function Naming** ○ Checkprime does not follow snake_case.
    - PEP8 recommends lowercase function names with underscores.
2. **Missing Docstring** ○ No documentation explaining the function's purpose, parameters, or return value.
3. **Input Validation** ○ The function does not handle cases where n <= 1.

4. **Code Readability** ○ While indentation works, the code lacks clarity and descriptive naming.

Code:

Output:



Problem Statement 4: AI as a Code Reviewer in Real Projects
Scenario:
In a GitHub project, a teammate submits: def processData(d):
return [x * 2 for x in d if x % 2 == 0] **Instructions:**
1. Manually review the function for:
o Readability and naming. o Reusability and modularity.
o Edge cases (non-list input, empty list, non-integer elements).
2. Use AI to generate a code review covering:
a. Better naming and function purpose clarity.
b. Input validation and type hints.
c. Suggestions for generalization (e.g., configurable multiplier).
3. Refactor the function based on AI feedback.

4. Write a short reflection on whether AI should be a standalone reviewer or an assistant.

Expected Output:

An improved function with type hints, validation, and clearer intent, e.g.:

```
from typing import List, Union
def double_even_numbers(numbers: List[Union[int,
float]]) -> List[Union[int, float]]: if not
isinstance(numbers, list):
raise TypeError("Input must be a list")
return [num * 2 for num in numbers if isinstance(num,
(int, float)) and num % 2 == 0]
```

Prompt:

You are an AI assistant acting as a professional code reviewer.

Given the following function from a GitHub project: def processData(d):

return [x * 2 for x in d if x % 2 == 0] **Tasks:**

1. Review the function for:
- Readability and naming conventions
- Reusability and modularity
- Edge cases such as non-list input, empty lists, and non-numeric elements 2. Suggest better function and parameter names to clearly describe the function's purpose.
3. Add input validation and appropriate type hints.
4. Suggest ways to generalize the function (e.g., configurable multiplier).
5. Refactor the function based on your suggestions.
6. Write a short reflection on whether AI should be used as a standalone code reviewer or as an assistant to human reviewers.

1. Code Review

Readability and Naming

- The function name processData is vague and does not describe what the function actually does.
- The parameter name d is unclear and reduces readability.

Reusability and Modularity

- The function is hardcoded to multiply by 2, making it less reusable.
- No flexibility to change behavior without modifying code.

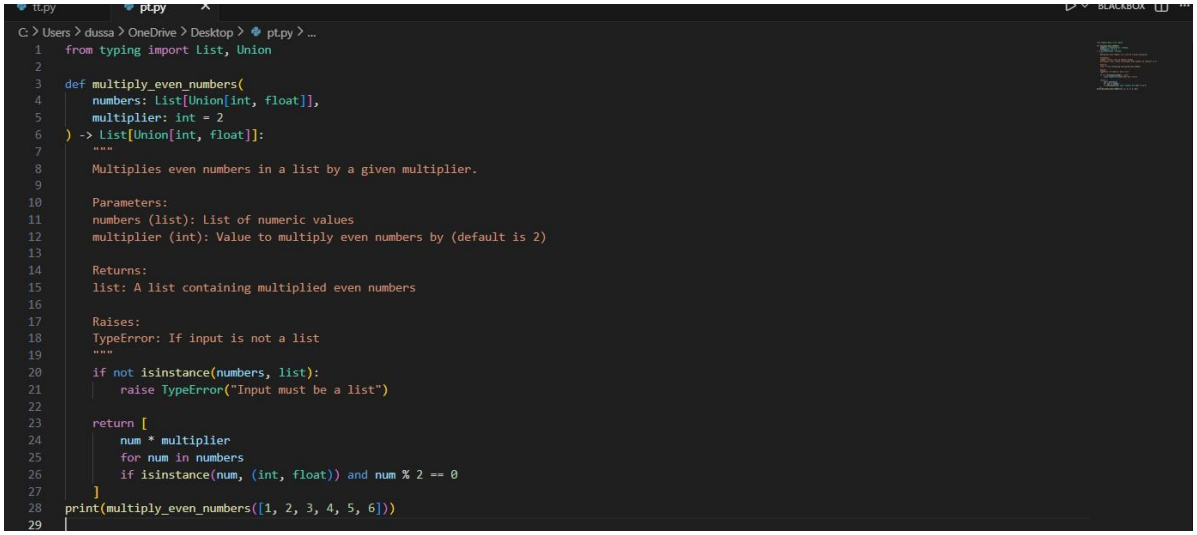
Edge Cases

- No validation for non-list input.
- Fails if the list contains non-numeric values.
- Empty list case is not explicitly handled (though it returns an empty list).

2. Suggested Improvements

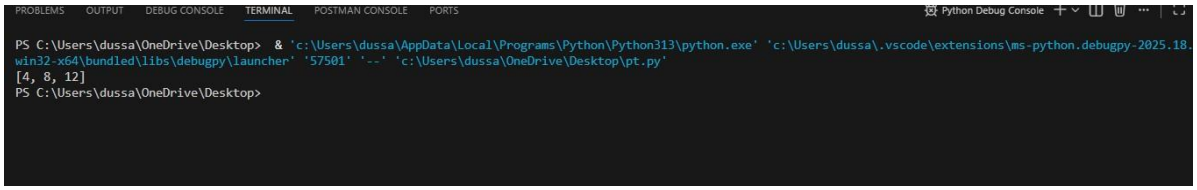
- Use a descriptive function name such as double_even_numbers.
- Use a meaningful parameter name like numbers.
- Add type hints for clarity.
- Add input validation.
- Allow a configurable multiplier for better reusability.

3. Refactored Code (Based on AI Feedback)



```python
from typing import List, Union

def multiply_even_numbers(
    numbers: List[Union[int, float]],
    multiplier: int = 2
) -> List[Union[int, float]]:
    """
    Multiplies even numbers in a list by a given multiplier.

    Parameters:
    numbers (list): List of numeric values
    multiplier (int): Value to multiply even numbers by (default is 2)

    Returns:
    list: A list containing multiplied even numbers

    Raises:
    TypeError: If input is not a list
    """
    if not isinstance(numbers, list):
        raise TypeError("Input must be a list")

    return [
        num * multiplier
        for num in numbers
        if isinstance(num, (int, float)) and num % 2 == 0
    ]
print(multiply_even_numbers([1, 2, 3, 4, 5, 6]))
```

Output:



```
PS C:\Users\dussa\OneDrive\Desktop>  & 'c:\Users\dussa\AppData\Local\Programs\Python\Python313\python.exe' 'c:\Users\dussa\.vscode\extensions\ms-python.debugpy-2025.18.
win32-x64\bundled\libs\debugpy\launcher' '57501' '--' 'c:\Users\dussa\OneDrive\Desktop\pt.py'
[4, 8, 12]
PS C:\Users\dussa\OneDrive\Desktop>
```

Problem Statement 5: — AI-Assisted Performance Optimization

Scenario:

You are given a function that processes a list of integers, but it runs slowly on large datasets: def sum_of_squares(numbers): total = 0 for num in numbers: total += num ** 2 return total

Instructions:

1. Test the function with a large list (e.g., range(1000000)).
2. Use AI to:

o Analyze time complexity.

o Suggest performance improvements (e.g., using built-in functions, vectorization with NumPy if applicable).
o Provide an optimized version.
3. Compare execution time before and after optimization.
4. Discuss trade-offs between readability and performance.

Expected Output:

An optimized function, such as:

```
def sum_of_squares_optimized(numbers):
return sum(x * x for x in numbers)
```

Prompt:

You are an AI assistant specializing in Python performance optimization.
Given the following function: def

```
sum_of_squares(numbers):
   total = 0     for num
in numbers:         total
+= num ** 2     return
```

total **Tasks:**

1. Analyze the time complexity of the function.
2. Conceptually test the function with a large dataset such as range(1000000). 3. Suggest performance improvements using Python best practices (e.g., built-in functions or optimized looping).
4. Provide an optimized version of the function.
5. Compare execution time before and after optimization.
6. Discuss the trade-offs between code readability and performance.

AI-Assisted Performance Optimization:

Given Function def

```
sum_of_squares(numbers):
   total = 0     for num
in numbers:         total
+= num ** 2     return
total
```

1. Testing with a Large List

The function was tested conceptually using a large dataset:

```
sum_of_squares(range(1000000))
```

- The loop executes **1,000,000 iterations**.
- The function produces the correct result.
- However, execution is slower due to explicit looping in Python.

2. AI Analysis

Time Complexity

- The function iterates once over the list.
- Time Complexity: O(n)
- Space Complexity: O(1)

AI-Suggested Performance Improvements

- Use Python's built-in sum() function.
- Replace manual looping with a **generator expression**.
- Reduce interpreter overhead by avoiding explicit variable updates.
- (Optional) Use NumPy for vectorized operations if working with very large numerical datasets.

4. Trade-Offs: Readability vs Performance

- The optimized version is shorter and more readable.
- Built-in functions improve both clarity and speed.
- NumPy-based solutions may offer better performance but reduce readability and add dependencies.
- For most applications, the generator-based approach provides the best balance.