AI ASSISTED CODING

Name: P.Swaran Raj **Roll.No:** 2303A52247

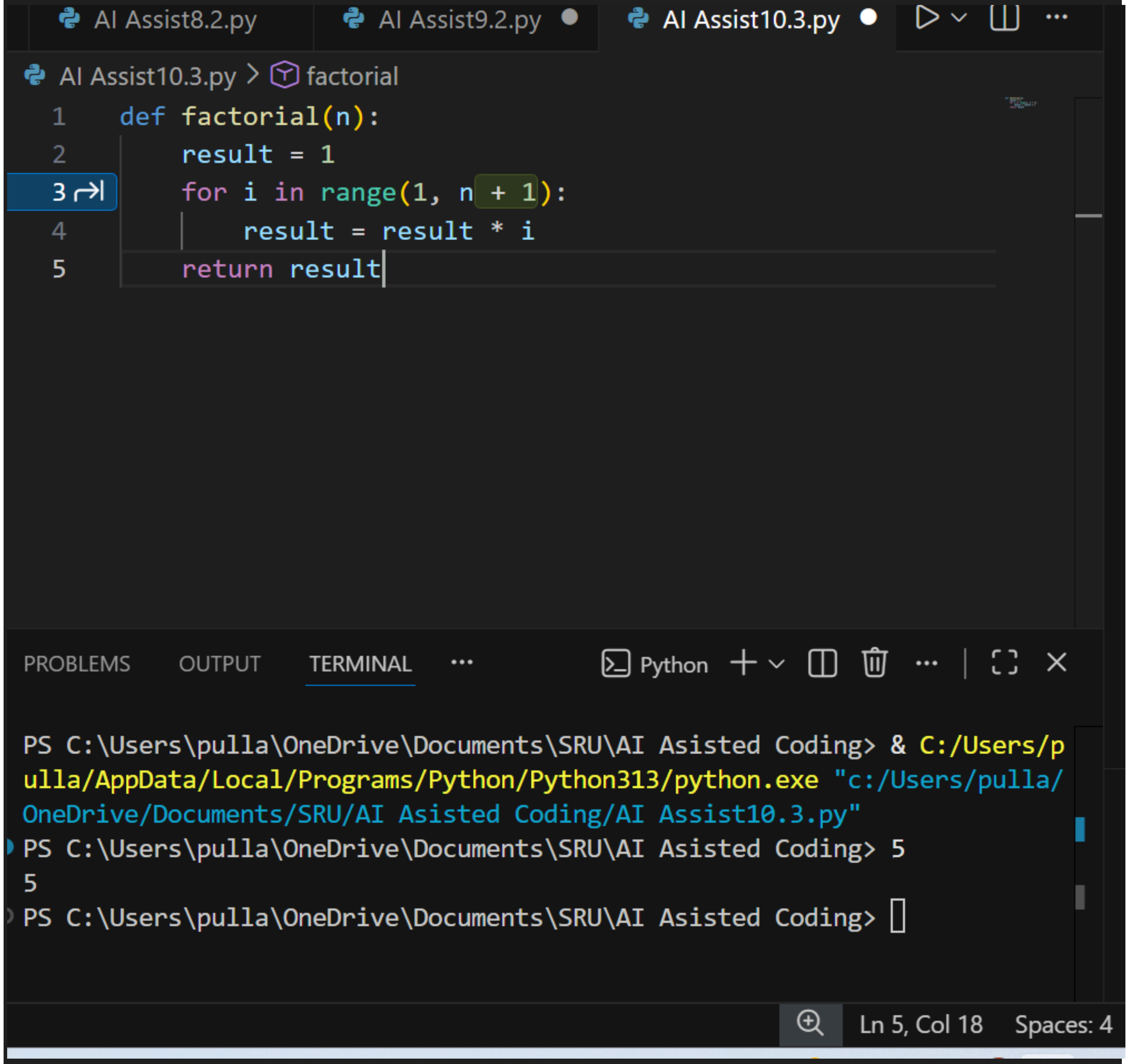
Batch: 37 **Assignment-** 10.3

Problem Statement 1: AI-Assisted Bug Detection

Prompt:

"#identify the logical bug in the following code and fix it"

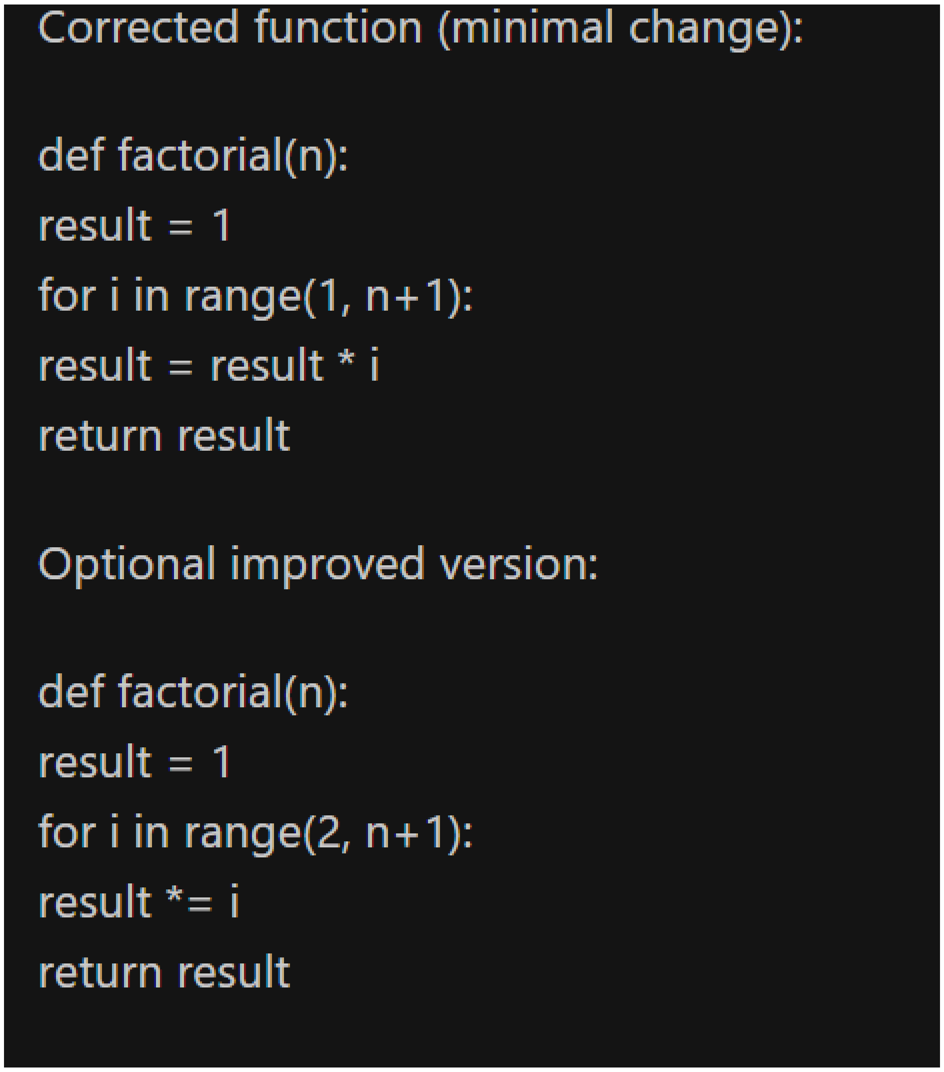
```
def factorial(n):
    result = 1
    for i in range(1, n):
result = result * i
    return result
```



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

```
PS C:\Users\pulla\OneDrive\Documents\SRU\AI Asisted Coding> & C:/Users/pulla/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/pulla/OneDrive/Documents/SRU/AI Asisted Coding/AI Assist10.3.py"
PS C:\Users\pulla\OneDrive\Documents\SRU\AI Asisted Coding> 5
5
PS C:\Users\pulla\OneDrive\Documents\SRU\AI Asisted Coding>
```

AI Fixed Code:



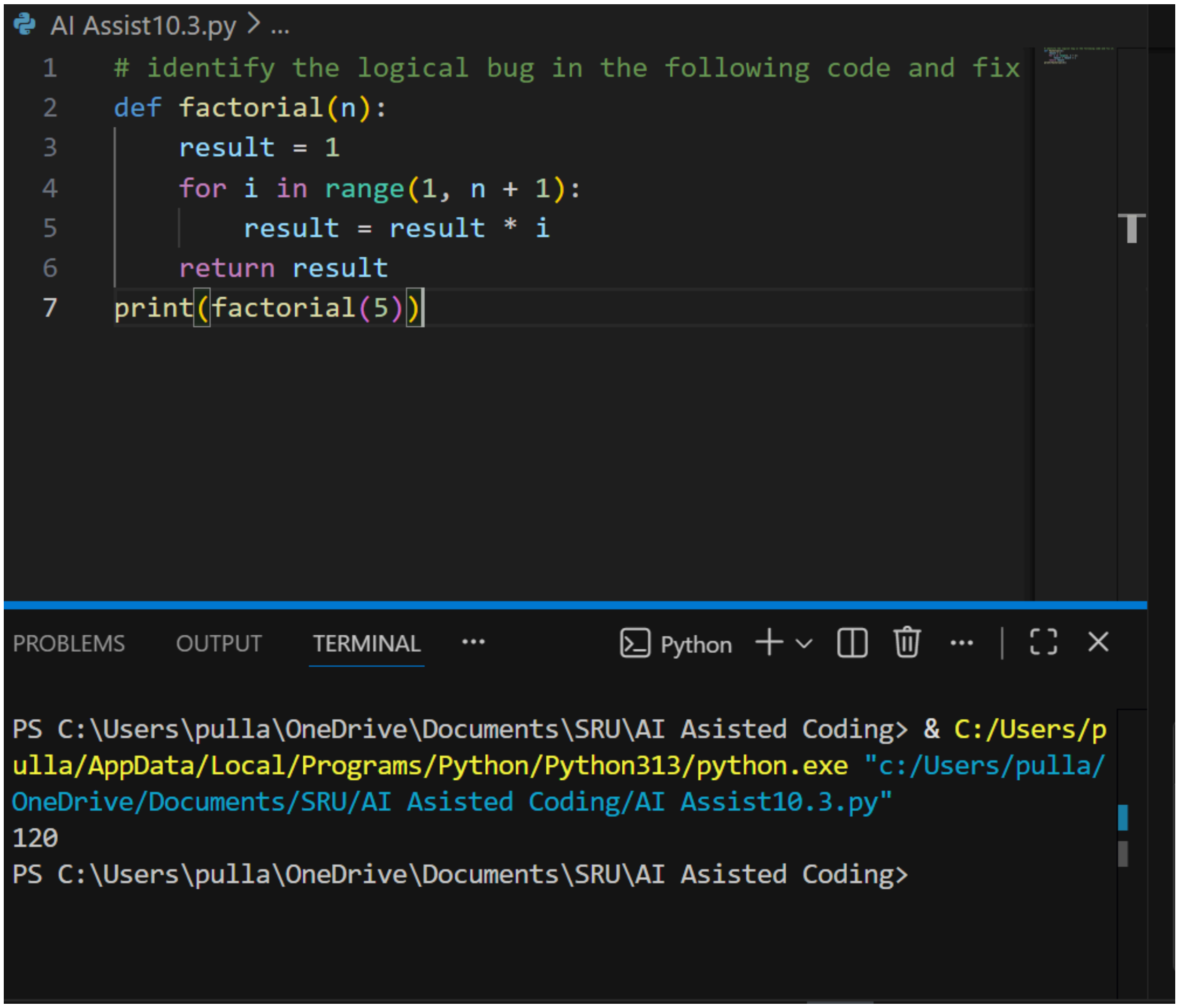
Corrected function (minimal change):

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

Optional improved version:

```
def factorial(n):
result = 1
for i in range(2, n+1):
result *= i
return result
```

Own Manual Fix Code:


AI Assist10.3.py > ...

```python
# identify the logical bug in the following code and fix
def factorial(n):
    result = 1
    for i in range(1, n + 1):
        result = result * i
    return result
print(factorial(5))
```

PROBLEMS   OUTPUT   TERMINAL   ···   Python

```
PS C:\Users\pulla\OneDrive\Documents\SRU\AI Asisted Coding> & C:/Users/p
ulla/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/pulla/
OneDrive/Documents/SRU/AI Asisted Coding/AI Assist10.3.py"
120
PS C:\Users\pulla\OneDrive\Documents\SRU\AI Asisted Coding>
```

Explanation:

AI fixed the logical code bug and gave a corrected version of it. But,still the output is 5 because it is not printing, just returning the output.
So, i have added one line to print the fact(5) and yes it has printed 120

Output:

changed the loop to **for i in range(1, n+1):**
AI output: 5
Manual fix: **<u>120</u>**

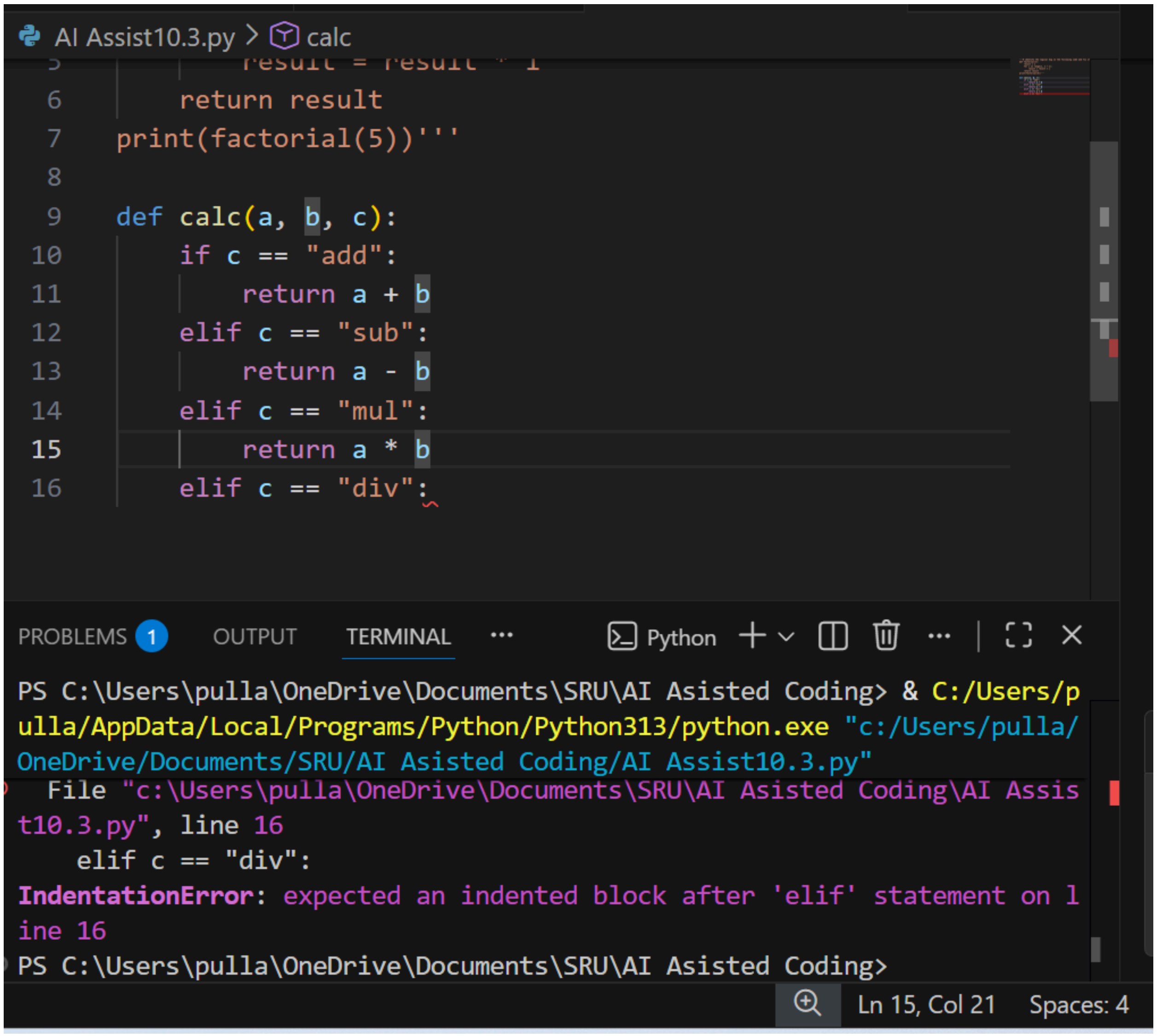
Observation:

AI fixed the bug and also suggested an optimal solution as well. But, didn’t add an extra line to print, this code review fixed the printing issue.

Problem Statement 2: Task 2 — Improving Readability & Documentation

Prompt:

“#identify the logical bug in the following code and fix it”



AI Assist10.3.py > calc

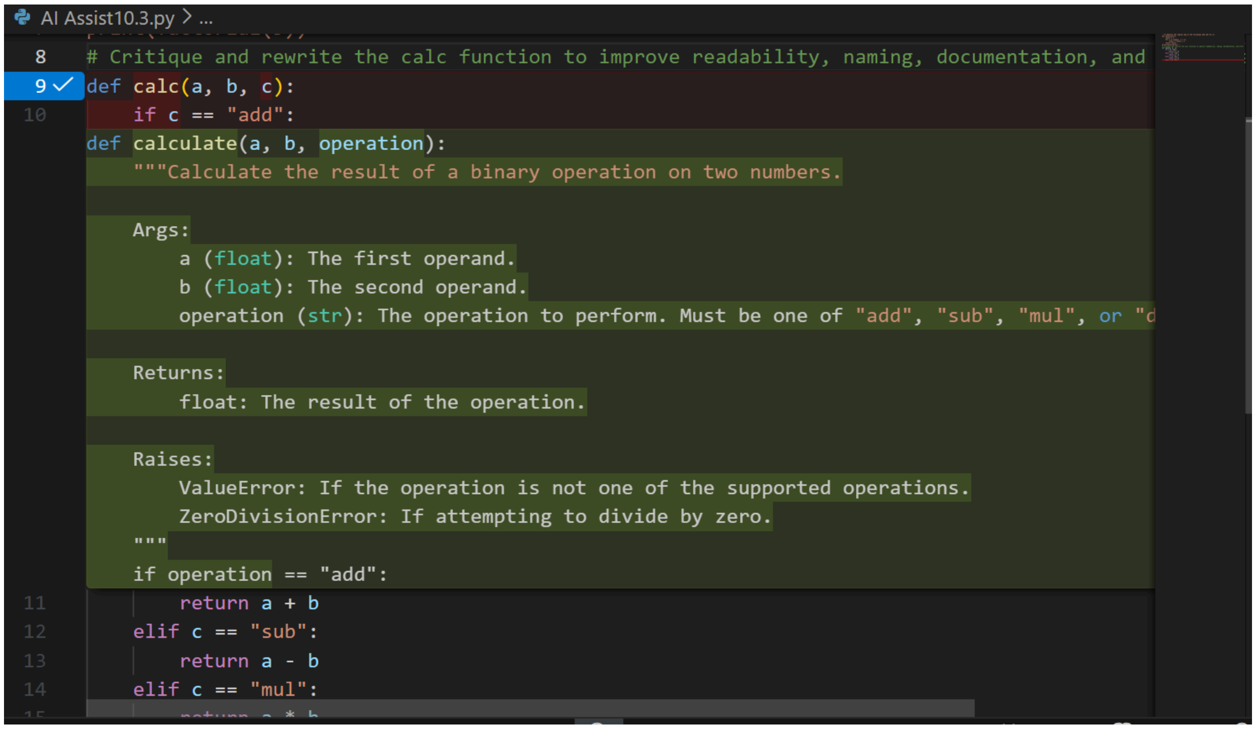
```
        result = result * i
    return result
print(factorial(5))'''

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":
```

PROBLEMS 1 OUTPUT TERMINAL

```
PS C:\Users\pulla\OneDrive\Documents\SRU\AI Asisted Coding> & C:/Users/pulla/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/pulla/OneDrive/Documents/SRU/AI Asisted Coding/AI Assist10.3.py"
  File "c:\Users\pulla\OneDrive\Documents\SRU\AI Asisted Coding\AI Assist10.3.py", line 16
    elif c == "div":
IndentationError: expected an indented block after 'elif' statement on line 16
PS C:\Users\pulla\OneDrive\Documents\SRU\AI Asisted Coding>
```

Ln 15, Col 21 Spaces: 4

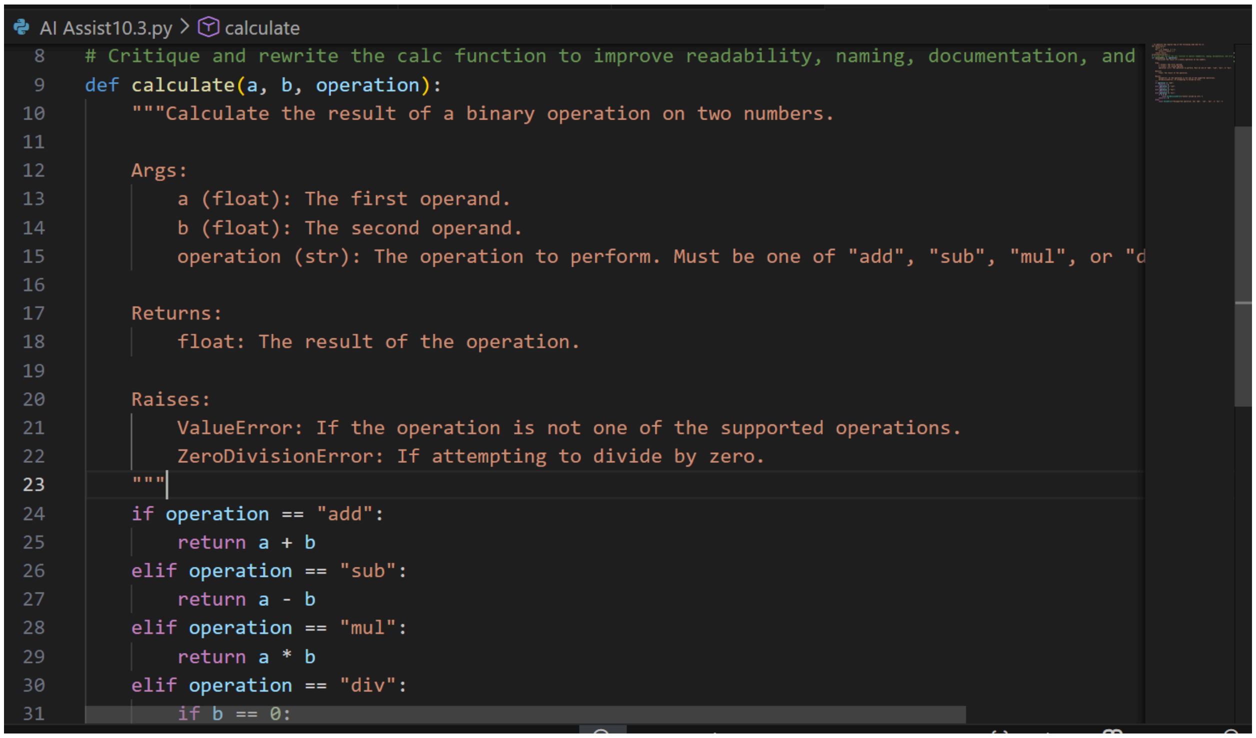


```
AI Assist10.3.py > ...
# Critique and rewrite the calc function to improve readability, naming, documentation, and
def calc(a, b, c):
    if c == "add":
def calculate(a, b, operation):
    """Calculate the result of a binary operation on two numbers.

    Args:
        a (float): The first operand.
        b (float): The second operand.
        operation (str): The operation to perform. Must be one of "add", "sub", "mul", or "d

    Returns:
        float: The result of the operation.

    Raises:
        ValueError: If the operation is not one of the supported operations.
        ZeroDivisionError: If attempting to divide by zero.
    """
    if operation == "add":
        return a + b
    elif c == "sub":
        return a - b
    elif c == "mul":
```



```
AI Assist10.3.py > calculate
# Critique and rewrite the calc function to improve readability, naming, documentation, and
def calculate(a, b, operation):
    """Calculate the result of a binary operation on two numbers.

    Args:
        a (float): The first operand.
        b (float): The second operand.
        operation (str): The operation to perform. Must be one of "add", "sub", "mul", or "d

    Returns:
        float: The result of the operation.

    Raises:
        ValueError: If the operation is not one of the supported operations.
        ZeroDivisionError: If attempting to divide by zero.
    """
    if operation == "add":
        return a + b
    elif operation == "sub":
        return a - b
    elif operation == "mul":
        return a * b
    elif operation == "div":
        if b == 0:
```

AI Review & Improvements:

AI suggested renaming the function and parameters for clarity.
It added a detailed docstring, input validation, and exception handling for division by zero.

Output:

A well-structured function with clear naming and robust error handling.

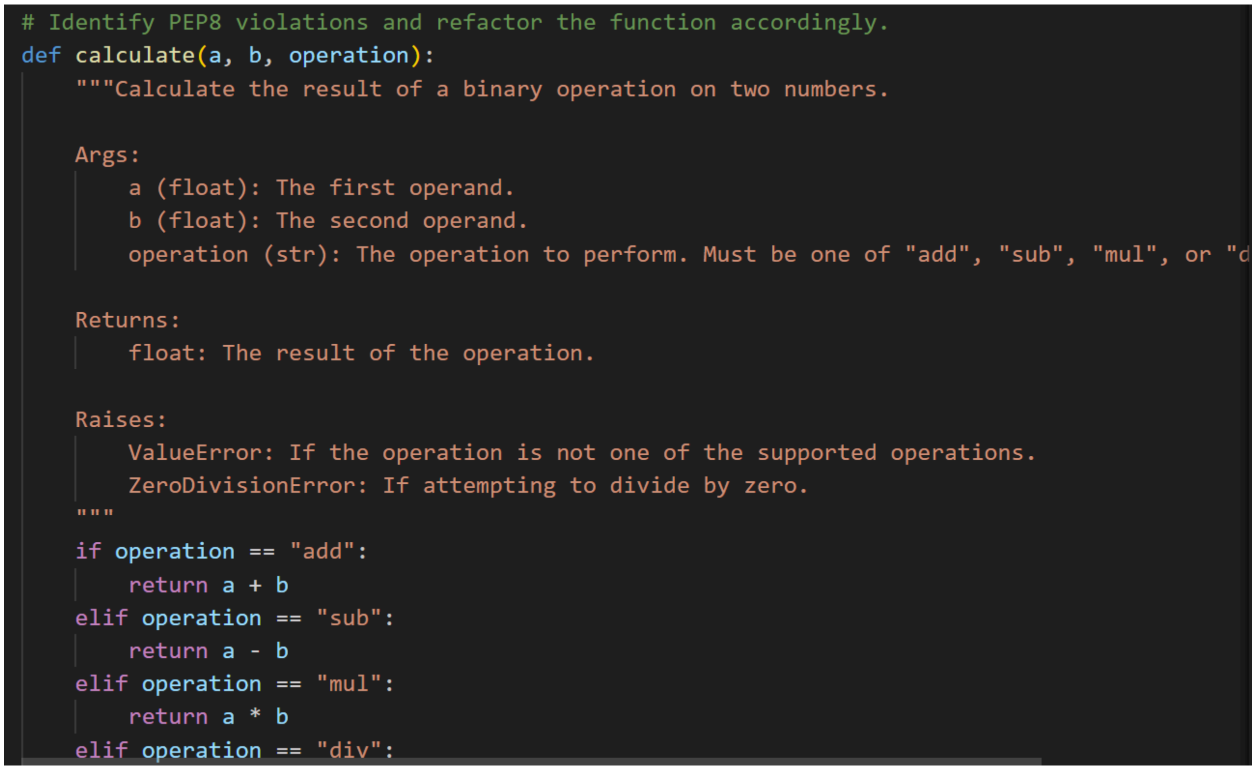
Comparison:

The improved version is more readable, maintainable, and user-friendly compared to the original.

Problem Statement 3: Enforcing Coding Standards

Prompt:

“#Identify PEP8 violations and refactor the function accordingly.”



```python
# Identify PEP8 violations and refactor the function accordingly.
def calculate(a, b, operation):
    """Calculate the result of a binary operation on two numbers.

    Args:
        a (float): The first operand.
        b (float): The second operand.
        operation (str): The operation to perform. Must be one of "add", "sub", "mul", or "d

    Returns:
        float: The result of the operation.

    Raises:
        ValueError: If the operation is not one of the supported operations.
        ZeroDivisionError: If attempting to divide by zero.
    """
    if operation == "add":
        return a + b
    elif operation == "sub":
        return a - b
    elif operation == "mul":
        return a * b
    elif operation == "div":
```

AI Review:

AI detected naming violations (Checkprime → check_prime), improper indentation, and spacing issues.

Refactored Version:

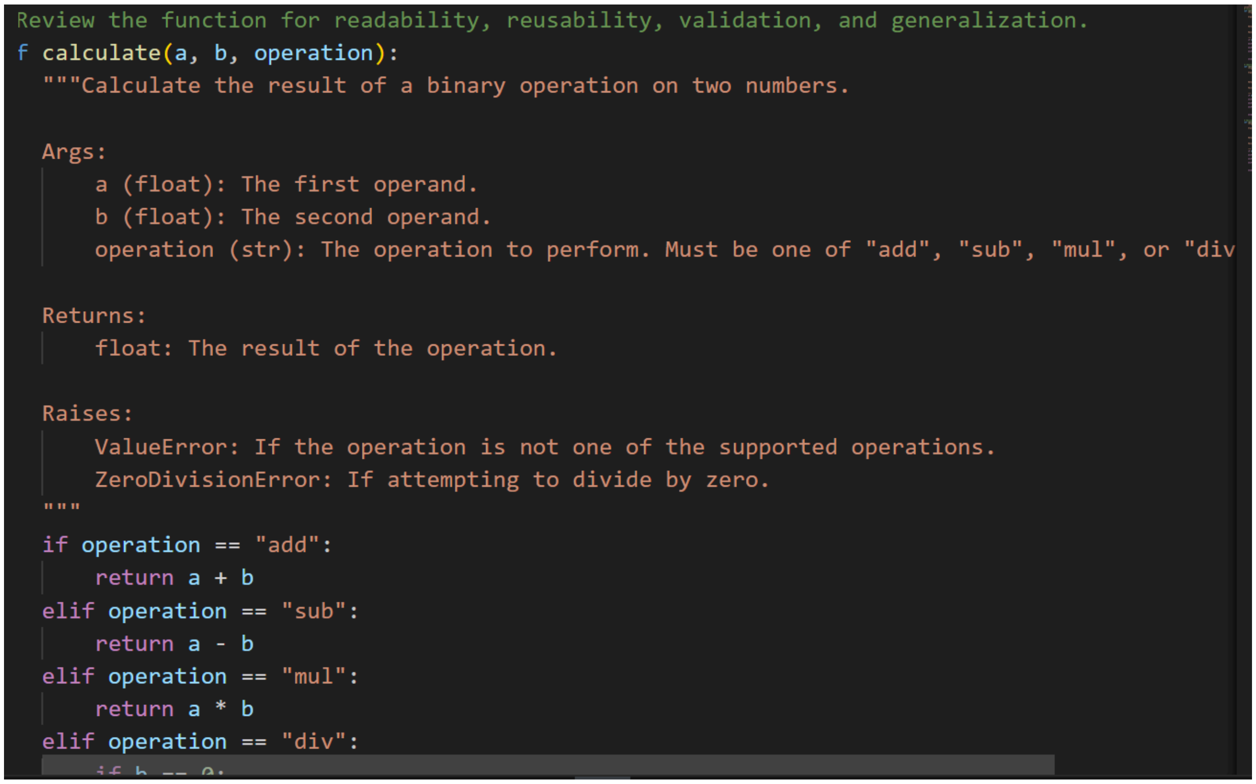
The function was renamed and formatted to comply with PEP8 standards.

Output:
The function works correctly and follows standard naming conventions.

Observation:
AI-based code review helps maintain consistency and improves collaboration in large teams.

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

Prompt:
"#Review the function for readability, reusability, validation, and generalization."



```
Review the function for readability, reusability, validation, and generalization.
f calculate(a, b, operation):
  """Calculate the result of a binary operation on two numbers.

  Args:
      a (float): The first operand.
      b (float): The second operand.
      operation (str): The operation to perform. Must be one of "add", "sub", "mul", or "div

  Returns:
      float: The result of the operation.

  Raises:
      ValueError: If the operation is not one of the supported operations.
      ZeroDivisionError: If attempting to divide by zero.
  """
  if operation == "add":
      return a + b
  elif operation == "sub":
      return a - b
  elif operation == "mul":
      return a * b
  elif operation == "div":
      if b == 0:
```

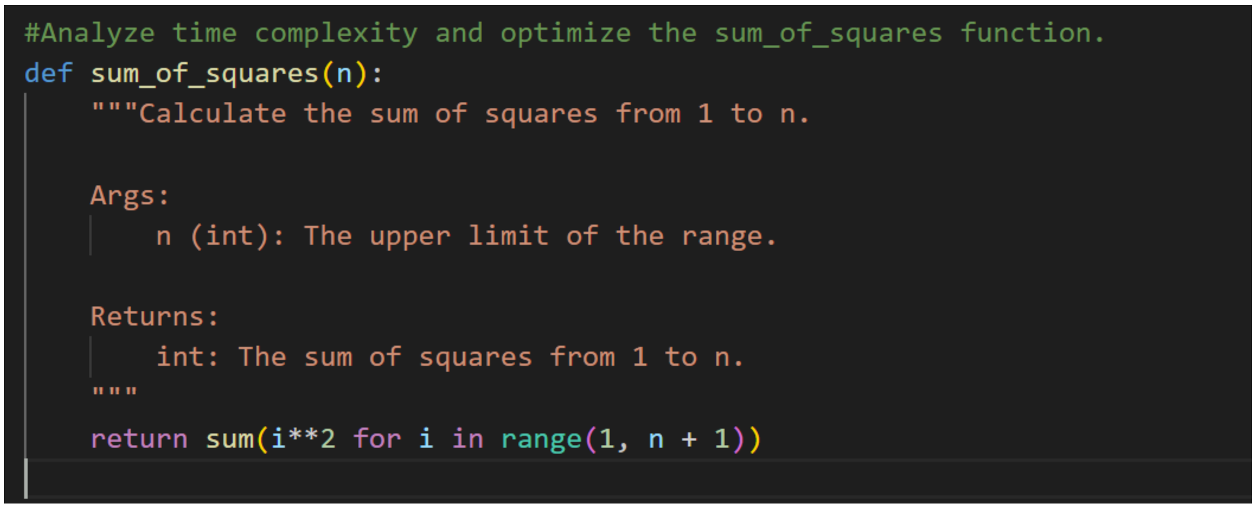
AI Review & Refactoring:
AI suggested better function naming, added type hints, input validation, and improved modularity.
It also recommended generalizing the multiplier logic.

Output:
An improved function with validation and clearer intent.

Observation:
AI works best as an assistant reviewer rather than a standalone reviewer.
Human oversight remains essential for contextual decision-making.

Problem Statement 5: — AI-Assisted Performance Optimization

Prompt Used:
“#Analyze time complexity and optimize the sum_of_squares function.”



```python
#Analyze time complexity and optimize the sum_of_squares function.
def sum_of_squares(n):
    """Calculate the sum of squares from 1 to n.

    Args:
        n (int): The upper limit of the range.

    Returns:
        int: The sum of squares from 1 to n.
    """
    return sum(i**2 for i in range(1, n + 1))
```

AI Analysis:
The original function has O(n) time complexity.
AI suggested using a generator expression inside `sum()` for cleaner and slightly optimized code.

Optimized Version:

```
return sum(x * x for x in numbers)
```

Output:
Optimized function performs efficiently on large datasets.

Comparison & Observation:

The optimized version improves readability and may offer slight performance benefits.
Trade-off analysis shows that readability and maintainability are equally important as performance.

Conclusion:

This lab demonstrates how AI-assisted code reviews improve correctness, readability, maintainability, and performance while reinforcing the importance of human oversight.