AI Assisted Coding Assessment (Lab 10.3).

A .Vasantha Shoba Rani

2303A51395

BT.NO: 06

Problem Statement 1: AI-Assisted Bug Detection

Question

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
```

Tasks:

1. Identify the logical bug in the code.
2. Explain why the bug occurs.
3. Provide a corrected version.
4. Compare AI-corrected code with a manual fix.
5. Discuss edge cases (zero and negative numbers).

Python Code

```
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
```

Test

print(factorial(5))

Output

120

Code Explanation

- The bug was an **off-by-one error**.
- range(1, n) stops at n-1, so n was never multiplied.
- The corrected loop uses range(1, n + 1).
- AI also suggested handling **negative values**, which improves robustness.
- factorial(0) correctly returns 1, which is mathematically valid.

Problem Statement 2: Improving Readability and Documentation

Question

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":
```

Tasks:

1. Critique readability and naming.
2. Rewrite using descriptive names and docstrings.
3. Add exception handling and input validation.
4. Test valid and invalid inputs.

Python Code

```python
def calculate(number1, number2, operation):
    """
    Performs a basic arithmetic operation.

    Parameters:
    number1 (float): First number
    number2 (float): Second number
    operation (str): 'add', 'sub', 'mul', or 'div'

    Returns:
    float: Result of the operation

    Raises:
    ValueError: If operation is invalid
    ZeroDivisionError: If division by zero occurs
    """

    if not isinstance(operation, str):
        raise ValueError("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("Division by zero is not allowed")
        return number1 / number2
    else:
```

```
        raise ValueError("Invalid operation")



# Tests
print(calculate(10, 5, "add"))
print(calculate(10, 5, "div"))
```

Output

15

2.0

Code Explanation

- Descriptive function and parameter names improve readability.
- A **docstring** explains purpose, parameters, return value, and errors.
- Input validation prevents unexpected failures.
- Division by zero is handled explicitly.
- AI-assisted refactoring made the function more maintainable.

Problem Statement 3: Enforcing Coding Standards (PEP8)

Question

Original code:

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

Tasks:

1. Identify PEP8 violations.
2. Refactor the code.
3. Verify functionality.
4. Explain AI-assisted code review benefits.

Python Code

```
def check_prime(n):
    if n <= 1:
        return False

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



# Tests
print(check_prime(7))
print(check_prime(10))
```

Output

```
True
False
```

Code Explanation

- Function name changed to snake_case.
- Added validation for numbers less than or equal to 1.
- Proper spacing and indentation applied.
- AI tools quickly detect PEP8 issues and reduce review time in large teams.

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

Question

Original code:

```
def processData(d):
    return [x * 2 for x in d if x % 2 == 0]
```

Tasks:

1. Review readability and edge cases.
2. Add validation and type hints.
3. Improve clarity and reusability.

Python 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")

    result = []
    for num in numbers:
        if isinstance(num, (int, float)) and num % 2 == 0:
            result.append(num * 2)

    return result



# Tests
print(double_even_numbers([1, 2, 3, 4, 6]))
```

Output

```
[4, 8, 12]
```

Code Explanation

- Function name clearly states intent.
- Type hints improve clarity and tooling support.
- Input validation avoids runtime errors.

- AI is best used as an **assistant**, not a replacement for human judgment.

Problem Statement 5: AI-Assisted Performance Optimization

Question

Original code:

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

Tasks:

1. Analyze time complexity.
2. Optimize performance.
3. Compare readability and speed.

Python Code

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



# Test
print(sum_of_squares_optimized(range(10)))
```

Output

285

Code Explanation

- Time complexity remains **O(n)**.
- Generator expressions reduce memory overhead.
- Built-in sum() is faster and more readable.

- AI helped balance **performance and simplicity**.