AI Assisted Coding

Lab Assignment 10.3

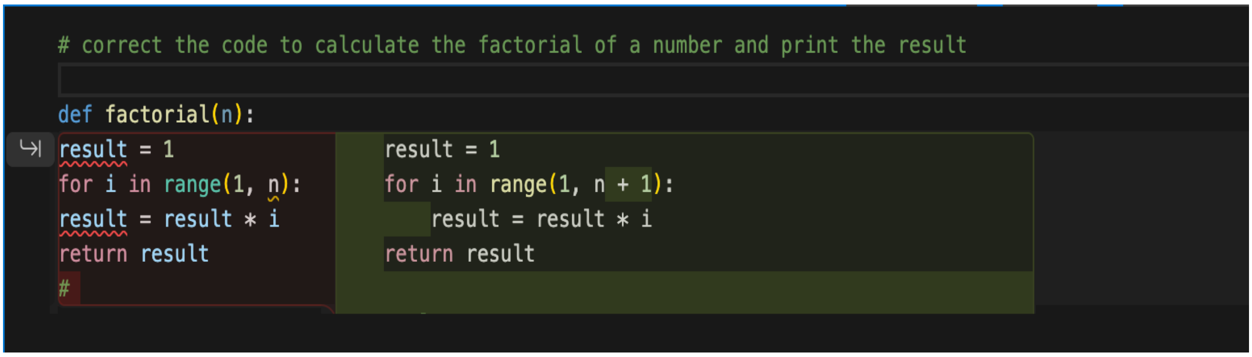
Name : V Sai Krishnachaithanya

Hall Ticket no : 2303A52154

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:
3. Identify the logical bug in the code.
4. Explain why the bug occurs (e.g., off-by-one error).
5 Provide a corrected version.
6. Compare the AI's corrected code with your own manual fix.
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).

Prompt:

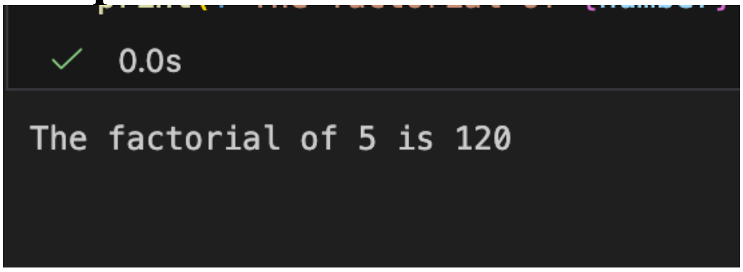
Correct the code to calculate the factorial of a number and print the result



```
# correct the code to calculate the factorial of a number and print the result

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

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

Output:



```
✓ 0.0s

The factorial of 5 is 120
```

Problem Statement 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:

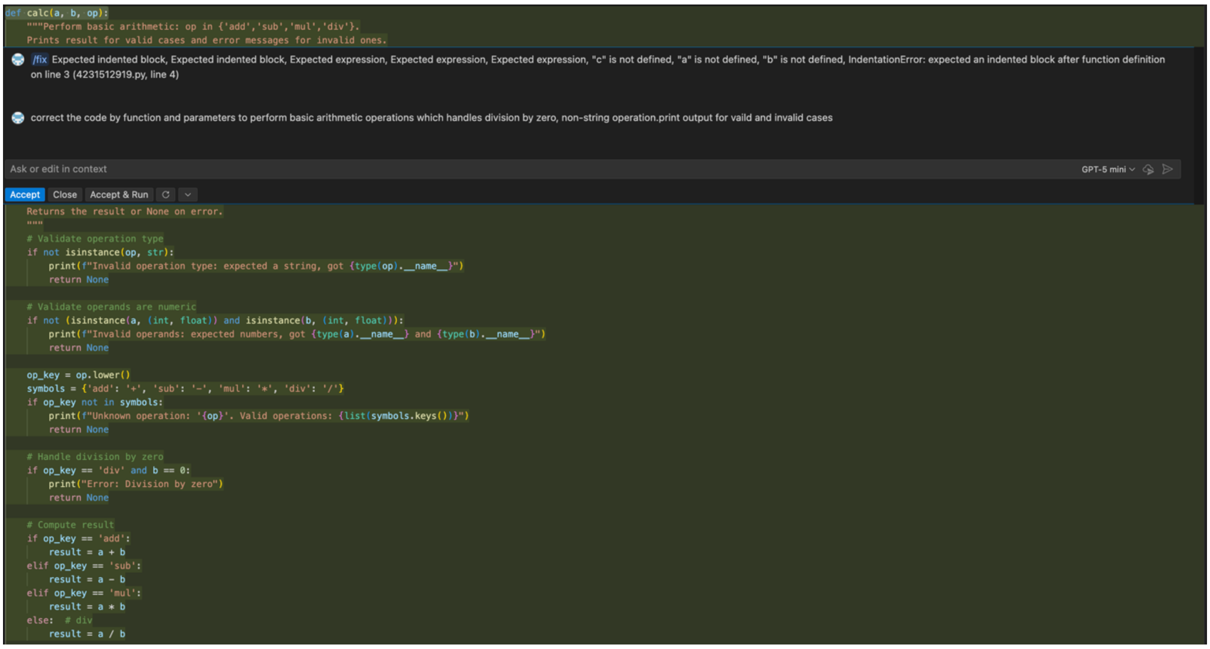
1. Descriptive function and parameter names.
2. A complete docstring (description, parameters,return value, examples).
3. Exception handling for division by zero.
4. Consideration of input validation.
5. Compare the original and AI-improved versions.
6. Test both with valid and invalid inputs (e.g., division by zero, non-string operation).

Expected Output:

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

Prompt :

correct the code by function and parameters to perform basic arithmetic operations which handles division by zero, non-string operation.print output for vaild and invalid cases



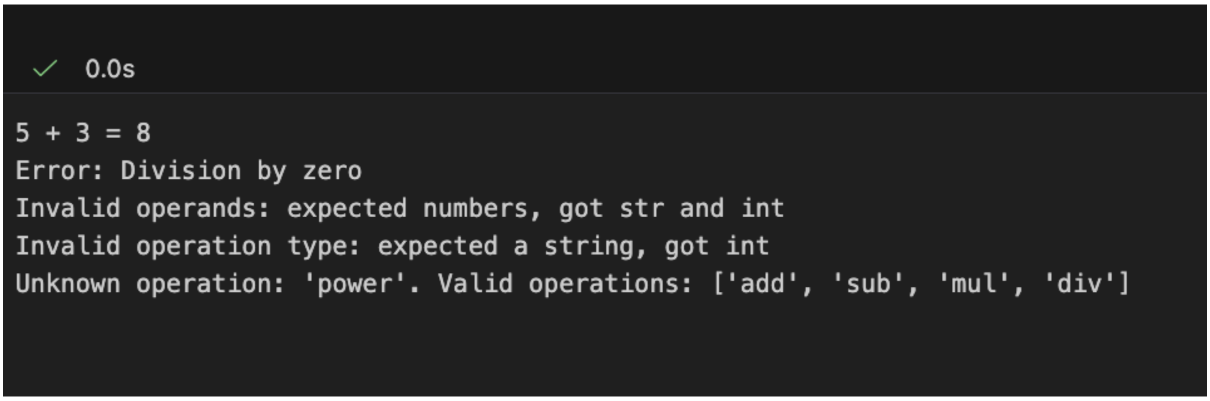
```
def calc(a, b, op):
    """Perform basic arithmetic: op in {'add','sub','mul','div'}.
    Prints result for valid cases and error messages for invalid ones.
```

/fix Expected indented block, Expected indented block, Expected expression, Expected expression, Expected expression, "c" is not defined, "a" is not defined, "b" is not defined, IndentationError: expected an indented block after function definition on line 3 (4231512919.py, line 4)

correct the code by function and parameters to perform basic arithmetic operations which handles division by zero, non-string operation.print output for vaild and invalid cases

```
    Returns the result or None on error.
    """
    # Validate operation type
    if not isinstance(op, str):
        print(f"Invalid operation type: expected a string, got {type(op).__name__}")
        return None

    # Validate operands are numeric
    if not (isinstance(a, (int, float)) and isinstance(b, (int, float))):
        print(f"Invalid operands: expected numbers, got {type(a).__name__} and {type(b).__name__}")
        return None

    op_key = op.lower()
    symbols = {'add': '+', 'sub': '-', 'mul': '*', 'div': '/'}
    if op_key not in symbols:
        print(f"Unknown operation: '{op}'. Valid operations: {list(symbols.keys())}")
        return None

    # Handle division by zero
    if op_key == 'div' and b == 0:
        print("Error: Division by zero")
        return None

    # Compute result
    if op_key == 'add':
        result = a + b
    elif op_key == 'sub':
        result = a - b
    elif op_key == 'mul':
        result = a * b
    else:  # div
        result = a / b
```

Output :



```
5 + 3 = 8
Error: Division by zero
Invalid operands: expected numbers, got str and int
Invalid operation type: expected a string, got int
Unknown operation: 'power'. Valid operations: ['add', 'sub', 'mul', 'div']
```

Problem Statement 3: Enforcing Coding Standards

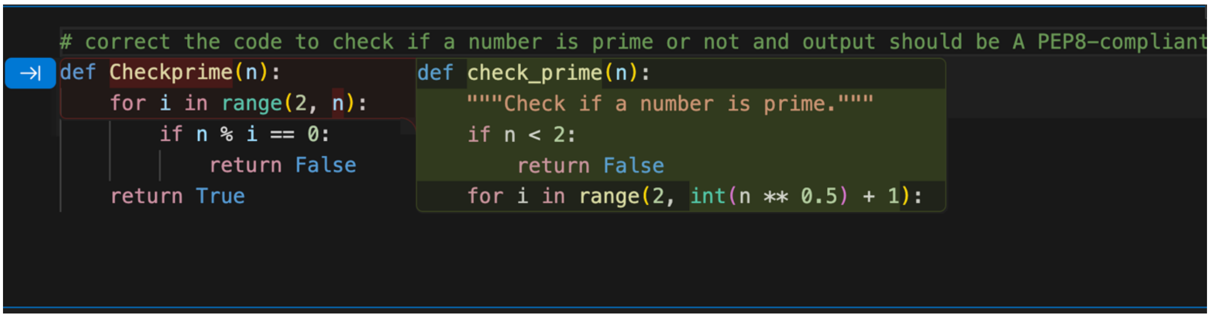
Scenario: A team project requires PEP8 compliance. A developer submits:

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

return False
return True
Instructions:
Verify the function works correctly for sample inputs.
Use an AI tool (e.g., ChatGPT, GitHub Copilot, or a PEP8 linter with AI explanation) to:
List all PEP8 violations.
Refactor the code (function name, spacing, indentation, naming).
Apply the AI-suggested changes and verify functionality is preserved.
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

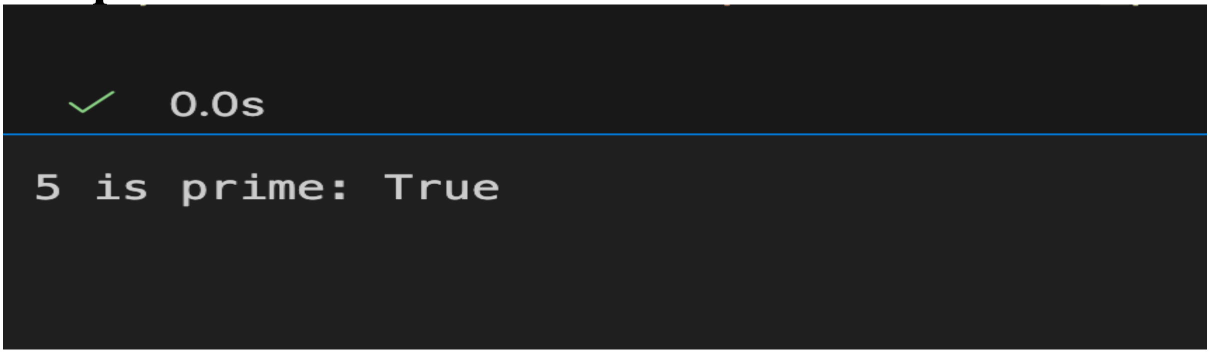
Prompt:

correct the code to check if a number is prime or not and output should be A PEP8-compliant version of the function



```
# correct the code to check if a number is prime or not and output should be A PEP8-compliant
def Checkprime(n):            def check_prime(n):
    for i in range(2, n):         """Check if a number is prime."""
        if n % i == 0:            if n < 2:
            return False              return False
    return True                   for i in range(2, int(n ** 0.5) + 1):
```

Output :



```
✓  0.0s

5 is prime: True
```

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]
```

1.Instructions:

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]
```

Prompt:

Optimized Prompt Act as a senior Python engineer reviewing code in a real GitHub project.Given this function submitted by a teammate:

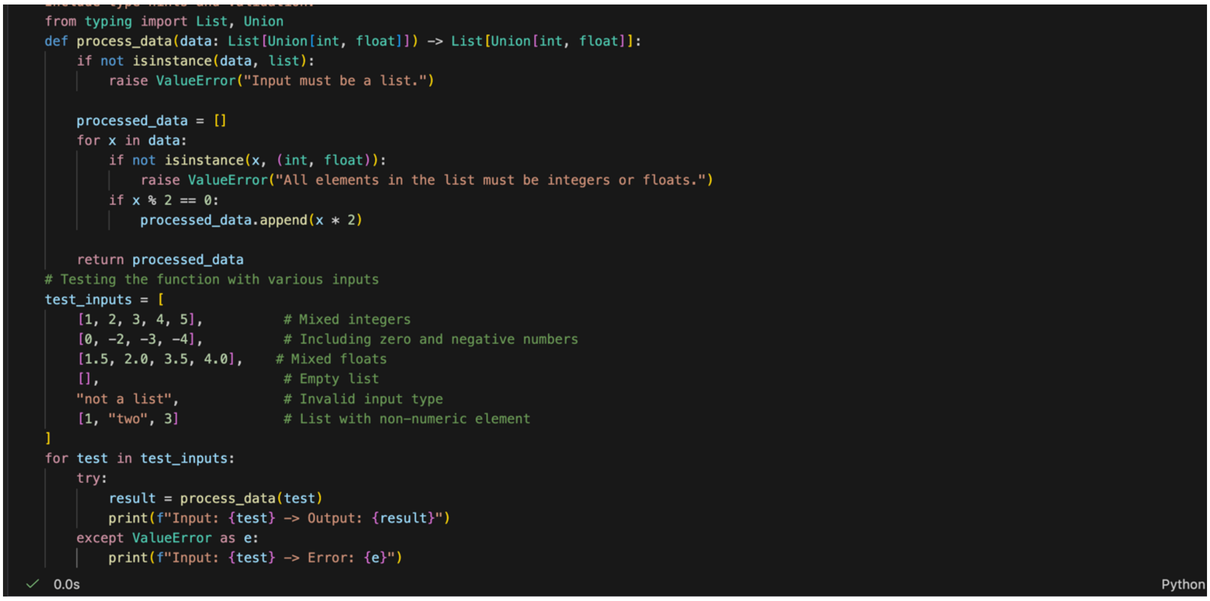
```
def processData(d):
   return [x * 2 for x in d if x % 2 == 0]
```

Do the following:Manual review:

Evaluate readability and naming.,modularity.

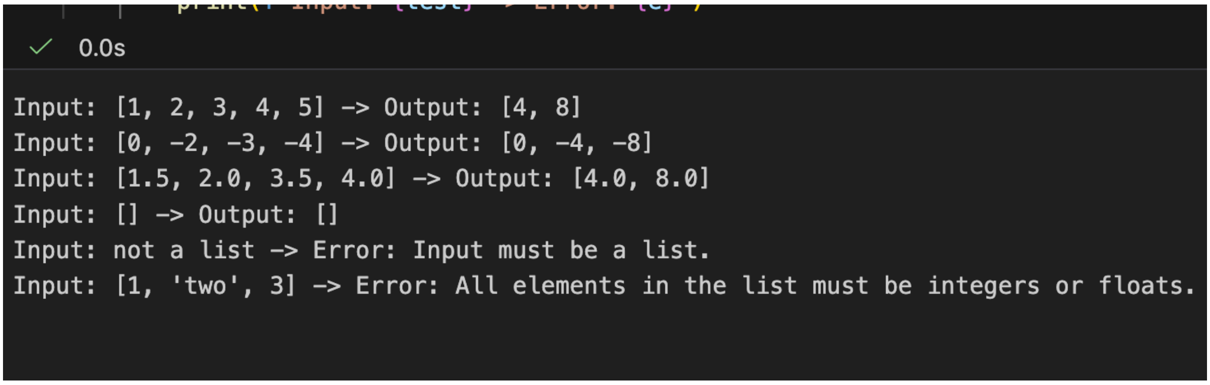
Identify edge cases (non-list input, empty list, non-integer elements, etc.).

suggest better function and variable names and clarify the function's purpose.Add input validation and proper type hints.Suggest how to generalize the function Refactor the code based on the review:
Provide a clean, production-quality version.
Include type hints and validation.



```
from typing import List, Union
def process_data(data: List[Union[int, float]]) -> List[Union[int, float]]:
    if not isinstance(data, list):
        raise ValueError("Input must be a list.")

    processed_data = []
    for x in data:
        if not isinstance(x, (int, float)):
            raise ValueError("All elements in the list must be integers or floats.")
        if x % 2 == 0:
            processed_data.append(x * 2)

    return processed_data
# Testing the function with various inputs
test_inputs = [
    [1, 2, 3, 4, 5],            # Mixed integers
    [0, -2, -3, -4],            # Including zero and negative numbers
    [1.5, 2.0, 3.5, 4.0],      # Mixed floats
    [],                         # Empty list
    "not a list",               # Invalid input type
    [1, "two", 3]               # List with non-numeric element
]
for test in test_inputs:
    try:
        result = process_data(test)
        print(f"Input: {test} -> Output: {result}")
    except ValueError as e:
        print(f"Input: {test} -> Error: {e}")
```

Output :



```
Input: [1, 2, 3, 4, 5] -> Output: [4, 8]
Input: [0, -2, -3, -4] -> Output: [0, -4, -8]
Input: [1.5, 2.0, 3.5, 4.0] -> Output: [4.0, 8.0]
Input: [] -> Output: []
Input: not a list -> Error: Input must be a list.
Input: [1, 'two', 3] -> Error: All elements in the list must be integers or floats.
```

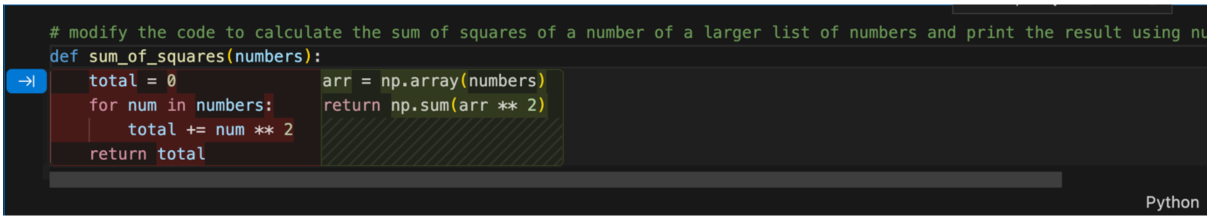
Problem Statement 5: — AI-Assisted Performance Optimization
Scenario: You are given a function that processes a list of integers, but it runs slowly on large datasets:

```
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)

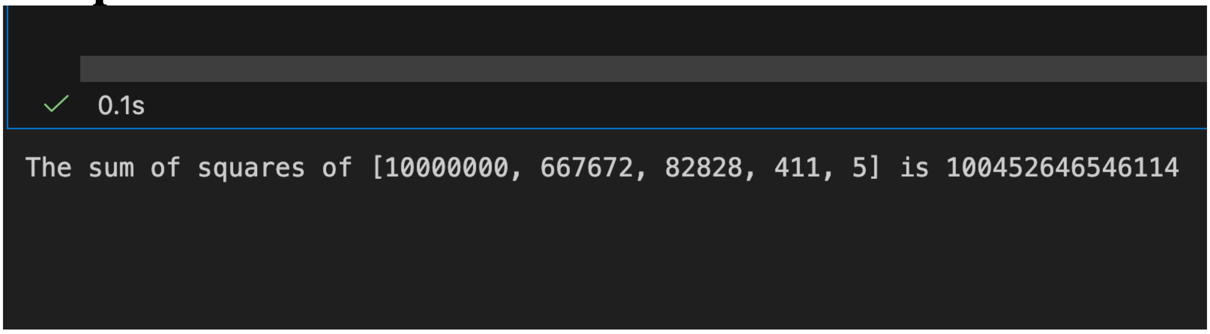
Prompt :

modify the code to calculate the sum of squares of a number of larger list of numbers and print the result using numpy library



```
# modify the code to calculate the sum of squares of a number of a larger list of numbers and print the result using nu
def sum_of_squares(numbers):
    total = 0                  arr = np.array(numbers)
    for num in numbers:        return np.sum(arr ** 2)
        total += num ** 2
    return total
```

Output :



```
The sum of squares of [10000000, 667672, 82828, 411, 5] is 100452646546114
```