SCHOOL OF COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE	**DEPARTMENT OF COMPUTER SCIENCE ENGINEERING**

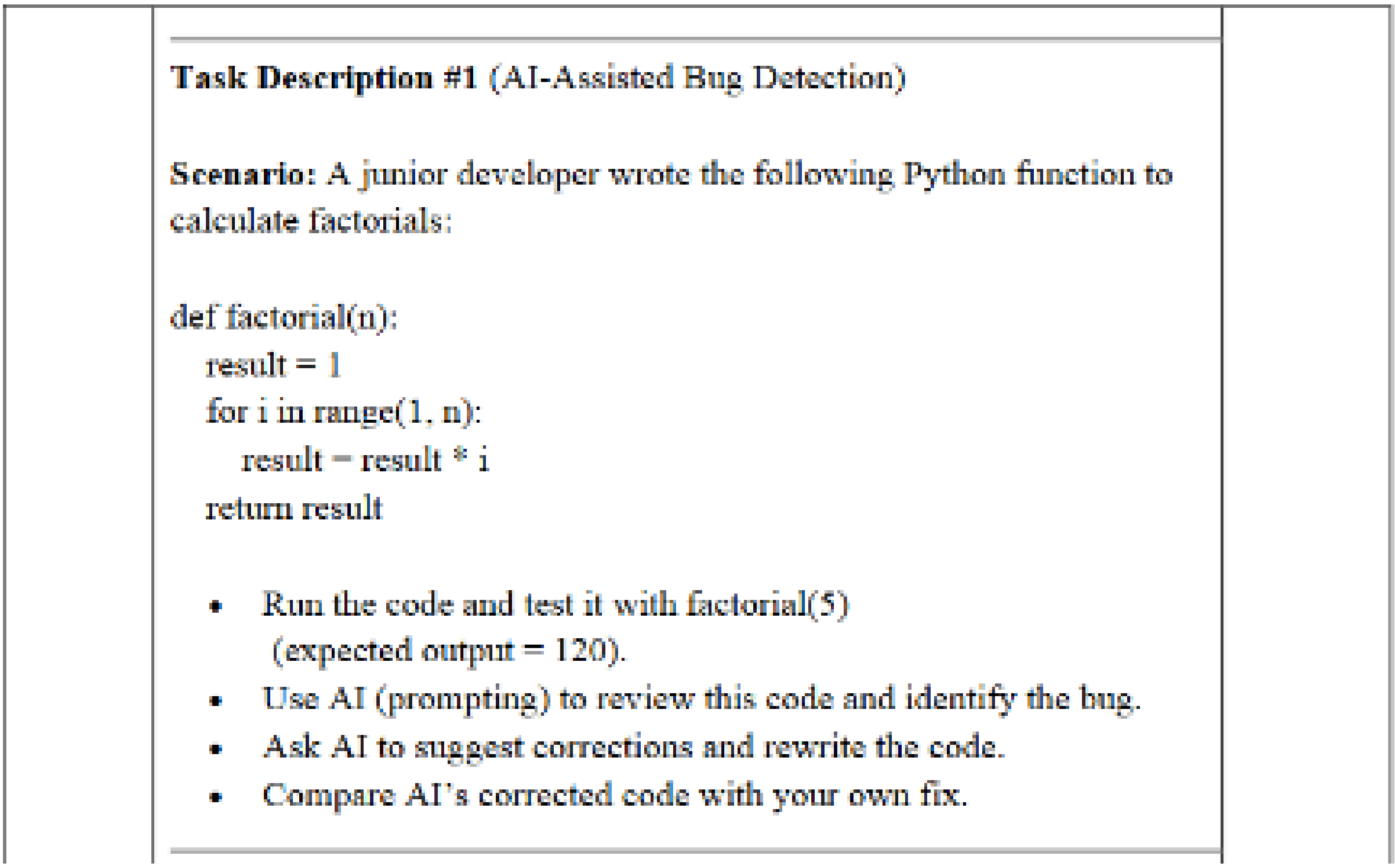
Program Name: B. Tech	**Assignment Type: Lab**	**Academic Year:**2025-2026

Course Coordinator Name	Venkataramana Veeramsetty
Instructor(s) Name	Dr. V. Venkataramana (Co-ordinator) Dr. T. Sampath Kumar Dr. Pramoda Patro Dr. Brij Kishor Tiwari Dr.J.Ravichander Dr. Mohammand Ali Shaik Dr. Anirodh Kumar Mr. S.Naresh Kumar Dr. RAJESH VELPULA Mr. Kundhan Kumar Ms. Ch.Rajitha Mr. M Prakash Mr. B.Raju Intern 1 (Dharma teja) Intern 2 (Sai Prasad) Intern 3 (Sowmya) NS_2 (Mounika)

Course Code	24CS002PC215	**Course Title**	AI Assisted Coding
Year/Sem	II/I	**Regulation**	R24
Date and Day of Assignment	Week5 - Monday	**Time(s)**	
Duration	2 Hours	**Applicable to Batches**	

AssignmentNumber: 9.1(Present assignment number)/**24**(Total number of assignments)

Q.No.	**Question**	***Expected Time to complete***
1	**Lab 9 – Code Review and Quality: Using AI to improve code quality and readability** **Lab Objectives** • Inline comments • Docstrings • Auto-documentation tools • AI-assisted summarization	Week5 - Monday



Task Description #1 (AI-Assisted Bug Detection)

Scenario: A junior developer wrote the following Python function to calculate factorials:

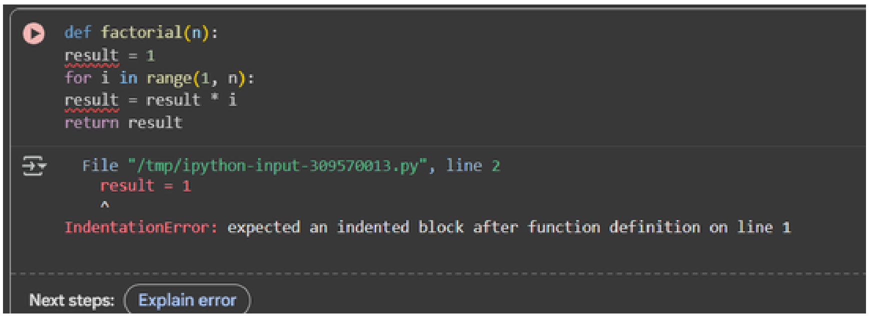
```
def factorial(n):
    result = 1
    for i in range(1, n):
        result = result * i
    return result
```

- Run the code and test it with factorial(5) (expected output = 120).
- Use AI (prompting) to review this code and identify the bug.
- Ask AI to suggest corrections and rewrite the code.
- Compare AI's corrected code with your own fix.

prompts :

Review the following factorial code, identify the bug, suggest corrections, and rewrite the fixed version.

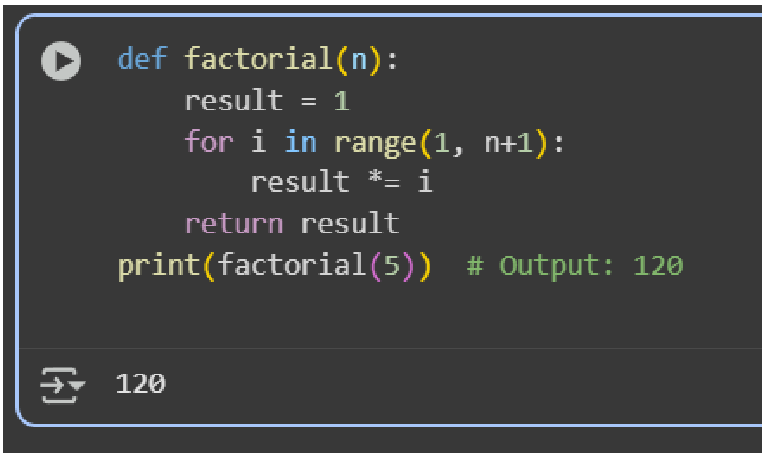
code::



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

```
  File "/tmp/ipython-input-309570013.py", line 2
    result = 1
    ^
IndentationError: expected an indented block after function definition on line 1
```

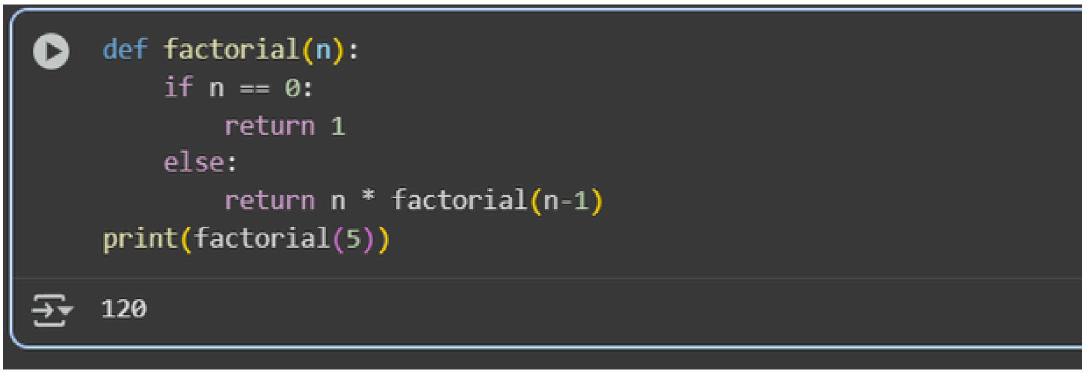
manual corrected code :



```
def factorial(n):
    result = 1
    for i in range(1, n+1):
        result *= i
    return result
print(factorial(5))  # Output: 120
```

```
120
```

AI corrected code :

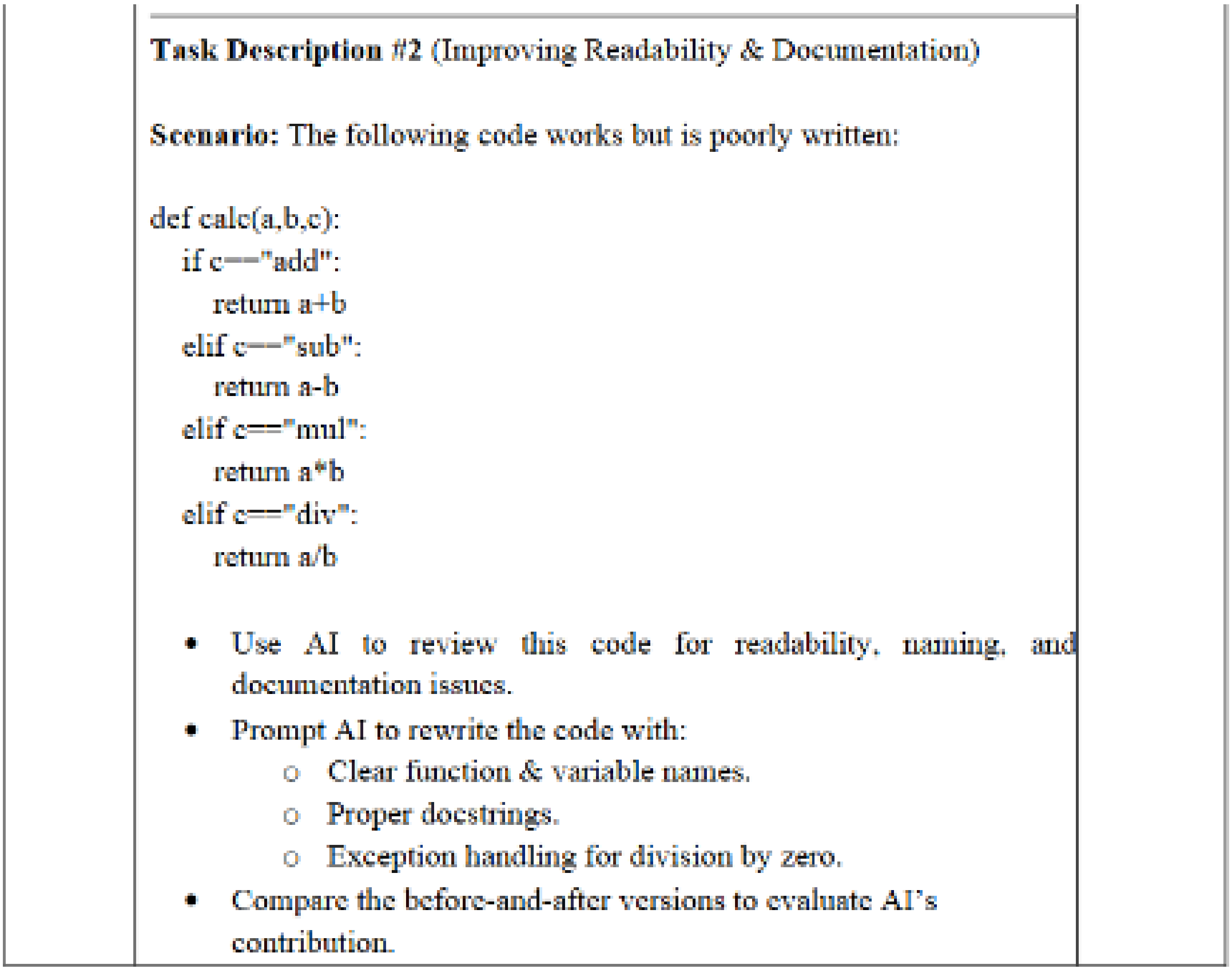


```
def factorial(n):
    if n == 0:
        return 1
    else:
        return n * factorial(n-1)
print(factorial(5))
```

```
120
```

observations and code expalnation

- AI’s corrected version keeps recursion but fixes the base case (return 1 instead of return 0).
- My fix uses an iterative loop, which avoids recursion depth issues.
- Both corrected codes return the correct output: factorial(5) = 120.
- Recursive code is simple and closer to the mathematical definition.
- Iterative code is more efficient and reliable for larger input values.



Task Description #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":
    return a/b
```

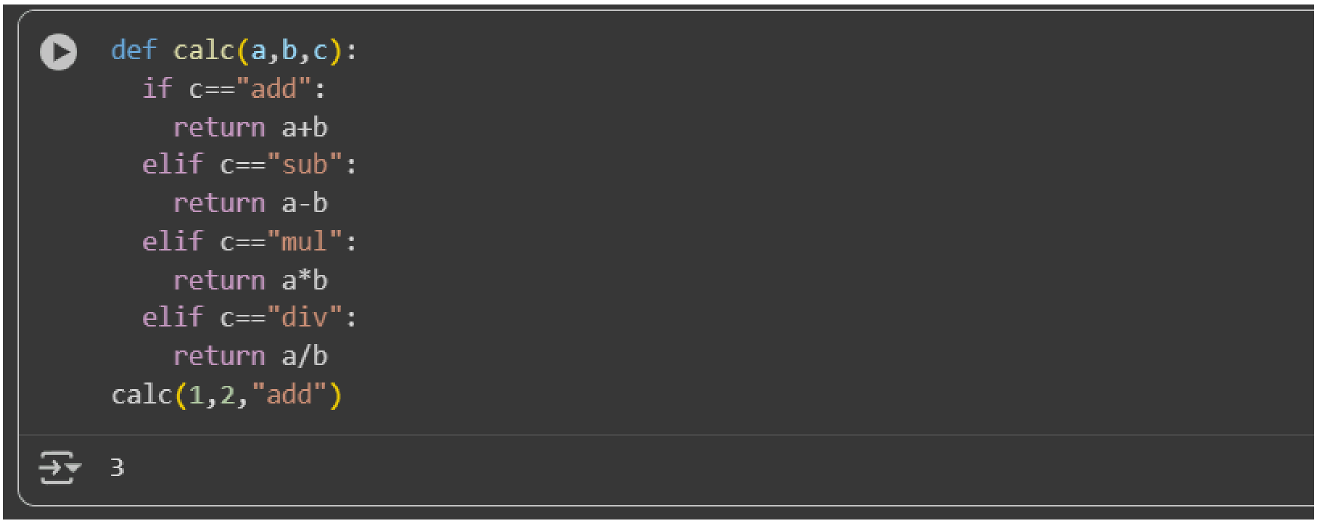
- Use AI to review this code for readability, naming, and documentation issues.
- Prompt AI to rewrite the code with:
  - Clear function & variable names.
  - Proper docstrings.
  - Exception handling for division by zero.
- Compare the before-and-after versions to evaluate AI’s contribution.

prompts :

Review the following calculator function for readability, naming, and documentation issues. Rewrite it with clearer function and variable names, proper docstrings, and exception handling for division by zero.

code:

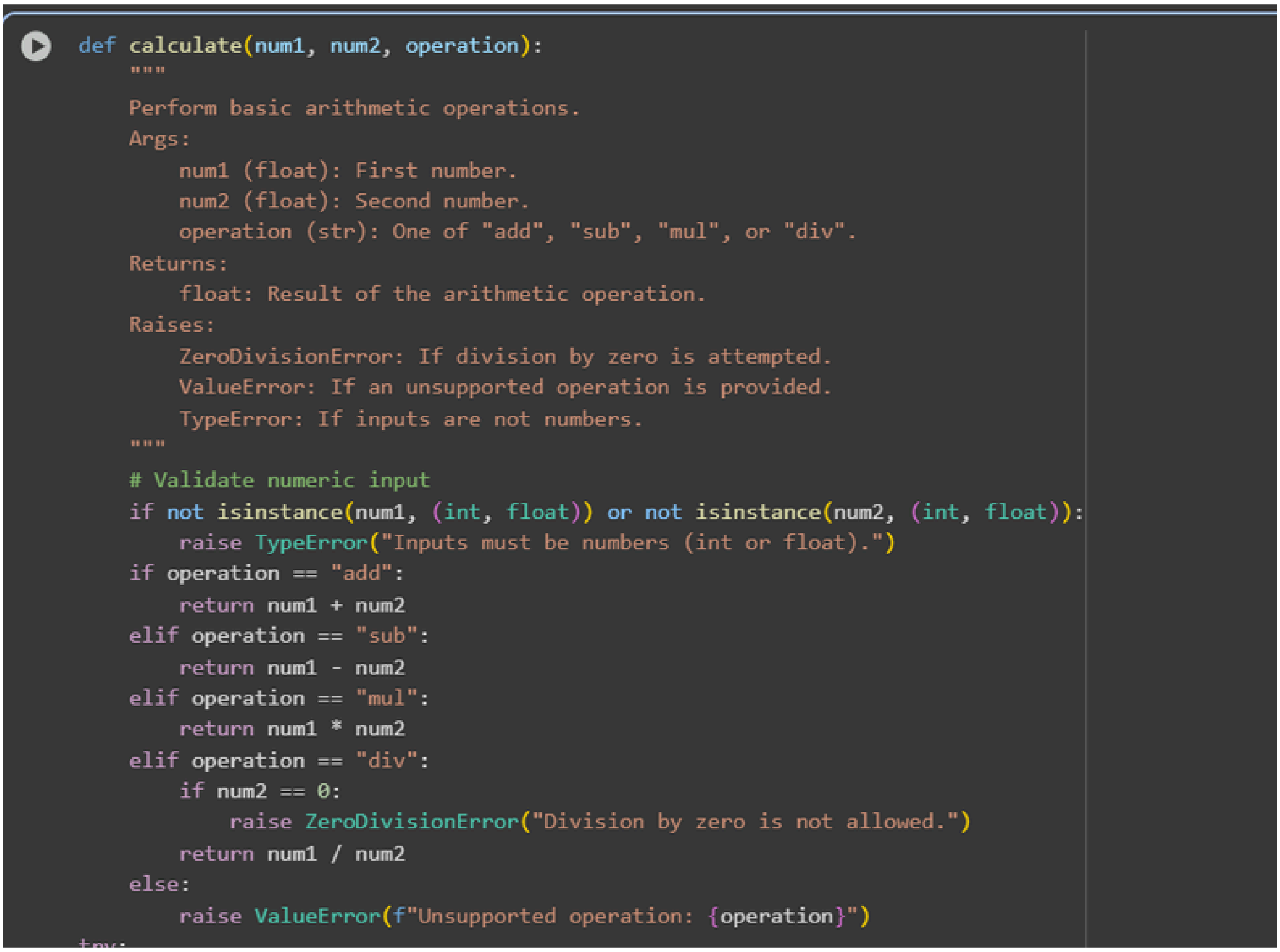
non edited code with no ai corrections:



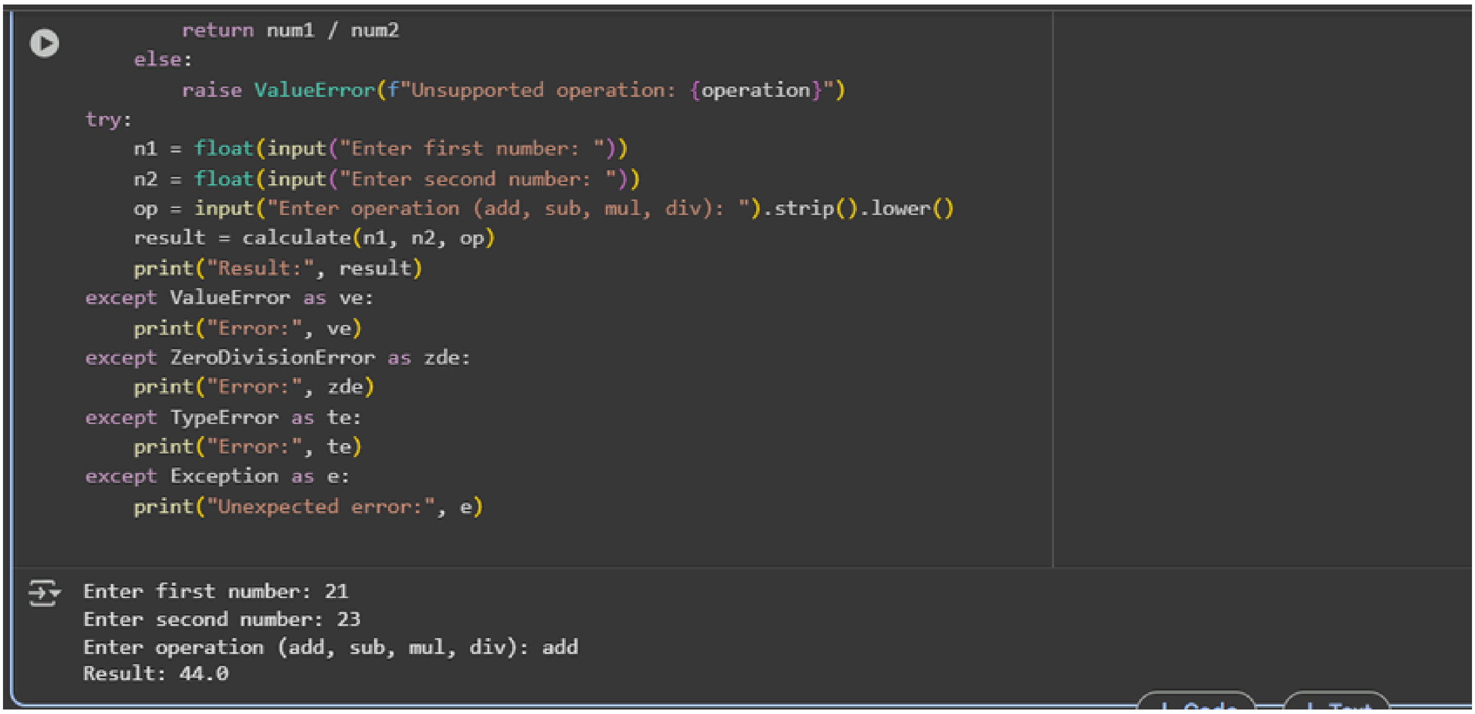
```python
def calc(a,b,c):
  if c=="add":
    return a+b
  elif c=="sub":
    return a-b
  elif c=="mul":
    return a*b
  elif c=="div":
    return a/b
calc(1,2,"add")
```

```
3
```

edited code with ai corrections:



```python
def calculate(num1, num2, operation):
    """
    Perform basic arithmetic operations.
    Args:
        num1 (float): First number.
        num2 (float): Second number.
        operation (str): One of "add", "sub", "mul", or "div".
    Returns:
        float: Result of the arithmetic operation.
    Raises:
        ZeroDivisionError: If division by zero is attempted.
        ValueError: If an unsupported operation is provided.
        TypeError: If inputs are not numbers.
    """
    # Validate numeric input
    if not isinstance(num1, (int, float)) or not isinstance(num2, (int, float)):
        raise TypeError("Inputs must be numbers (int or float).")
    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("Division by zero is not allowed.")
        return num1 / num2
    else:
        raise ValueError(f"Unsupported operation: {operation}")
try:
```



```
        return num1 / num2
    else:
        raise ValueError(f"Unsupported operation: {operation}")
try:
    n1 = float(input("Enter first number: "))
    n2 = float(input("Enter second number: "))
    op = input("Enter operation (add, sub, mul, div): ").strip().lower()
    result = calculate(n1, n2, op)
    print("Result:", result)
except ValueError as ve:
    print("Error:", ve)
except ZeroDivisionError as zde:
    print("Error:", zde)
except TypeError as te:
    print("Error:", te)
except Exception as e:
    print("Unexpected error:", e)
```

```
Enter first number: 21
Enter second number: 23
Enter operation (add, sub, mul, div): add
Result: 44.0
```

observations and code expalnation

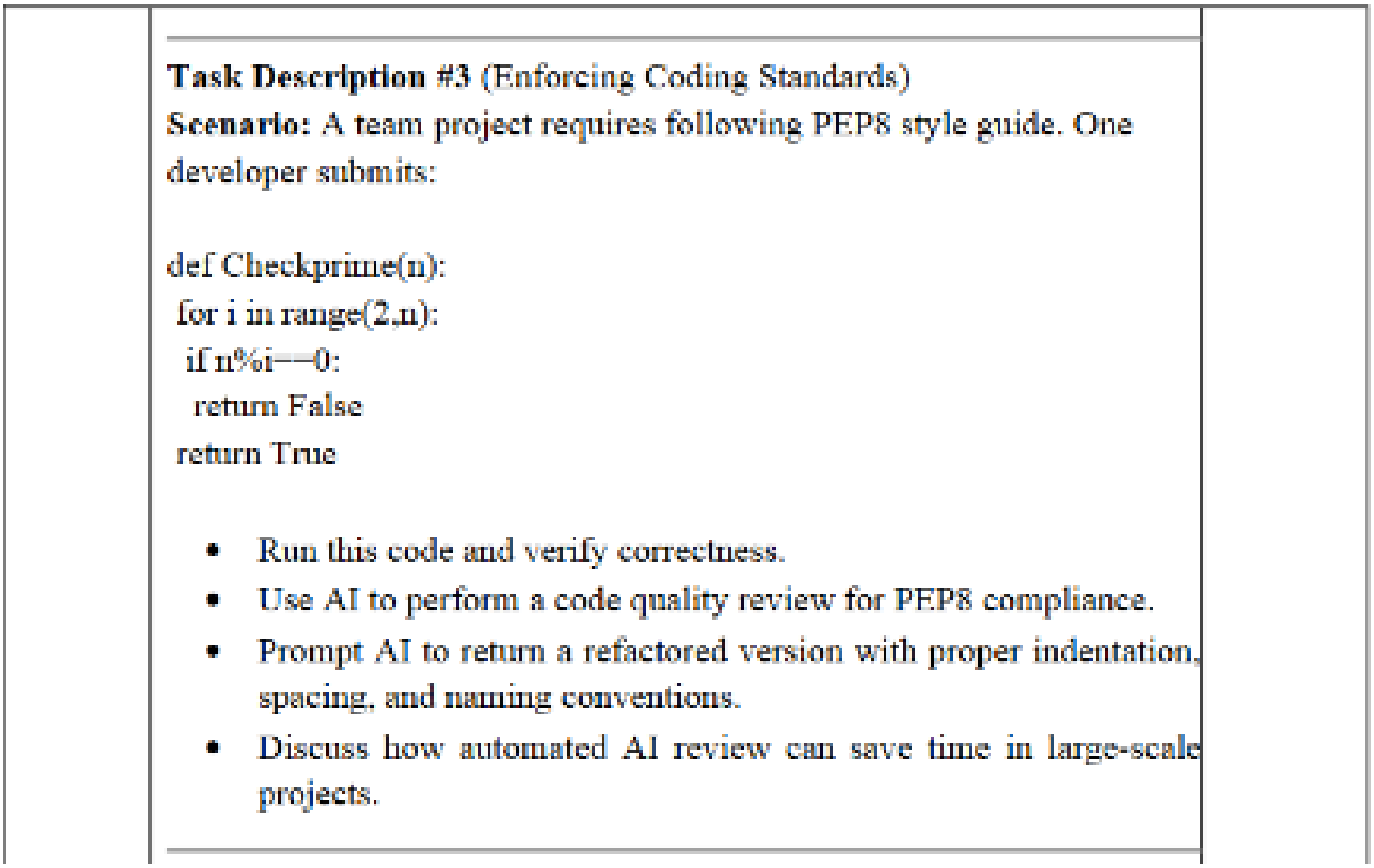
- Data Analysis Key Findings

- The original calc function used single-letter variable names (a, b, c), lacked documentation, and did not handle division by zero.
- The rewritten function, perform_operation, uses descriptive variable names (num1, num2, operation) and includes a comprehensive docstring explaining its purpose, arguments, and return value.
- Error handling for division by zero was successfully implemented, raising a ZeroDivisionError with a descriptive message when the divisor is 0.
- Tests confirmed that perform_operation correctly performs addition, subtraction, multiplication, and division for valid inputs and properly raises a ZeroDivisionError when attempting to divide by zero.

- Insights or Next Steps

- The improved function is significantly more readable, maintainable, and robust due to descriptive naming, documentation, and error handling.
- Future enhancements could include handling other potential errors, such as invalid operation names or non-numeric inputs.

.



Task Description #3 (Enforcing Coding Standards)

Scenario: A team project requires following PEP8 style guide. One developer submits:

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

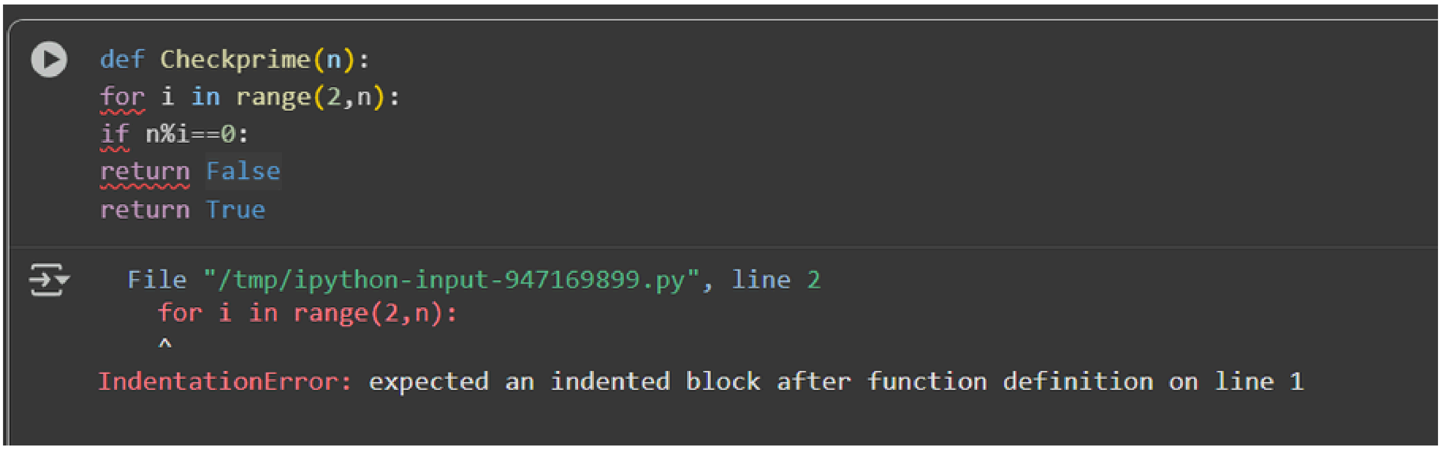
- Run this code and verify correctness.
- Use AI to perform a code quality review for PEP8 compliance.
- Prompt AI to return a refactored version with proper indentation, spacing, and naming conventions.
- Discuss how automated AI review can save time in large-scale projects.

prompts :

Review the following prime-checking function for PEP8 compliance. Refactor it with proper indentation, spacing, naming conventions, and add a short docstring

code :

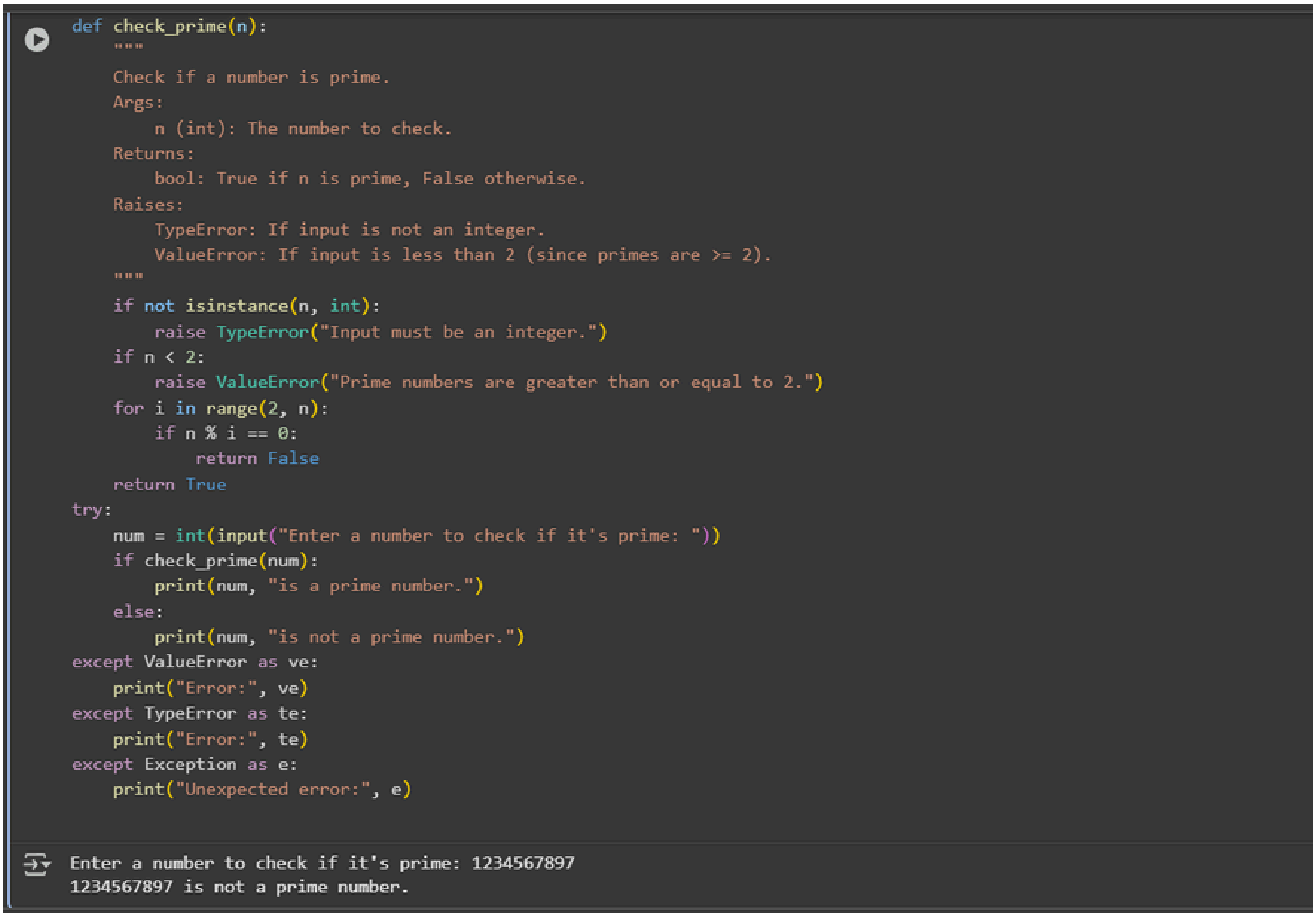
nomal code with no AI corrrections :



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

```
  File "/tmp/ipython-input-947169899.py", line 2
    for i in range(2,n):
    ^
IndentationError: expected an indented block after function definition on line 1
```

AI generated code with AI corrrections :



```python
def check_prime(n):
    """
    Check if a number is prime.
    Args:
        n (int): The number to check.
    Returns:
        bool: True if n is prime, False otherwise.
    Raises:
        TypeError: If input is not an integer.
        ValueError: If input is less than 2 (since primes are >= 2).
    """
    if not isinstance(n, int):
        raise TypeError("Input must be an integer.")
    if n < 2:
        raise ValueError("Prime numbers are greater than or equal to 2.")
    for i in range(2, n):
        if n % i == 0:
            return False
    return True
try:
    num = int(input("Enter a number to check if it's prime: "))
    if check_prime(num):
        print(num, "is a prime number.")
    else:
        print(num, "is not a prime number.")
except ValueError as ve:
    print("Error:", ve)
except TypeError as te:
    print("Error:", te)
except Exception as e:
    print("Unexpected error:", e)
```

```
Enter a number to check if it's prime: 1234567897
1234567897 is not a prime number.
```

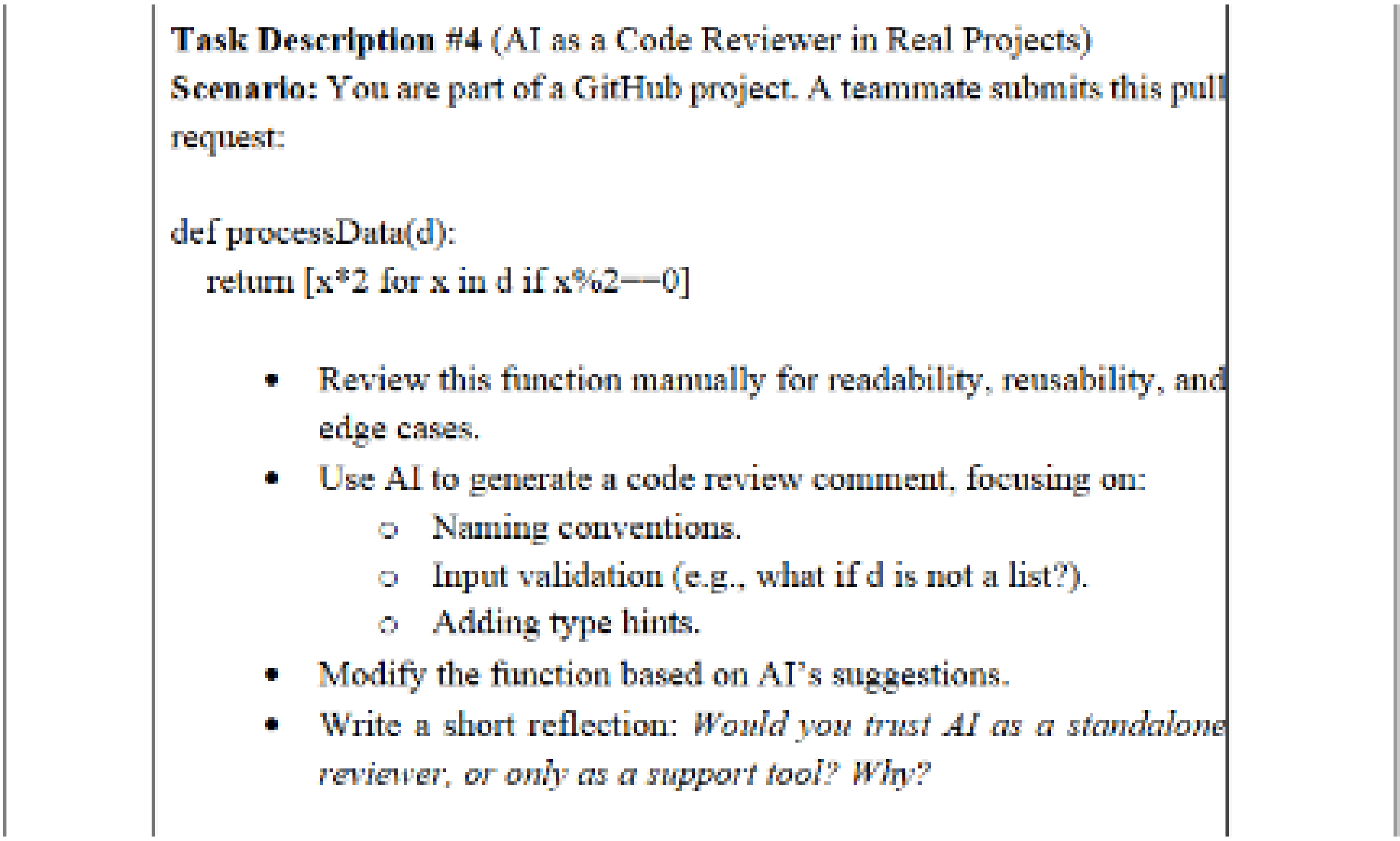
observations and code expalnation :

- Data Analysis Key Findings

- The original Checkprime function had several PEP 8 compliance issues, including inconsistent indentation, lack of consistent spacing, CamelCase naming (Checkprime), and no docstring.
- The original function also contained a logical error where it would incorrectly return True after checking only the first possible divisor.
- The refactored function, renamed is_prime, now adheres to PEP 8 standards with proper indentation (4 spaces), consistent spacing, and snake_case naming.
- A docstring was successfully added to the is_prime function, explaining its purpose, arguments, and return value.
- Testing confirmed that the refactored is_prime function correctly identifies prime numbers, non-prime numbers, and handles edge cases like 0, 1, and negative numbers.

- Insights or Next Steps

- The refactored is_prime function is now more readable, maintainable, and functionally correct due to adherence to PEP 8 and correction of the logical error.
- For larger numbers, optimizing the prime checking logic (e.g., checking divisibility only up to the square root of the number) could improve performance.



Task Description #4 (AI as a Code Reviewer in Real Projects)

Scenario: You are part of a GitHub project. A teammate submits this pull request:

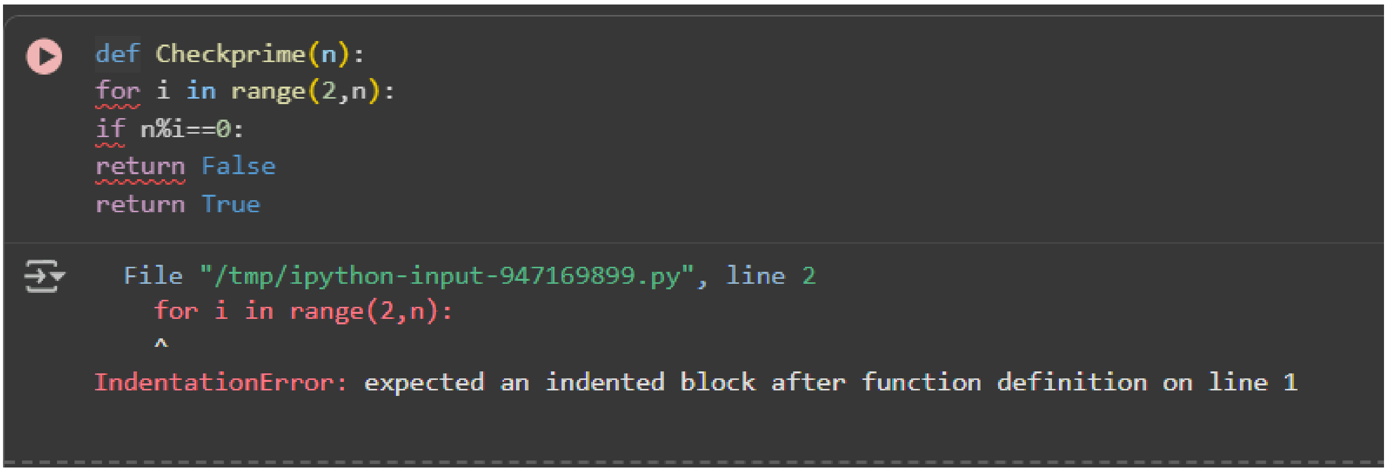
```
def processData(d):
    return [x*2 for x in d if x%2==0]
```

- Review this function manually for readability, reusability, and edge cases.
- Use AI to generate a code review comment, focusing on:
    - Naming conventions.
    - Input validation (e.g., what if d is not a list?).
    - Adding type hints.
- Modify the function based on AI's suggestions.
- Write a short reflection: *Would you trust AI as a standalone reviewer, or only as a support tool? Why?*

Prompt :
Review this function for readability and edge cases, then rewrite it with better naming, input validation, and type hints

Code:

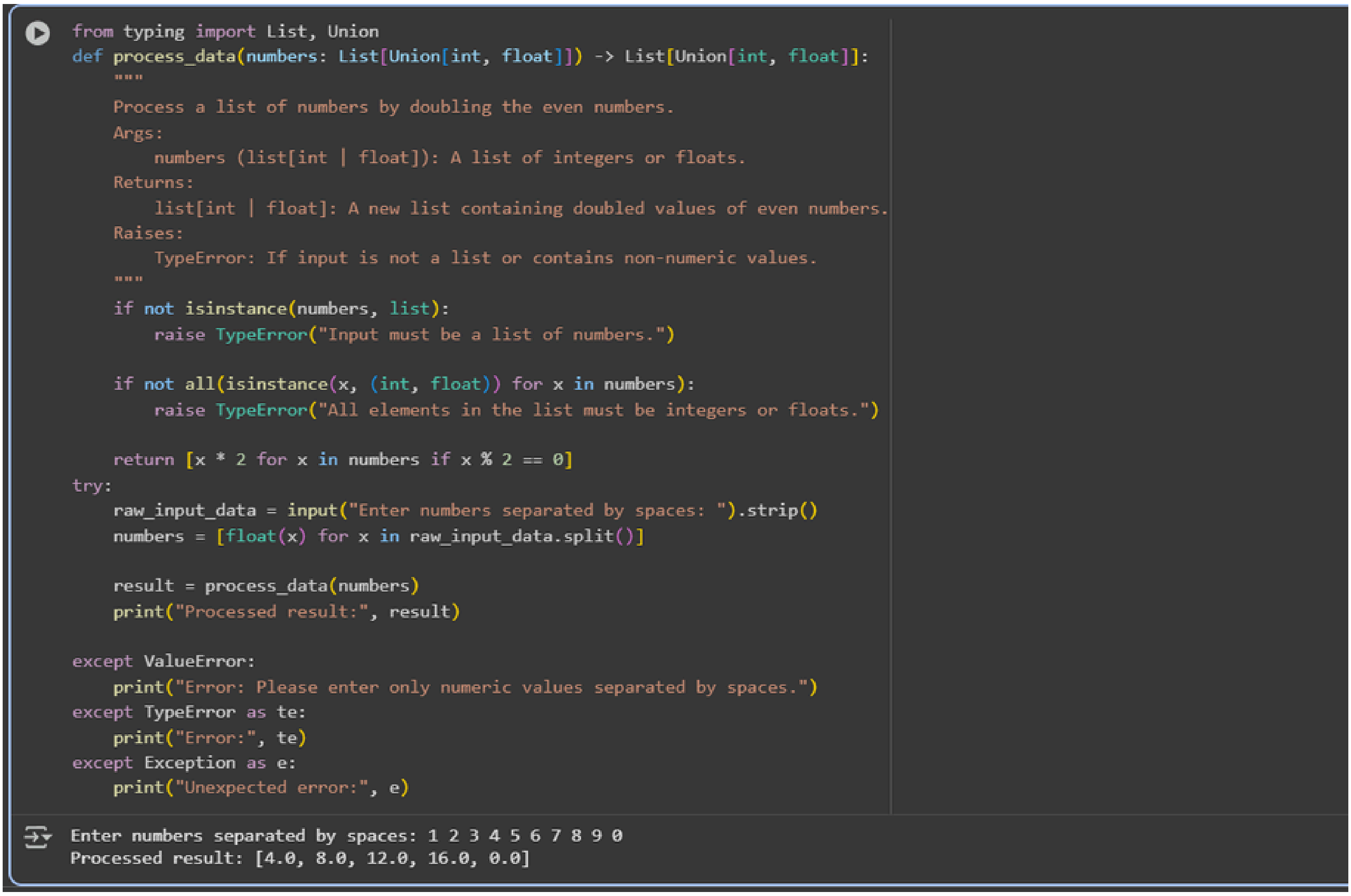
non edited code with no ai corrections and improvements



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

```
  File "/tmp/ipython-input-947169899.py", line 2
    for i in range(2,n):
    ^
IndentationError: expected an indented block after function definition on line 1
```

AI edited code with ai corrections and improvements



```
from typing import List, Union
def process_data(numbers: List[Union[int, float]]) -> List[Union[int, float]]:
    """
    Process a list of numbers by doubling the even numbers.
    Args:
        numbers (list[int | float]): A list of integers or floats.
    Returns:
        list[int | float]: A new list containing doubled values of even numbers.
    Raises:
        TypeError: If input is not a list or contains non-numeric values.
    """
    if not isinstance(numbers, list):
        raise TypeError("Input must be a list of numbers.")

    if not all(isinstance(x, (int, float)) for x in numbers):
        raise TypeError("All elements in the list must be integers or floats.")

    return [x * 2 for x in numbers if x % 2 == 0]
try:
    raw_input_data = input("Enter numbers separated by spaces: ").strip()
    numbers = [float(x) for x in raw_input_data.split()]

    result = process_data(numbers)
    print("Processed result:", result)

except ValueError:
    print("Error: Please enter only numeric values separated by spaces.")
except TypeError as te:
    print("Error:", te)
except Exception as e:
    print("Unexpected error:", e)
```

```
Enter numbers separated by spaces: 1 2 3 4 5 6 7 8 9 0
Processed result: [4.0, 8.0, 12.0, 16.0, 0.0]
```

Camparision and code explanation

- AI may miss subtle logical errors or performance bottlenecks.
- AI follows patterns but lacks full project context (business logic, scalability).
- AI is best used as a support tool to catch style, naming, and common mistakes quickly, saving reviewers’ time.
- Final approval should always involve a human reviewer for deeper insight.