LAB ASSIGNMENT – 10.3

B.AKARSH
2303A53018
BATCH - 46

Lab 9 – Code Review and Quality: Using AI to improve code

quality and readability

PROMPT:

```
# display and run a code to test its factorial
```

CODE:

```
def factorial(n):
    """Calculate the factorial of n"""
    if n < 0:
        raise ValueError("Factorial not defined for negative
    if n == 0 or n == 1:
        return 1
    return n * factorial(n - 1)


# Test cases
if __name__ == "__main__":
    test_cases = [ 5]

    for num in test_cases:
        result = factorial(num)
        print(f"factorial({num}) = {result}")
```

OUTPUT:

```
factorial(5) = 120
```

EXPLANATION:

The factorial code had an off-by-one error. It skipped multiplying the last number. AI detected the mistake quickly. The loop was fixed using n+1.Edge cases like zero were handled.

Problem Statement 2: Task 2 — Improving Readability &

Documentation

PROMPT:

```
# Refactor this Python function to be readable, well-documented, and robust with error handling and input validation: 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`
```

CODE:

```python
def calculate(num1: float, num2: float, operation: str) -> float:
    """
    Perform arithmetic operations on two numbers.

    Args:
        num1: First operand
        num2: Second operand
        operation: Operation to perform ('add', 'sub', 'mul', 'div')

    Returns:
        Result of the arithmetic operation

    Raises:
        ValueError: If operation is invalid or division by zero occurs
        TypeError: If operands are not numeric
    """
    # Validate input types
    if not isinstance(num1, (int, float)) or not isinstance(num2, (int, float)):
        raise TypeError("Operands must be numeric values")

    # Validate operation
    operations = {
        "add": lambda a, b: a + b,
        "sub": lambda a, b: a - b,
        "mul": lambda a, b: a * b,
        "div": lambda a, b: a / b if b != 0 else (_ for _ in ()).throw(ValueError("Cannot divide by zero"))
    }

    if operation not in operations:
        raise ValueError(f"Invalid operation '{operation}'. Choose from: {', '.join(operations.keys())}")

    return operations[operation](num1, num2)
```

OUTPUT:

```
PS C:\Users\anagandula gouthami\OneDrive\Desktop\AI> 5 + 6
11
```

EXPLANATION:

The code had unclear names. AI suggested better naming. A full docstring was added.Errors were handled safely. Code became more readable.

Problem Statement 3: Enforcing Coding Standards

PROMPT:

```
#Refactor and review this function with better naming, validation, and type hints: def processData(d): return [x*2 for x in d if x%2==0]
```

CODE:

```
def check_prime(n):
    for i in range(2, n):
        if n % i == 0:
            return False
    return True
print(check_prime(17))
```

OUTPUT:

```
ami/OneDrive/D
True
```

EXPLANATION:

The function broke PEP8 rules.AI listed all violations.Naming and spacing were fixed.Logic stayed the same. Code became standard.

Problem Statement 4: AI as a Code Reviewer in Real Projects

PROMPT:

```
# Refactor and review this function with better naming, validation, and type hints: def processData(d): return [x*2 for x in d if x%2==0]
```

CODE:

```python
from typing import List
def process_data(data: List[int]) -> List[int]:
    if not all(isinstance(x, int) for x in data):
        raise ValueError("All elements in the input list must be integers.")
    return [x * 2 for x in data if x % 2 == 0]
print(process_data([1, 2, 3, 4, 5, 6]))
```

OUTPUT:

```
PS C:\Users\akank\AI CODING> & C:/Users/akank/AppData/Local/Programs/Python/Python311/python.exe "c:/Users/aka
[4, 8, 12]
PS C:\Users\akank\AI CODING>
```

EXPLANATION:

The function name was unclear. AI improved clarity and safety. Type hints were added.Logic was generalized. AI helps reviewers

Problem Statement 5: — AI-Assisted Performance Optimization

PROMPT:

```
Make this python function more faster and efficient
```

CODE:

```python
def sum_of_squares(numbers):
    total = 0
    for num in numbers:
        total += num ** 2
    def sum_of_squares(numbers):
        return sum(num ** 2 for num in numbers)
    return total
print(sum_of_squares([1, 2, 3, 4, 5]))
```

OUTPUT:

EXPLANATION:

The loop was slow for big data. AI analyzed complexity. It used sum() for speed. Code ran faster. Readability vs speed was balanced