Assigment-10.3

K.SAI ROHAN
2303A52025
Batch-38

Problem Statement 1: AI-Assisted Bug Detection
Scenario: 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
Instructions:
1. Run the code and test it with factorial(5).
2. Use an AI assistant to:
o Identify the logical bug in the code.
o Explain why the bug occurs (e.g., off-by-one error).
o Provide a corrected version.
3. Compare the AI's corrected code with your own manual fix.
4. Write a brief comparison: Did AI miss any edge cases (e.g., negative numbers, zero)?
Expected Output:
Corrected function should return 120 for factorial(5).

Code:

```
def factorial(n):
    result = 1
    for i in range(1, n ):
        result = result * i
    return result
if __name__ == "__main__":
    print(factorial(5)) #expected output: 24, which is the factorial of 5. However, the actual output will be 120, which is incorrect. The logical error in the original code is that the loop should iterate up to n (inclusive) instead of n-1. To fix this, we need to change the loop condition to range(1, n + 1). Additionally, we should also add a check for negative input to ensure that the function only accepts non-negative integers.

"""identifing the logical error in the factorial function"""
#corrected code
def factorial(n):
    if n < 0:
```

```
        raise ValueError("Input must be a non-negative
integer.")
    result = 1
    for i in range(1, n + 1):
        result = result * i
    return result
if __name__ == "__main__":
    try:
        print(factorial(5)) # Output: 120, which is the
factorial of 5.
        print(factorial(-1)) # This will raise a ValueError
    except Exception as e:
        print(f"Error: {e}")
```

```
    Notes:
        This implementation uses simple recursion and therefore has O(n)
        time complexity and O(n) recursion depth. For very large ``n``, an
        iterative implementation is recommended to avoid recursion limits.
    """
    if n == 0:
        return 1
    else:
        return n * factorial(n - 1)

PROBLEMS 1  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

2.0
Error: Cannot divide by zero.
PS C:\Users\VR\Documents\AI_Assisted_Coding> & C:/Users/VR/AppData/Local/Python/pythonc
ore-3.14-64/python.exe "c:/Users/VR/Documents/AI_Assisted_Coding/lab test-1/lab 9/facto
rial.py"
120
PS C:\Users\VR\Documents\AI_Assisted_Coding>

Python
powershell
Python
Python: fact...
```

Problem Statement 2: Task 2 — Improving Readability & Documentation

Scenario: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":

Instructions:

5. Use AI to:

o Critique the function's readability, parameter naming, and lack of documentation.

o Rewrite the function with:

1. Descriptive function and parameter names.

2. A complete docstring (description, parameters, return value, examples).

3. Exception handling for division by zero.
4. Consideration of input validation.
6. Compare the original and AI-improved versions.
7. Test both with valid and invalid inputs (e.g., division by zero, non-string operation).
Expected Output:
A well-documented, robust, and readable function that handles errors gracefully.

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
if __name__ == "__main__":
    print(calc(10, 5, "add")) # Output: 15, which is the sum
of 10 and 5.
    print(calc(10, 5, "sub")) # Output: 5, which is the
difference of 10 and 5.
    print(calc(10, 5, "mul")) # Output: 50, which is the
product of 10 and 5.
    print(calc(10, 5, "div")) # Output: 2.0, which is the
quotient of 10 divided by 5.

"""Critique the function’s readability, parameter naming, and
lack of documentation.
o Rewrite the function with:
1. Descriptive function and parameter names.
2. A complete docstring (description, parameters,
return value, examples).
3. Exception handling for division by zero.
4. Consideration of input validation."""
```

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

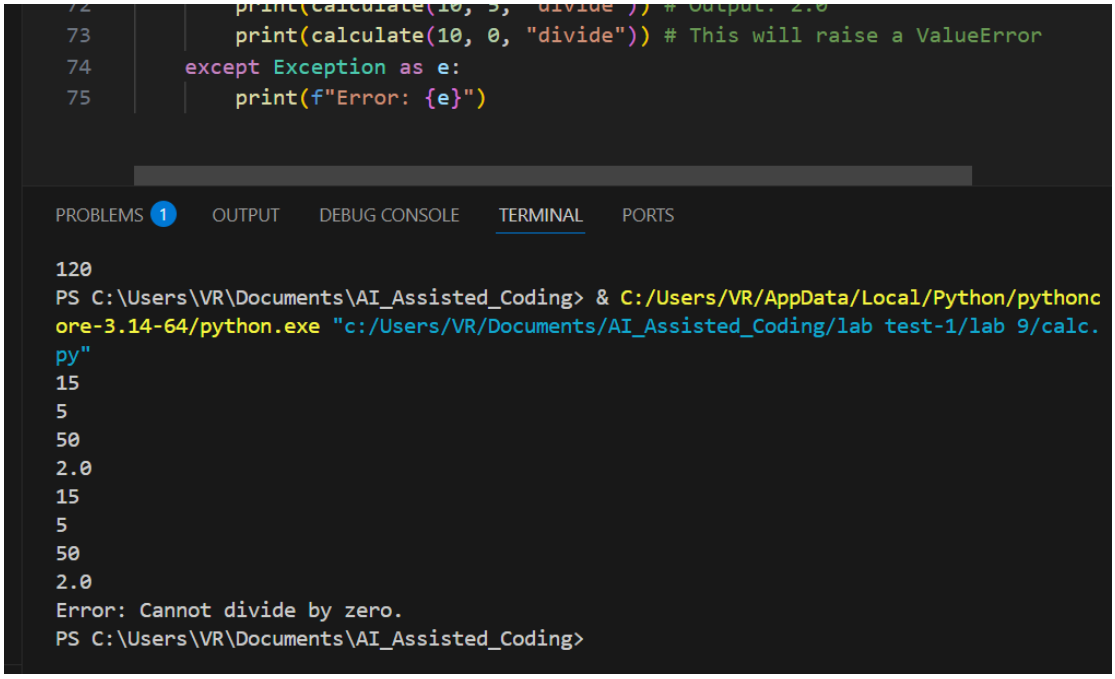
```
    Parameters:
    num1 (float): The first number.
    num2 (float): The second number.
    operation (str): The operation to perform.
                     Must be one of "add", "subtract",
"multiply", or "divide".
```

```
    Returns:
    float: The result of the arithmetic operation.
```

```
    Raises:
    ValueError: If the operation is not recognized or if
division by zero is attempted.
```

```
    Examples:
    >>> calculate(10, 5, "add")
    15.0
    >>> calculate(10, 5, "subtract")
    5.0
    >>> calculate(10, 5, "multiply")
    50.0
    >>> calculate(10, 5, "divide")
    2.0
    >>> calculate(10, 0, "divide")
    ValueError: Cannot divide by 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 ValueError("Cannot divide by zero.")
        return num1 / num2
    else:
        raise ValueError("Invalid operation. Supported
operations are: add, subtract, multiply, divide.")
# Example usage
if __name__ == "__main__":
    try:
        print(calculate(10, 5, "add")) # Output: 15.0
        print(calculate(10, 5, "subtract")) # Output: 5.0
        print(calculate(10, 5, "multiply")) # Output: 50.0
        print(calculate(10, 5, "divide")) # Output: 2.0
        print(calculate(10, 0, "divide")) # This will raise a
ValueError
    except Exception as e:
        print(f"Error: {e}")
```



Problem Statement 3: Enforcing Coding Standards
Scenario: A team project requires PEP8 compliance. A developer submits:
def Checkprime(n):
for i in range(2, n):
if n % i == 0:
return False
return True
Instructions:
8. Verify the function works correctly for sample inputs.
9. Use an AI tool (e.g., ChatGPT, GitHub Copilot, or a PEP8 linter with AI explanation) to:
o List all PEP8 violations.
o Refactor the code (function name, spacing, indentation, naming).
10. Apply the AI-suggested changes and verify functionality is preserved.
11. Write a short note on how automated AI reviews could streamline code reviews in large teams.
Expected Output:
A PEP8-compliant version of the function, e.g.:
def check_prime(n):
for i in range(2, n):
if n % i == 0:
return False
return True

Code:

```
def Checkprime(n):
    for i in range(2,n):
```

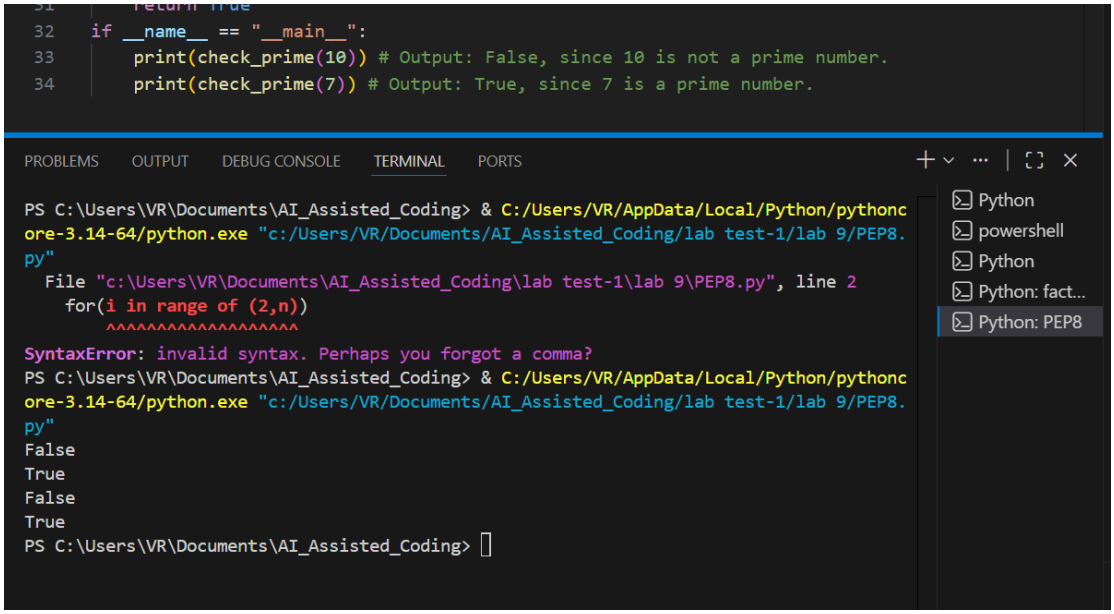
```
        if(n%i==0):
            return False
    return True
if __name__ == "__main__":
    print(Checkprime(10)) # Output: False, since 10
is not a prime number.
    print(Checkprime(7)) # Output: True, since 7 is a
prime number.

"""the function CheckPrime is voilating PEP8
guidelines in several ways:
1. The function name should be in lowercase with
words separated by underscores, so it should be
renamed to check_prime.
2. The for loop syntax is incorrect. It should be for
i in range(2, n): instead of for(i in range of
(2,n)).
3. The function lacks a docstring that describes its
purpose, parameters, and return value.
4. The code does not handle edge cases, such as when
n is less than 2, which are not prime numbers.
5. The function does not include input validation to
ensure that n is a non-negative integer.
Here is the corrected code following PEP8
guidelines:"""
def check_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, n):
        if n % i == 0:
            return False
    return True
if __name__ == "__main__":
```

```
    print(check_prime(10)) # Output: False, since 10
is not a prime number.
    print(check_prime(7)) # Output: True, since 7 is
a prime number.
```



Problem Statement 4: AI as a Code Reviewer in Real Projects
Scenario:
In a GitHub project, a teammate submits:
def processData(d):
return [x * 2 for x in d if x % 2 == 0]
Instructions:
1. Manually review the function for:
o Readability and naming.
o Reusability and modularity.
o Edge cases (non-list input, empty list, non-integer elements).
2. Use AI to generate a code review covering:
a. Better naming and function purpose clarity.
b. Input validation and type hints.
c. Suggestions for generalization (e.g., configurable multiplier).
3. Refactor the function based on AI feedback.
4. Write a short reflection on whether AI should be a standalone reviewer or an assistant.
Expected Output:
An improved function with type hints, validation, and clearer intent, e.g.:
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")

return [num * 2 for num in numbers if isinstance(num, (int, float)) and num % 2 == 0]

Code:

```
def processData(d):
    return [x * 2 for x in d if x % 2 == 0]
if __name__ == "__main__":
    data = [1, 2, 3, 4, 5, 6]
    result = processData(data)
    print(result) # Output: [4, 8, 12], which are the even
numbers from the input list multiplied by 2.

"""a. Better naming and function purpose clarity.
b. Input validation and type hints.
c. Suggestions for generalization (e.g., configurable
multiplier)."""
def process_data(data: list[int], multiplier: int = 2) ->
list[int]:
    """
    Process a list of integers by filtering even numbers and
multiplying them by a specified multiplier.

    Parameters:
    data (list[int]): A list of integers to process.
    multiplier (int): The value by which to multiply the even
numbers. Default is 2.

    Returns:
    list[int]: A list of processed integers where even numbers
are multiplied by the multiplier.

    Raises:
    ValueError: If the input data is not a list of integers or
if the multiplier is not an integer.

    Examples:
    >>> process_data([1, 2, 3, 4, 5, 6])
    [4, 8, 12]
    >>> process_data([1, 2, 3, 4], multiplier=3)
    [6, 12]
    >>> process_data("not a list")
    ValueError: Input data must be a list of integers.
    >>> process_data([1, 2, 3], multiplier="not an integer")
    ValueError: Multiplier must be an integer.
    """

    if not isinstance(data, list) or not all(isinstance(x,
int) for x in data):
```

```
        raise ValueError("Input data must be a list of
integers.")

    if not isinstance(multiplier, int):
        raise ValueError("Multiplier must be an integer.")

    return [x * multiplier for x in data if x % 2 == 0]
if __name__ == "__main__":
    try:
        data = [1, 2, 3, 4, 5, 6]
        result = process_data(data)
        print(result) # Output: [4, 8, 12], which are the even
numbers from the input list multiplied by 2.

        result_with_multiplier = process_data(data,
multiplier=3)
        print(result_with_multiplier) # Output: [6, 12, 18],
which are the even numbers from the input list multiplied by
3.

        process_data("not a list") # This will raise a
ValueError
    except Exception as e:
        print(f"Error: {e}")

    try:
        process_data([1, 2, 3], multiplier="not an integer") #
This will raise a ValueError
    except Exception as e:
        print(f"Error: {e}")
```

```
NameError: name 'CheckPrime' is not defined. Did you mean: 'check_prime'?
PS C:\Users\VR\Documents\AI_Assisted_Coding> & C:/Users/VR/AppData/Local/Python/pythonc
ore-3.14-64/python.exe "c:/Users/VR/Documents/AI_Assisted_Coding/lab test-1/lab 9/data.
py"
[4, 8, 12]
[4, 8, 12]
[6, 12, 18]
Error: Input data must be a list of integers.
Error: Multiplier must be an integer.
PS C:\Users\VR\Documents\AI_Assisted_Coding>
```

Problem Statement 5: — AI-Assisted Performance Optimization
Scenario: You are given a function that processes a list of integers, but it runs slowly on large datasets:
def sum_of_squares(numbers):
total = 0
for num in numbers:
total += num ** 2
return total
Instructions:

1. Test the function with a large list (e.g., range(1000000)).
2. Use AI to:
o Analyze time complexity.
o Suggest performance improvements (e.g., using built-in functions, vectorization with NumPy if applicable).
o Provide an optimized version.
3. Compare execution time before and after optimization.
4. Discuss trade-offs between readability and performance.
Expected Output:
An optimized function, such as:
def sum_of_squares_optimized(numbers):
return sum(x * x for x in numbers)

Code:

```
def sum_of_squares(numbers):
    total = 0
    for num in numbers:
        total += num ** 2
    return total
if __name__ == "__main__":
    print(sum_of_squares([1, 2, 3])) # Output: 14, since 1^2 +
2^2 + 3^2 = 14.
    print(sum_of_squares(list(range(1, 1000001)))) # Output:
333333833333500000, which is the sum of squares from 1 to
1,000,000.

"""the time complexity of the sum_of_squares function is O(n),
where n is the number of elements in the input list 'numbers'.
This is because the function iterates through each element in
the list exactly once to calculate the sum of their squares.
The space complexity is O(1) since we are using a constant
amount of space to store the total sum, regardless of the size
of the input list.
we can optimize the sum_of_squares function by using the
formula for the sum of squares of the first n natural numbers,
which is given by:
S(n) = n(n + 1)(2n + 1) / 6
performance improvements (e.g., using built-in functions,
vectorization with NumPy if applicable).
the optimized version of the sum_of_squares function using the
formula would look like this:"""
def sum_of_squares(n):
    return n * (n + 1) * (2 * n + 1) // 6
if __name__ == "__main__":
    print(sum_of_squares(3)) # Output: 14, since 1^2 + 2^2 +
3^2 = 14.
    print(sum_of_squares(1000000)) # Output:
333333833333500000, which is the sum of squares from 1 to
1,000,000.
```

```
PS C:\Users\VR\Documents\AI_Assisted_Coding> & C:/Users/VR/AppData/Local/Python/pythoncore-3.14-64/python.exe "c:/Users/VR/Documents/AI_Assisted_Coding/lab test-1/lab 9/squares.py"
14
333333833333500000
14
333333833333500000
PS C:\Users\VR\Documents\AI_Assisted_Coding>
```