LAB-10.3

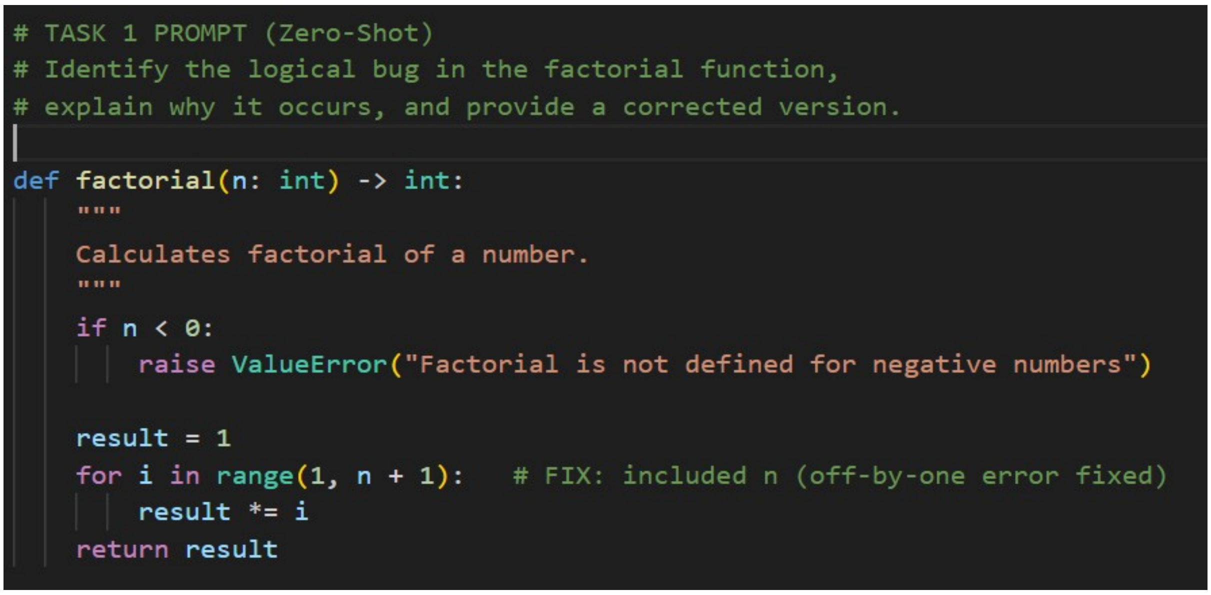
Name: G. Sai teja

Batch:41

Roll-No:2303A52135

Problem Statement 1: AI-Assisted Bug Detection
PROMPT: Identify the logical bug in the factorial function. explain why it occurs, and provide a corrected version.

Code:

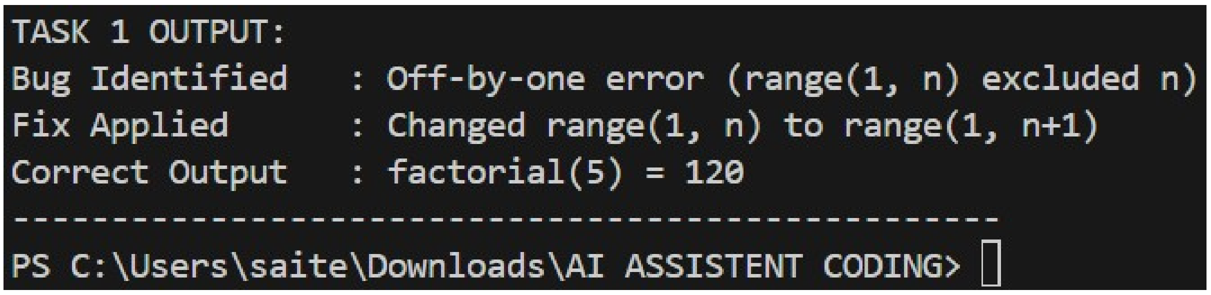


```python
# TASK 1 PROMPT (Zero-Shot)
# Identify the logical bug in the factorial function,
# explain why it occurs, and provide a corrected version.

def factorial(n: int) -> int:
    """
    Calculates factorial of a number.
    """
    if n < 0:
        raise ValueError("Factorial is not defined for negative numbers")

    result = 1
    for i in range(1, n + 1):   # FIX: included n (off-by-one error fixed)
        result *= i
    return result
```

OUTPUT:

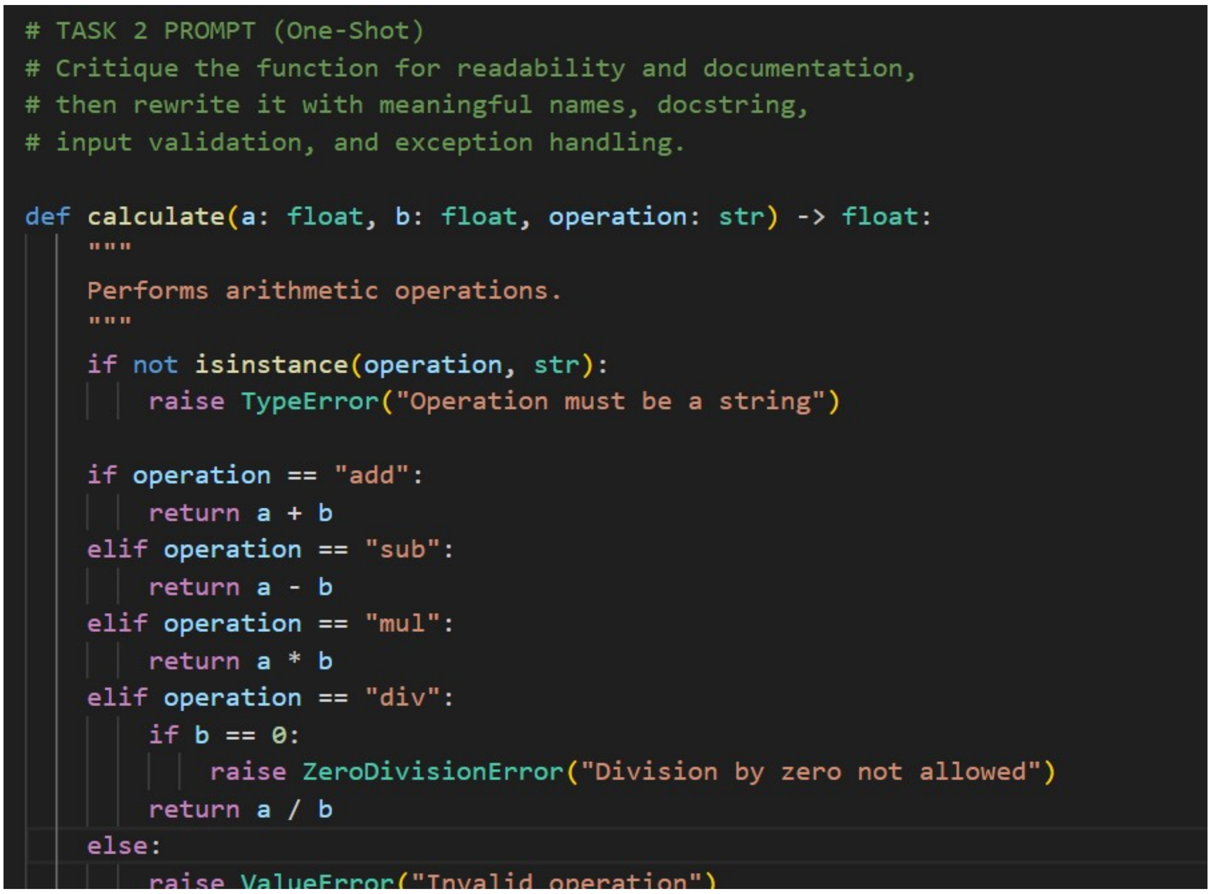


```
TASK 1 OUTPUT:
Bug Identified    : Off-by-one error (range(1, n) excluded n)
Fix Applied       : Changed range(1, n) to range(1, n+1)
Correct Output    : factorial(5) = 120
--------------------------------------------------
PS C:\Users\saite\Downloads\AI ASSISTENT CODING> 
```

Justification:
The factorial code had an off-by-one error because the loop did not include the number n.AI helped identify the logical mistake quickly.The loop range was corrected to include n.This produced the correct output for factorial calculation.

Problem Statement 2: Task 2 — Improving Readability & Documentation
PROMPT: Critique the function for readability and documentation. then rewrite it with meaningful names, docstring.input validation, and exception handling.

CODE:

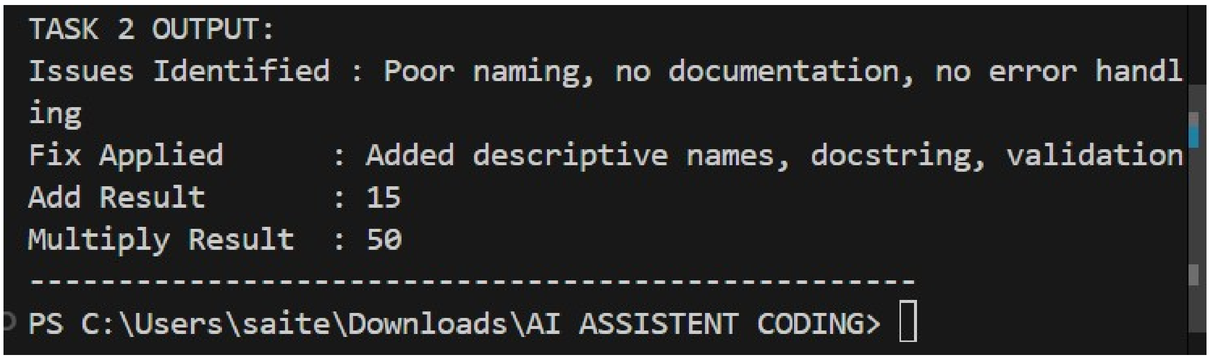


```python
# TASK 2 PROMPT (One-Shot)
# Critique the function for readability and documentation,
# then rewrite it with meaningful names, docstring,
# input validation, and exception handling.

def calculate(a: float, b: float, operation: str) -> float:
    """
    Performs arithmetic operations.
    """
    if not isinstance(operation, str):
        raise TypeError("Operation must be a string")

    if operation == "add":
        return a + b
    elif operation == "sub":
        return a - b
    elif operation == "mul":
        return a * b
    elif operation == "div":
        if b == 0:
            raise ZeroDivisionError("Division by zero not allowed")
        return a / b
    else:
        raise ValueError("Invalid operation")
```

OUTPUT:



```
TASK 2 OUTPUT:
Issues Identified : Poor naming, no documentation, no error handl
ing
Fix Applied       : Added descriptive names, docstring, validation
Add Result        : 15
Multiply Result   : 50
--------------------------------------------------
PS C:\Users\saite\Downloads\AI ASSISTENT CODING>
```

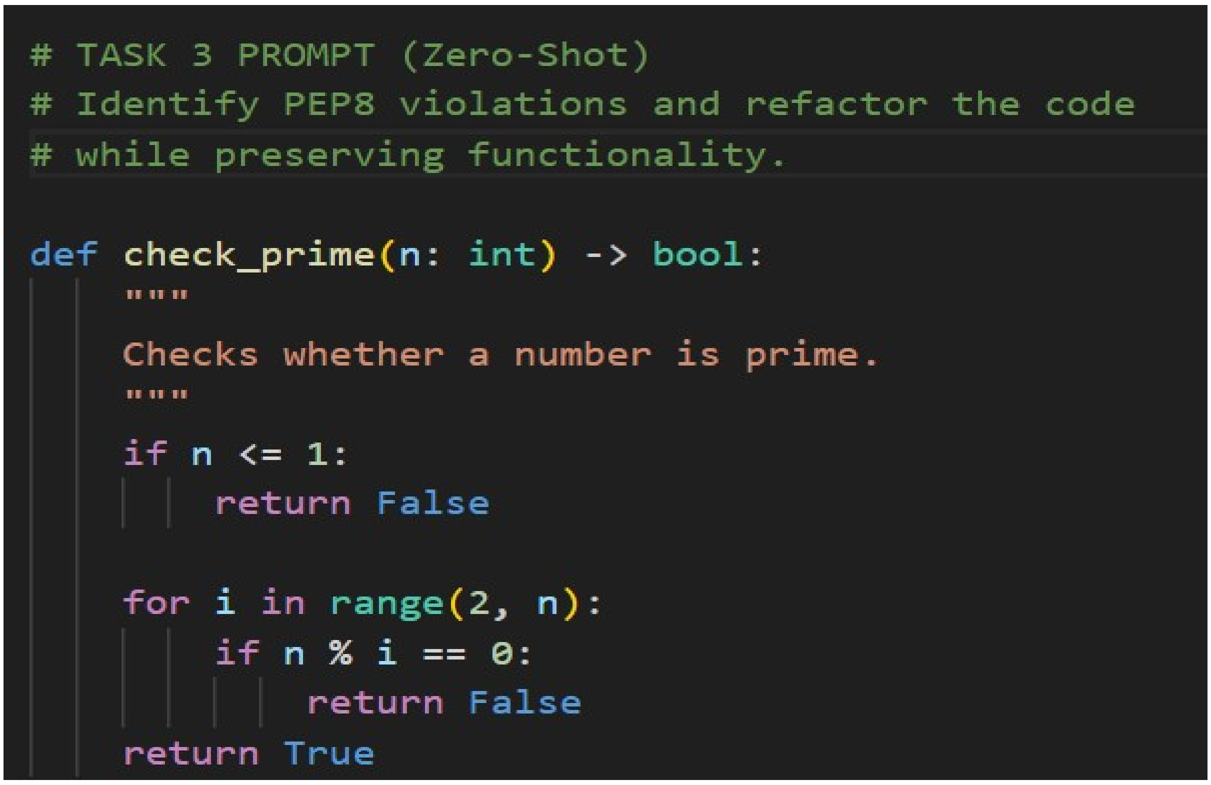
Justification:

The original function had unclear variable names and no documentation.AI suggested better naming and added a proper docstring.Error handling and input validation were included.This made the function more readable and reliable.

Problem Statement 3: Enforcing Coding Standards

PROMPT:
Identify PEP8 violations and refactor the code. while preserving functionality.

CODE:

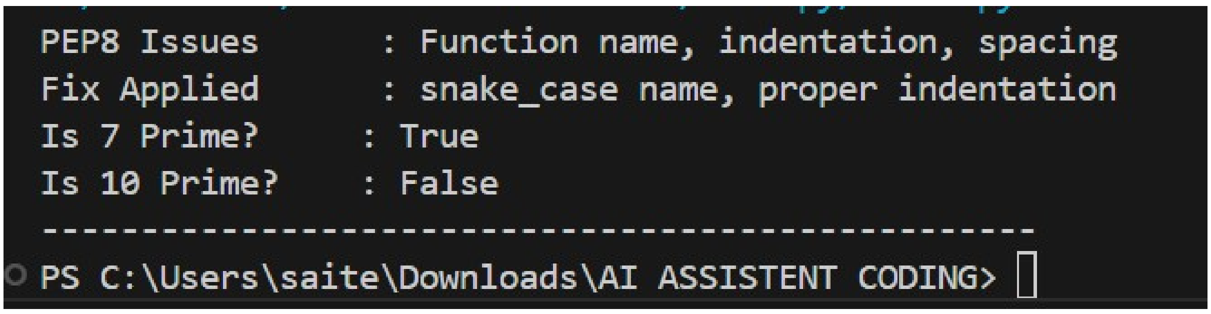


```python
# TASK 3 PROMPT (Zero-Shot)
# Identify PEP8 violations and refactor the code
# while preserving functionality.

def check_prime(n: int) -> bool:
    """
    Checks whether a number is prime.
    """
    if n <= 1:
        return False

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

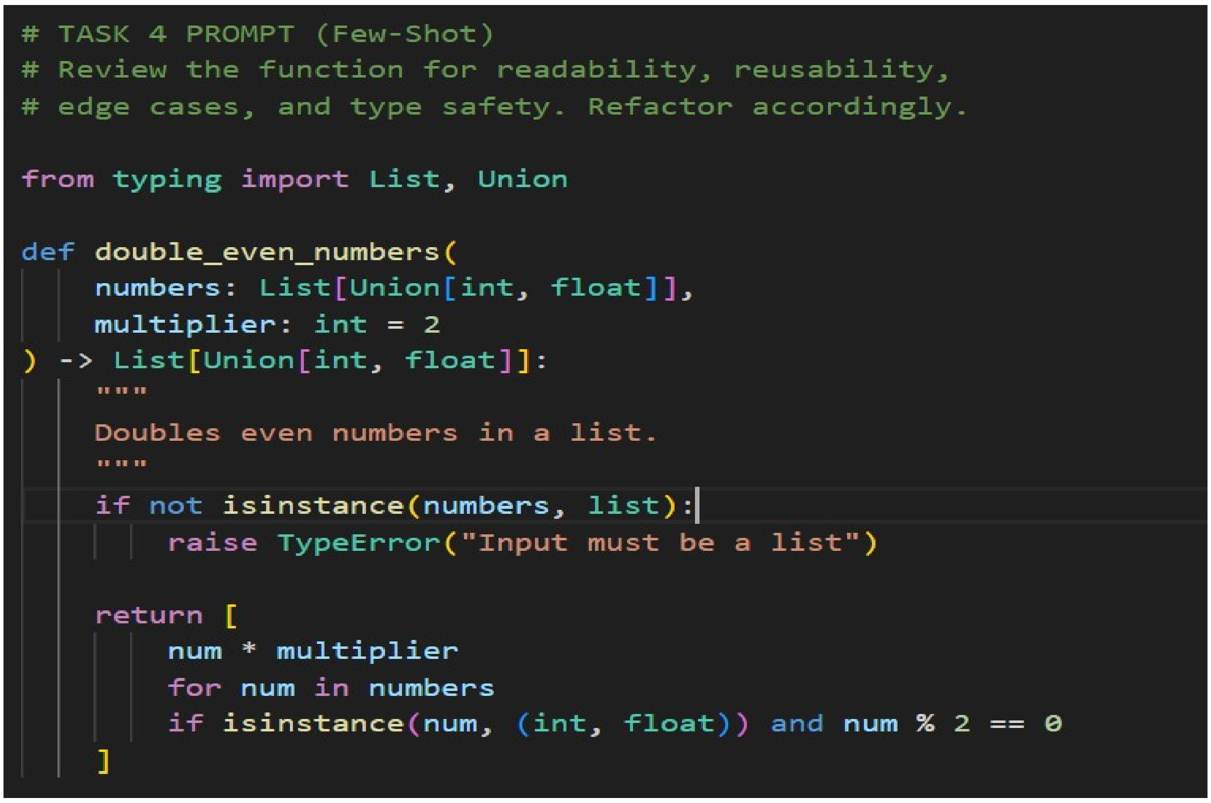
OUTPUT:



```
PEP8 Issues      : Function name, indentation, spacing
Fix Applied      : snake_case name, proper indentation
Is 7 Prime?     : True
Is 10 Prime?    : False
---------------------------------------------------
PS C:\Users\saite\Downloads\AI ASSISTENT CODING>
```

Justification: The original code violated PEP8 naming and indentation rules.
AI identified these style issues accurately.The function was refactored using snake_case and proper formatting.Functionality was preserved with improved code quality.

Problem Statement 4: AI as a Code Reviewer in Real Projects
PROMPT: Review the function for readability, reusability. edge cases, and type safety. Refactor accordingly.
CODE:



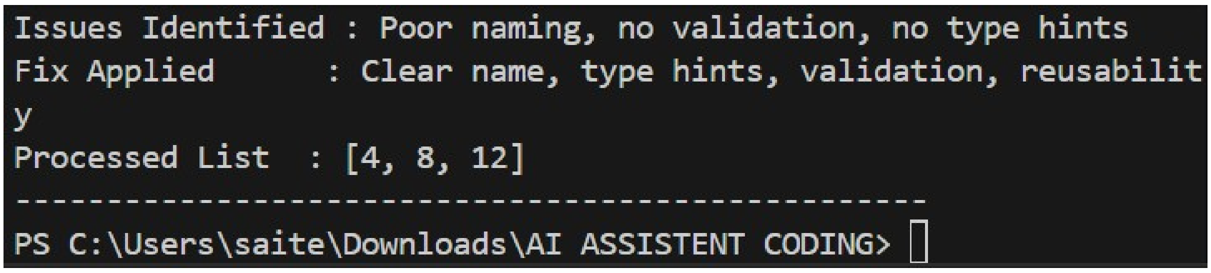
```python
# TASK 4 PROMPT (Few-Shot)
# Review the function for readability, reusability,
# edge cases, and type safety. Refactor accordingly.

from typing import List, Union

def double_even_numbers(
    numbers: List[Union[int, float]],
    multiplier: int = 2
) -> List[Union[int, float]]:
    """
    Doubles even numbers in a list.
    """
    if not isinstance(numbers, list):
        raise TypeError("Input must be a list")

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

OUTPUT:

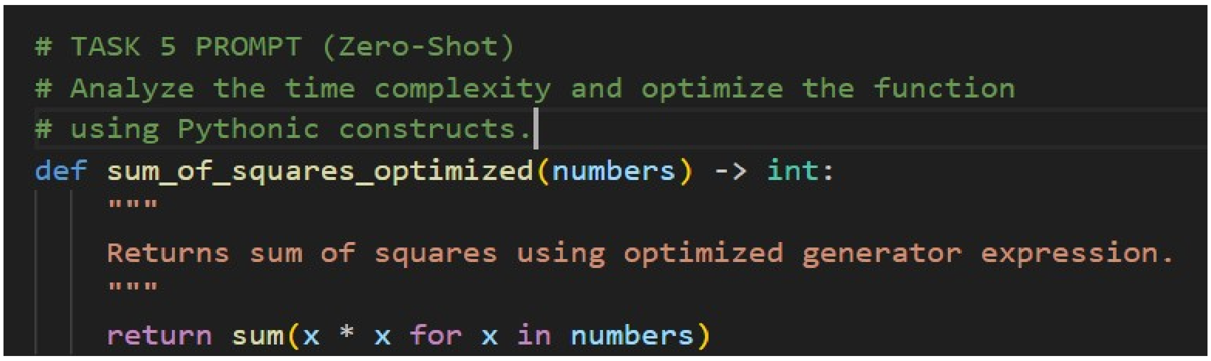


```
Issues Identified : Poor naming, no validation, no type hints
Fix Applied       : Clear name, type hints, validation, reusabilit
y
Processed List   : [4, 8, 12]
--------------------------------------------------
PS C:\Users\saite\Downloads\AI ASSISTENT CODING>
```

Justification:
The original function lacked clarity and input validation.AI recommended meaningful names and type hints.Validation and reusability were added.This improved robustness and real-world usability.

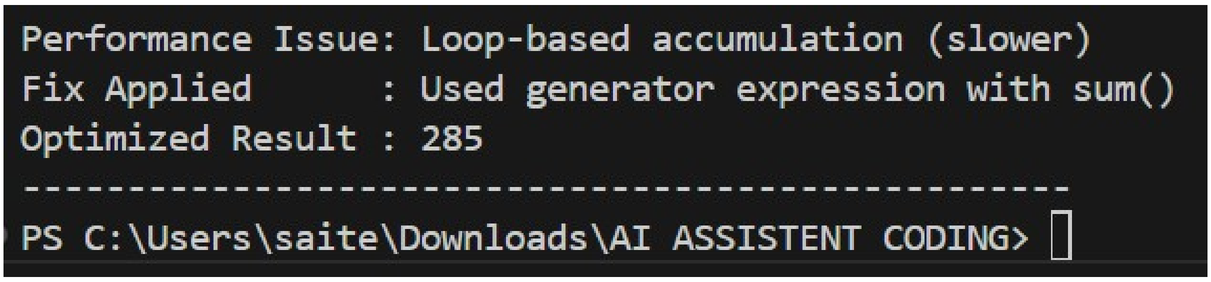
Problem Statement 5: AI-Assisted Performance Optimization
PROMPT: Analyze the time complexity and optimize the function.using Pythonic constructs.

CODE:



```python
# TASK 5 PROMPT (Zero-Shot)
# Analyze the time complexity and optimize the function
# using Pythonic constructs.
def sum_of_squares_optimized(numbers) -> int:
    """
    Returns sum of squares using optimized generator expression.
    """
    return sum(x * x for x in numbers)
```

OUTPUT:



```
Performance Issue: Loop-based accumulation (slower)
Fix Applied      : Used generator expression with sum()
Optimized Result : 285
--------------------------------------------------
PS C:\Users\saite\Downloads\AI ASSISTENT CODING> 
```

Justification:
The original function used a manual loop which was slower.AI analyzed the time complexity and suggested optimization. A generator expression with sum() was used.
This improved performance while keeping the code readable.