ASSIGNMENT-10

Roll no:2303A52074
Batch-37

Problem Statement 1: AI-Assisted Bug Detection

Prompt

Identify the logical bug in the code and explain why the bug occurs and give a corrected version code.

Code

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

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

#Identify the logical bug in the code and explain why the bug occurs and give a corrected version code.
The logical bug in the code is an off-by-one error in the loop.
The loop iterates from 1 to n-1, which means it does not include n in the calculation of the factorial.
The correct implementation should include n in the loop.
def factorial(n):
    result = 1
    for i in range(1, n + 1):
        result = result * i
    return result
# Alternatively, we can also use the built-in math library to calculate the factorial:
import math
def factorial(n):
    return math.factorial(n)
# Example usage:
print(factorial(5))  # Output: 120
```

Output

```
120
```

Explanation

Ai fixed the logical bug in the given code and gave corrected version it explained what is the logical error and In the original function, the mistake happens because the loop does not include the number n itself. It only multiplies numbers from 1 up to n-1.

Problem Statement 2: Task 2 — Improving Readability & Documentation

Prompt

Please review the code and suggest improvements to make it more readable .Improve the function name and parameter names, add proper documentation, handle division by zero,and include input validation. Make the final version clean, well-structured, and easy to maintain.

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":
```

```
# Please review the code and suggest improvements to make it more readable .Improve the function name and parameter names, add proper do
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', 'sub', 'mul', or 'div'.

    Returns:
    float: The result of the arithmetic operation.

    Raises:
    ValueError: If the operation is not one of the specified options.
    ZeroDivisionError: If division by zero is attempted.
    """

    if operation == "add":
        return num1 + num2
    elif operation == "sub":
        return num1 - num2
    elif operation == "mul":
        return num1 * num2
    elif operation == "div":
        if num2 == 0:
            raise ZeroDivisionError("Cannot divide by zero.")
        return num1 / num2
    else:
        raise ValueError("Invalid operation. Please choose 'add', 'sub', 'mul', or 'div'.")
# Example usage:
try:
    result = calculate(10, 5, "div")
```

```
    if operation == "add":
        return num1 + num2
    elif operation == "sub":
        return num1 - num2
    elif operation == "mul":
        return num1 * num2
    elif operation == "div":
        if num2 == 0:
            raise ZeroDivisionError("Cannot divide by zero.")
        return num1 / num2
    else:
        raise ValueError("Invalid operation. Please choose 'add', 'sub', 'mul', or 'div'.")
# Example usage:
try:
    result = calculate(10, 5, "div")
    print(result)  # Output: 2.0
except (ValueError, ZeroDivisionError) as e:
    print(e)
```

Output

Explanation

In the improved version, the function name calculate and parameter names like num1, num2, and operation make the code easier to understand. A proper docstring was added to clearly describe what the function does, its parameters, return value, and possible errors. Error handling for invalid operations and division by zero makes the function more robust. Overall, the updated version is more readable.

Problem Statement 3: Enforcing Coding Standards

Prompt

Look at the following Python function and check if it follows PEP8 coding standards. #Point out any problems with naming, formatting, or missing comments.
Rewrite the function so it follows proper coding styleand improves readability. Also handle any possible edge cases if needed.

Code

```
def Checkprime(n):
    for i in range(2, n):
        if n % i == 0:
            return False
    return True
#Look at the following Python function and check if it follows PEP8 coding standards.
#Point out any problems with naming, formatting, or missing comments.
#Rewrite the function so it follows proper coding styleand improves readability. Also handle any possible edge cases if needed.
def is_prime(n):
    """
    Check if a number is prime.

    Parameters:
    n (int): The number to check.

    Returns:
    bool: True if the number is prime, False otherwise.
    """
    if n < 2:
        return False
    for i in range(2, int(n ** 0.5) + 1):
        if n % i == 0:
            return False
    return True
# Example usage:
print(is_prime(11))  # Output: True
print(is_prime(15))  # Output: False
```

Output

```
    """
    Check if a number is prime.

    Parameters:
    number (int): The number to check for primality.

    Returns:
    bool: True if the number is prime, False otherwise.
    """

    if number <= 1:
        return False
    for i in range(2, int(number**0.5) + 1):
        if number % i == 0:
            return False
    return True
# Example usage:
print(check_prime(11))  # Output: True
print(check_prime(15))  # Output: False
```

Explanation

The original function did not follow PEP8 because the function name was not written in snake_case and there were no comments or documentation. It also did not handle special cases like numbers less than or equal to 1.
In the improved version, the function name was changed to check prime and a proper docstring was added. The code now handles edge cases and is easier to read. It also checks divisibility more efficiently, making the function cleaner and better written.

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

Prompt

Review the following Python function and suggest improvements to make it more readable.
Improve naming, add type hints and input validation, and handle possible edge cases.Then rewrite the function

Code

```
def processData(d):
    return [x * 2 for x in d if x % 2 == 0]
#Review the following Python function and suggest improvements to make it more readable.
#Improve naming, add type hints and input validation, and handle possible edge cases.Then rewrite the
function
def process_data(data: list) -> list:
    """
    Process a list of numbers 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:
    ValueError: If the input is not a list or contains non-integer values.
    """

    if not isinstance(data, list):
        raise ValueError("Input must be a list.")

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

    return [x * 2 for x in data if x % 2 == 0]
# Example usage:
try:
    result = process_data([1, 2, 3, 4, 5])
    print(result)  # Output: [4, 8]
except ValueError as e:
    print(e)
```

Output

```python
def processData(d):
    return [x * 2 for x in d if x % 2 == 0]
#Review the following Python function and suggest improvements to make it more readable.
#Improve naming, add type hints and input validation, and handle possible edge cases.Then rewrite the function
def process_data(data: list) -> list:
    """
    Process a list of numbers 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:
    ValueError: If the input is not a list or contains non-integer values.
    """

    if not isinstance(data, list):
        raise ValueError("Input must be a list.")

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

    return [x * 2 for x in data if x % 2 == 0]
# Example usage:
try:
    result = process_data([1, 2, 3, 4, 5])
    print(result)  # Output: [4, 8]
except ValueError as e:
    print(e)
```

Explanation

The original function worked but was not very clear and had no input checking.
The improved version uses a better function name, adds documentation, and checks that the input is valid.
This makes the code easier to understand and safer to use.

Problem Statement 5: — AI-Assisted Performance Optimization

Prompt

Analyze the following Python function for performance.
Explain its time complexity and suggest possible optimizations.
Provide an improved version using built-in functions or other efficient methods

Code

```python
def sum_of_squares(numbers):
    total = 0
    for num in numbers:
        total += num ** 2
    return total
#Analyze the following Python function for performance.
#Explain its time complexity and suggest possible optimizations.
#Provide an improved version using built-in functions or other efficient methods
def sum_of_squares(numbers):
    """
    Calculate the sum of squares of a list of numbers.

    Parameters:
    numbers (list): A list of integers or floats.

    Returns:
    float: The sum of squares of the input numbers.
    """

    return sum(num ** 2 for num in numbers)
# Example usage:
print(sum_of_squares([1, 2, 3]))  # Output: 14
```

Output

```
14
```

Explanation

The function goes through each number in the list and adds its square.
It runs once through the list, so the time complexity is O(n).
The improved version uses sum() to make the code shorter and cleaner.
Both versions work the same, but the new one is easier to read