ASSIGNMENT – 10.3

Name: O. NARASIMHA RAJU
HT No: 2303A53034
Batch: 46

Problem Statement 1: AI-Assisted Bug Detection

Given Code

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

Test Result

```python
factorial(5)  # Output: 24 (Incorrect)
```

Bug Identification

- The loop runs from 1 to $n - 1$
- The value n is never multiplied
- This is an off-by-one error

Corrected Code

```python
def factorial(n):
    if n < 0:
        raise ValueError("Factorial is not defined for negative
            numbers")
    result = 1
    for i in range(1, n + 1):
        result *= i
    return result
```

Correct Output

```
factorial(5)  # Output: 120
```

Comparison (AI vs Manual Fix)

- AI correctly identified the off-by-one error
- AI handled edge cases such as negative numbers and zero
- Manual fixes often miss validation; AI included it

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 Identified

- Poor function and parameter names
- No documentation
- No error handling
- Division by zero not handled

Improved Code

```
def calculate(a: float, b: float, operation: str) -> float:
    """
    Performs arithmetic operations on two numbers.
    """
    if operation == "add":
        return a + b
    elif operation == "sub":
        return a - b
    elif operation == "mul":
        return a * b
    elif operation == "div":
        if b == 0:
            raise ValueError("Division by zero is not allowed")
        return a / b
    else:
        raise ValueError("Invalid operation")
```

Testing

- Valid input works correctly
- Division by zero raises an exception
- Invalid operations are handled safely

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
- Missing docstring

Refactored PEP8-Compliant Code

```
def check_prime(n: int) -> bool:
    """
    Checks whether a number is prime.
    """
    if n <= 1:
        return False
    for i in range(2, n):
        if n % i == 0:
            return False
    return True
```

Reflection

AI-based reviews quickly catch style and naming issues, reducing review time and improving consistency in large teams.

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 Found

- Unclear function name
- No validation
- No type hints
- Fails on non-list inputs

Improved Code

```
from typing import List, Union

def double_even_numbers(numbers: List[Union[int, float]]) -> List
   [Union[int, float]]:
    """
    Doubles all even numbers in a list.
    """
    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 standalone reviewer. Human judgment is still required for design decisions.

Problem Statement 5: AI-Assisted Performance Optimization

Original Code

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

Time Complexity

$$O(n)$$

Optimized Version

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

Performance Comparison

- Reduced loop overhead
- Generator expressions
- Built-in `sum()` optimization

Trade-Off Discussion

- Readability is preserved
- Performance is improved
- Suitable for large datasets

Conclusion

AI tools significantly improve:

- Code correctness
- Readability
- Performance
- Compliance with standards

When combined with human review, they produce high-quality, maintainable code.