

# Code Review and Quality: Using AI to Improve Code Quality and Readability

2303A51944

Lab-10.3

## Problem Statement 1: AI-Assisted Bug Detection Given

### Code

```
def factorial(n):  
    result = 1  
    for i in  
        range(1, n):  
            result = result * i  
    return result  
Testing  
print(factorial(5))
```

### Output:

24

### Issue Identified

The function contains an **off-by-one error**.

The loop range(1, n) stops at n-1, so it does not multiply by n.

### Corrected Code

```
def factorial(n):  
    if n < 0: raise ValueError("Factorial is not defined for negative  
    numbers")  
    if n == 0:
```

```
    return 1
```

```
result = 1
```

```
for i in range(1, n + 1):
```

```
    result *= i return
```

```
result
```

### **Correct Output:**

120

### **Comparison**

#### **Manual Fix**

Fixed range to n+1

No edge case handling

AI improved robustness by handling edge cases.

#### **AI Fix**

Fixed range and added validation

Handles negative & zero cases

### **Problem Statement 2: Improving Readability & Documentation**

#### **Original Code**

```
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
```

## Issues

- Poor function name (calc)
- No documentation
- No exception handling
- No input validation

**Improved Code**

```
def calculate(number1,  
number2, operation):  
    if not  
    isinstance(operation, str):  
        raise TypeError("Operation must be a string")  
  
    if operation == "add":  
        return  
        number1 + number2  
    elif  
    operation == "sub":  
        return number1 - number2  
    elif  
    operation == "mul":  
        return number1 * number2  
    elif  
    operation == "div":  
        if number2 == 0:  
            raise ZeroDivisionError("Cannot divide by zero")  
        return  
        number1 / number2  
    else:
```

```
raise ValueError("Invalid operation")
```

### **Problem Statement 3: Enforcing PEP8 Standards**

**Original Code** def Checkprime(n):

```
    for i in range(2, n):
```

```
        if n % i == 0:
```

```
            return False return
```

```
    True
```

#### **PEP8 Violations**

- Function name not in snake\_case
- No input validation
- No docstring

#### **Refactored Code**

```
def check_prime(n):
```

```
    if n <= 1:
```

```
        return False
```

```
    for i in range(2, n):
```

```
        if n % i == 0:
```

```
            return False return
```

```
    True
```

**Problem**

**Statement 4: AI as a**

**Code**            **Reviewer**

**Original**    **Code**    def

```
processData(d):
```

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

### Issues

- Poor naming
- No validation
- No type hints
- No documentation

### Improved Code

```
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
```

```
    ]
```

### Reflection

AI should act as an **assistant**, not a replacement for human reviewers.

It speeds up reviews but human judgment is still essential.

## Problem Statement 5: AI-Assisted Performance Optimization

**Original**      **Code**      def

```
sum_of_squares(numbers):
```

```
    total = 0
    for num in
```

```
        numbers:
```

```
            total += num ** 2
```

```
    return total
```

### Time Complexity

$O(n)$

**Optimized**      **Code**      def

```
sum_of_squares_optimized(numbers):
```

```
    return sum(x * x for x in numbers)
```

### Comparison

**Original**      **Optimized**

Uses manual loop      Uses generator expression

Slightly longer      More concise

Same time complexity      Cleaner implementation

### Trade-off Discussion

- Optimized version improves readability.
- For very large datasets, NumPy can provide further speed improvements.

- Built-in functions are generally faster and more Pythonic.