AIAC-lab10.3

2303A51494

B27

Problem Statement 1: AI-Assisted Bug Detection

CODE AND OUTPUT:

Corrected code:

```
7
8   def factorial(n):
9       result = 1
10      for i in range(1, n + 1):  # Fixed: Changed range(1, n) to range(1, n + 1)
11          result = result * i
12      return result
13
14  print(factorial(5))  # Now correctly outputs 120
```

Output:

```
PS C:\Users\Pavani\OneDrive\Documents\AI Assisted coding> & C:/Users/Pavani/AppData/Local/P
sers/Pavani/OneDrive/Documents/AI Assisted coding/codereview.py"
120
PS C:\Users\Pavani\OneDrive\Documents\AI Assisted coding>
```

Explanation:

Comparison of AI's Corrected Code vs. User's Manual Fix

- The AI's corrected code and the user's manual fix are **identical**:
- Both change range(1 to n) to range(1,n+1) in the loop.
- The function now correctly calculates factorial for positive integers (e.g. factorial(5) returns 120).
- **Zero (n==0)**: Both versions handle it correctly (returns 1, as 0! = 1). No issue missed.
- **Negative numbers (n<0)**: Both versions return 1 (since the range is empty), but this is mathematically incorrect (factorial is undefined for negatives). The AI missed adding input validation (e.g., raising a ValueError for negatives), as the core fix only addressed the loop range. If robustness is needed, validation should be added, but it wasn't part of the original bug fix.

Problem Statement 2: Task 2 — Improving Readability & Documentation

CODE:

```
16 def calc(a, b, c):
17     if c == "add":
18         return a + b
19     elif c == "sub":
20         return a - b
21     elif c=="mul":
22         return a*b
23     elif c=="div":
```

OUTPUT:

```
PS C:\Users\Pavani\OneDrive\Documents\AI Assisted coding> ^C
PS C:\Users\Pavani\OneDrive\Documents\AI Assisted coding> & C:/Users/Pavani/AppData/Local/Microsoft/WindowsApps/python3.12.exe "c:/Users/Pavani/OneDrive/Documents/AI Assisted coding/codereview.py"
  File "c:\Users\Pavani\OneDrive\Documents\AI Assisted coding\codereview.py", line 23
    elif c=="div":
IndentationError: expected an indented block after 'elif' statement on line 23
PS C:\Users\Pavani\OneDrive\Documents\AI Assisted coding>
```

Corrected code:

```python
def calculate(num1, num2, operation):
    """
    Perform a basic arithmetic operation on two numbers.

    Parameters:
    num1 (float): The first number.
    num2 (float): The second number.
    operation (str): The operation to perform. Can be "add", "subtract", "multiply", or "divide".

    Returns:
    float: The result of the arithmetic operation.

    Raises:
    ValueError: If the operation is not one of the specified strings.
    ZeroDivisionError: If the operation is "divide" and num2 is zero.
    """
    if operation == "add":
        return num1 + num2
    elif operation == "subtract":
        return num1 - num2
    elif operation == "multiply":
        return num1 * num2
    elif operation == "divide":
        if num2 == 0:
            raise ZeroDivisionError("Cannot divide by zero.")
        return num1 / num2
    else:
        raise ValueError("Invalid operation. Please choose 'add', 'subtract', 'multiply', or 'divide'.")
# Example usage:
try:
    result = calculate(10, 5, "divide")
    print(result)
except ValueError as ve:
    print(ve)
except ZeroDivisionError as zde:
    print(zde)
```

Output:

```
PS C:\Users\Pavani\OneDrive\Documents\AI Assisted coding> & C:/Users/Pavani/AppData/Local/Microsoft/WindowsApps/python3.12.exe "c:/Users/Pavani/OneDrive/Documents/AI Assisted coding/codereview.py"
2.0
PS C:\Users\Pavani\OneDrive\Documents\AI Assisted coding>
```

Explanation:

- The original function was renamed to "calculate" to better reflect its purpose
- The parameter names were changed to "num1", "num2", and "operation" for clarity
- A comprehensive docstring was added to explain the function's parameters, return value, and potential exceptions
- Exception handling was implemented to catch division by zero errors and invalid operations, providing informative error messages.

Problem Statement 3: Enforcing Coding Standards

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

print(Checkprime(2))
print(Checkprime(3))
print(Checkprime(4))
print(Checkprime(5))
```

OUTPUT:

```
PS C:\Users\Pavani\OneDrive\Documents\AI Assisted coding> & C:/Users/Pavani/AppData/Local/Microsoft/WindowsApps/python3.12.exe "c:/Users/Pavani/OneDrive/Document
w.py"
True
True
False
True
PS C:\Users\Pavani\OneDrive\Documents\AI Assisted coding>
```

Correct code:

```
def check_prime(n):
    """Check if a number is prime."""
    if n <= 1:
        return False
    for i in range(2, int(n**0.5) + 1):
        if n % i == 0:
            return False
    return True
# Sample inputs to verify the function works correctly
print(check_prime(2))  # True, 2 is a prime number
print(check_prime(3))  # True, 3 is a prime number
print(check_prime(4))  # False, 4 is not a prime number
print(check_prime(5))  # True, 5 is a prime number
```

OUTPUT:

```
PS C:\Users\Pavani\OneDrive\Documents\AI Assisted coding> & C:/Users/Pavani/AppData/Local/Microsoft/WindowsApps/python3.
w.py"
True
True
False
True
PS C:\Users\Pavani\OneDrive\Documents\AI Assisted coding>
```

Explanation:

- The original function name "Checkprime" was changed to "check_prime" to follow the PEP8 naming convention for functions.
- A docstring was added to explain the purpose of the function

- A check was added to return False for numbers less than or equal to 1, as they are not prime numbers.
- The loop was optimized to check for factors only up to the square root of n, improving efficiency.

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

Code:

```
def processData(d):
    return [x * 2 for x in d if x % 2 == 0]
# Sample input to verify the function works correctly
print(processData([1, 2, 3, 4, 5, 6]))  # Output: [4, 8, 12]
```

OUTPUT:

```
w.py
PS C:\Users\Pavani\OneDrive\Documents\AI Assisted coding> & C:/Users/Pavani/AppData/Local/Microsoft/WindowsApps/python3.12.exe "c:/Users/
w.py"
[4, 8, 12]
PS C:\Users\Pavani\OneDrive\Documents\AI Assisted coding>
```

Corrected code:

Code:

```
def process_data(data: list) -> list:
    """
    Process a list of integers by doubling the even numbers.

    Parameters:
    data (list): A list of integers to be processed.

    Returns:
    list: A new list containing the doubled values of the even numbers from the input list.

    Raises:
    TypeError: If the input is not a list or contains non-integer elements.
    """
    if not isinstance(data, list):
        raise TypeError("Input must be a list.")

    for item in data:
        if not isinstance(item, int):
            raise TypeError("All elements in the list must be integers.")

    return [x * 2 for x in data if x % 2 == 0]
# Sample input to verify the function works correctly
print(process_data([1, 2, 3, 4, 5, 6]))  # Output: [4, 8, 12]
```

Output:

```
[4, 8, 12]
PS C:\Users\Pavani\OneDrive\Documents\AI Assisted coding> & C:/Users/Pavani/AppData/Local/Microsoft
w.py"
[4, 8, 12]
PS C:\Users\Pavani\OneDrive\Documents\AI Assisted coding>
```

Explanation:

- AI can serve as a valuable assistant in the code review process, providing quick feedback on syntax, style, and potential errors.
- However, it should not be a standalone reviewer.
- Human reviewers bring context, creativity, and a deeper understanding of the project and its goals, which AI currently lacks
- AI can help identify common issues and enforce coding standards, but human judgment is essential for evaluating the overall design, architecture, and maintainability of the code.
- AI should be used as a tool to complement human reviewers rather than replace them.

Problem Statement 5: — AI-Assisted Performance Optimization

CODE:

```
175
176  def sum_of_squares(numbers):
177      total = 0
178      for num in numbers:
179          total += num ** 2
180      return total
181  print(sum_of_squares([1, 2, 3]))
```

OUTPUT:

```
PS C:\Users\Pavani\OneDrive\Documents\AI Assisted coding> & C:/Users/Pavani/AppData/Local/Microsoft/WindowsApps/python3.12.exe "c:/Users/Pavani/OneDri
w.py"
14
PS C:\Users\Pavani\OneDrive\Documents\AI Assisted coding>
```

Correct code:

```python
#Suggest performance improvements (e.g., using built-in functions, vectorization with NumPy if applicable).
def sum_of_squares(numbers):
    """
    Calculate the sum of squares of a list of numbers.

    Parameters:
    numbers (list): A list of numbers to be processed.

    Returns:
    int: The sum of squares of the input numbers.

    Raises:
    TypeError: If the input is not a list or contains non-numeric elements.
    """
    if not isinstance(numbers, list):
        raise TypeError("Input must be a list.")

    for item in numbers:
        if not isinstance(item, (int, float)):
            raise TypeError("All elements in the list must be numeric.")

    return sum(x ** 2 for x in numbers)
# Sample input to verify the function works correctly
print(sum_of_squares([1, 2, 3]))
```

Output:

```
  w.py"
PS C:\Users\Pavani\OneDrive\Documents\AI Assisted coding> & C:/Users/Pavani/AppData/Local/Microsoft/WindowsApps/python3.12.exe "c:/Users/Pavani/OneDrive/Documents/AI Assisted coding
  w.py"
  14
PS C:\Users\Pavani\OneDrive\Documents\AI Assisted coding>
```

Explanation:

- The original function used a for loop to iterate through the list and calculate the sum of squares, which is straightforward but can be less efficient for large lists.
- The refactored version uses a generator expression within the built-in sum() function
- Which is more efficient and concise
- This approach avoids the overhead of creating an intermediate list and allows for faster computation, especially with larger datasets
- Input validation was added to ensure that the function handles incorrect input gracefully.