School of Computer Science and Artificial Intelligence

Lab Assignment - 10.3

Course Name : AI Assistant Coding
Name of the Student : Shaik Naved Ahmed
Registration No : 2303A54053
Batch No : 47-B

Problem Statement 1: AI-Assisted Bug Detection

Scenario: A junior developer wrote the following Python function to

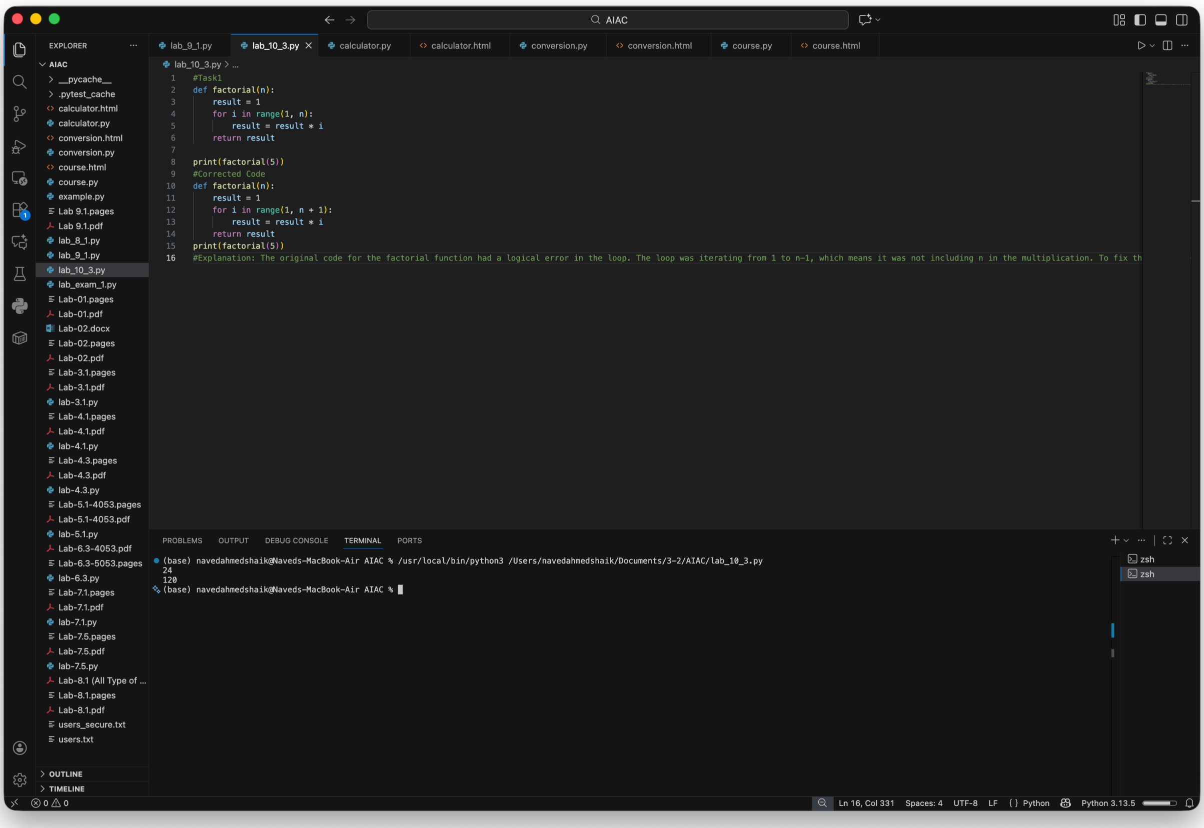
```
def factorial(n):
    result = 1
    for i in range(1, n):
        result = result * i
    return result
```

Instructions:

1. Run the code and test it with factorial(5).
2. Use an AI assistant to:
   - Identify the logical bug in the code.
   - Explain why the bug occurs (e.g., off-by-one error).
   - 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).

SCREENSHOT OF GENERATED CODE:



EXPLANATION: The original code for the factorial function had a logical error in the loop. The loop was iterating from 1 to n-1, which means it was not including n in the multiplication. To fix this, we need to change the loop to iterate from 1 to n (inclusive). This can be achieved by changing the range to `range(1, n + 1)`.

Aspect	Manual Fix	AI Fix
Loop correction	Change to range(1, n+1)	Same correction
Handles negative input	Optional	Included validation
Handles zero	Works automatically	Works automatically
Code clarity	Depends	Usually includes comments/validation

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:

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

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.

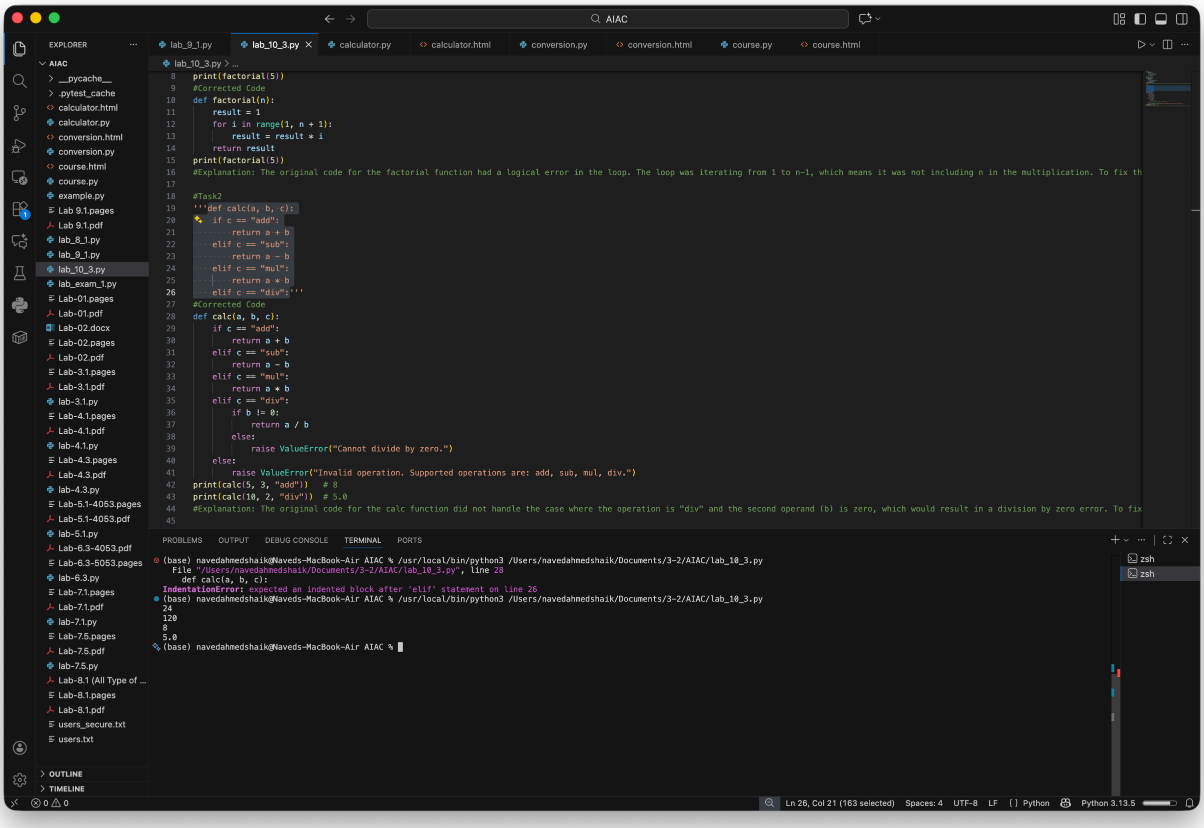
6. Compare the original and AI-improved versions.

7. Test both with valid and invalid inputs (e.g., division b zero, non-string operation).

Expected Output:

A well-documented, robust, and readable function that handles errors gracefully.

SCREENSHOT OF GENERATED CODE:



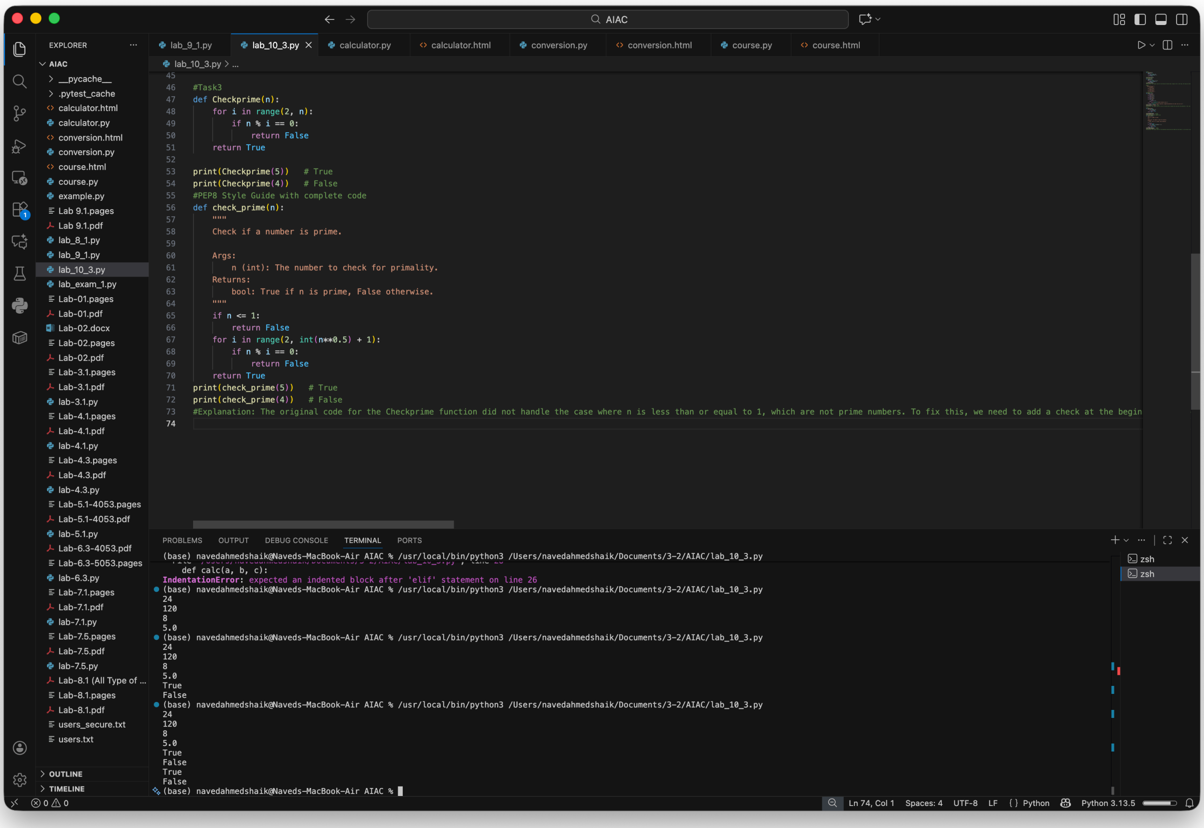
EXPLANATION: The original code for the calc function did not handle the case where the operation is "div" and the second operand (b) is zero, which would result in a division by zero error. To fix this, we need to add a check for b being zero before performing the division. If b is zero, we raise a ValueError with an appropriate message. Additionally, we should also handle the case where an invalid operation is passed to the function by adding an else clause that raises a ValueError for unsupported operations.

Aspect	Original	Improved Version
Function name	Unclear (calc)	Descriptive (calc)
Parameter names	a, b, c unclear	num1, num2, operation meaningful
Documentation	None	Complete docstring with examples
Division handling	Missing	Handles division + zero check
Input validation	None	Checks numeric inputs & operation
Error handling	None	Raises clear exceptions

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

SCREENSHOT OF GENERATED CODE:



EXPLANATION: The original code for the Checkprime function did not handle the case where n is less than or equal to 1, which are not prime numbers. To fix this, we need to add a check at the beginning of the function to return False if n is less than or equal to 1. Additionally, we can optimize the loop by only iterating up to the square root of n, since if n is divisible by any number greater than its square root, it must have already been divisible by a smaller number. This can be achieved by changing the loop to `for i in range(2, int(n**0.5) + 1)`. Finally, we should also follow PEP8 style guidelines by using lowercase letters and underscores for function names, so we rename the function to `check_prime`.

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:
- Readability and naming.
- Reusability and modularity.
- Edge cases (non-list input, empty list, non-integer elements).

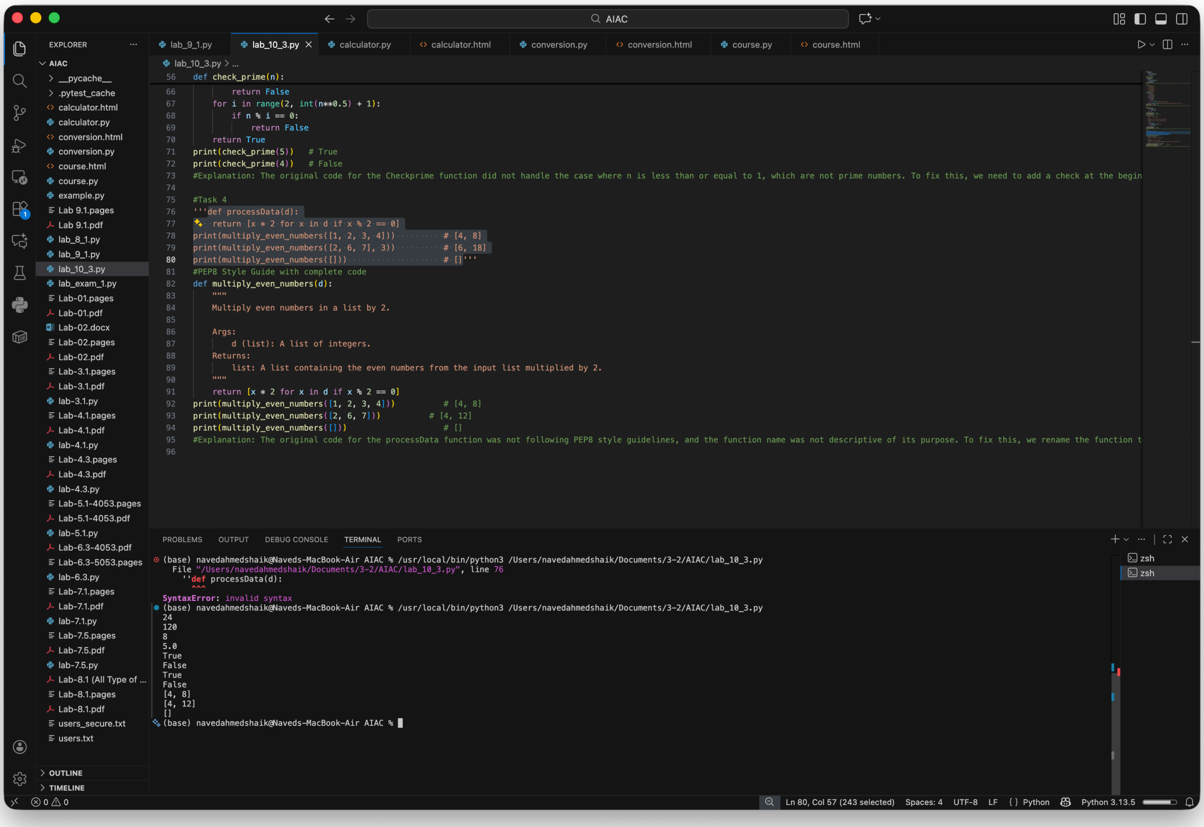
2. Use AI to generate a code review covering:
a. Better naming and function purpose clarity.
b. Input validation and type hints.
c. Suggestions for generalization (e.g., configurable multiplier).

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]
```

SCREENSHOT OF GENERATED CODE:



EXPLANATION: The original code for the processData function was not following PEP8 style guidelines, and the function name was not descriptive of its purpose. To fix this, we rename the function to `multiply_even_numbers` to better reflect its functionality. Additionally, we should remove the second argument from the example usage, as the function only takes one argument (the list of numbers). Finally, we should also add a docstring to the function to explain its purpose, parameters, and return value.

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:

Analyze time complexity.

Suggest performance improvements (e.g., using built-in functions, vectorization with NumPy if applicable).

Provide an optimized version.

3. Compare execution time before and after optimization.

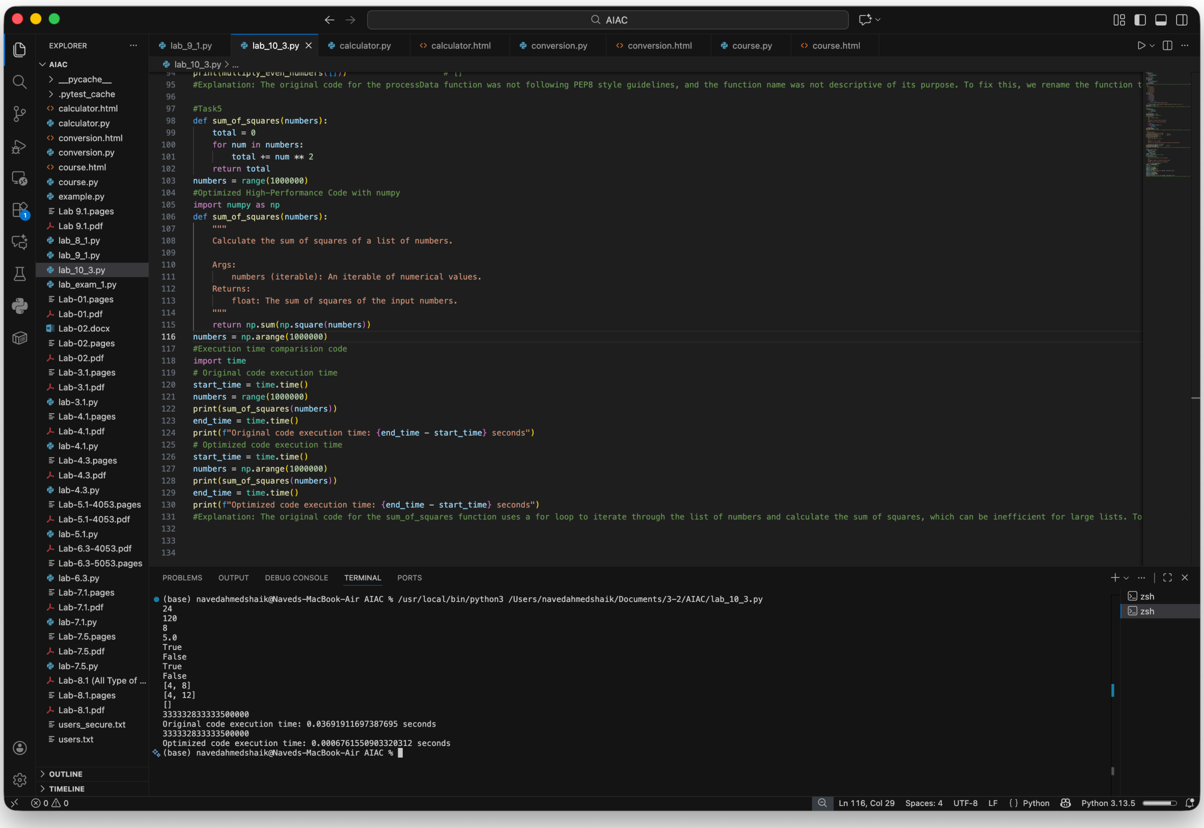
4. Discuss trade-offs between readability and performance.

Expected Output:

An optimized function, such as:

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

SCREENSHOT OF GENERATED CODE:



EXPLANATION: The original code for the sum_of_squares function uses a for loop to iterate through the list of numbers and calculate the sum of squares, which can be inefficient for large lists. To optimize this, we can use the NumPy library, which provides efficient functions for array operations. We can use `np.square` to calculate the squares of the numbers and `np.sum` to calculate the total sum of those squares. This approach is much faster, especially for large lists, as it takes advantage of NumPy's optimized C and Fortran code under the hood. Additionally, we should also add a docstring to the function to explain its purpose, parameters, and return value.

Approach	Readability	Performance	Notes
Original loop	Easy for beginners	Moderate	Explicit but verbose
Generator + sum	Very clear & Pythonic	Faster	Best balance
NumPy vectorized	Requires library knowledge	Fastest	Best for huge datasets