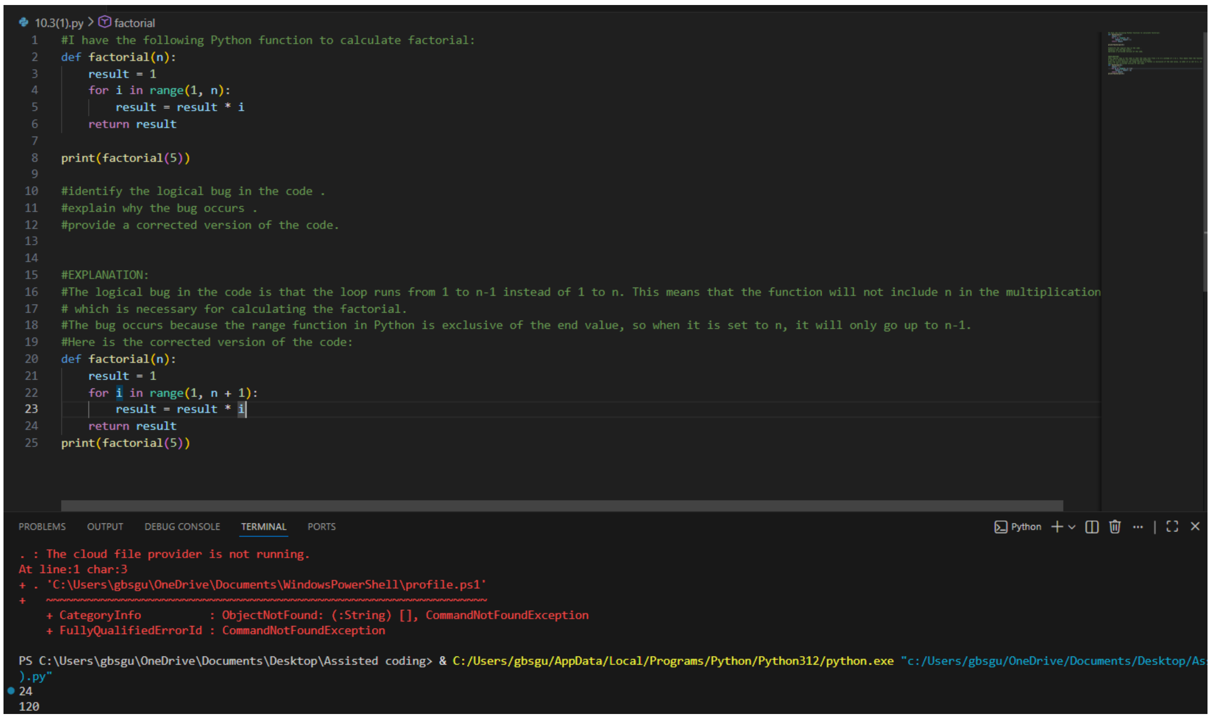
# Assignment-10.3

## 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
```



10.3(1).py > factorial

```
#I have the following Python function to calculate factorial:
def factorial(n):
    result = 1
    for i in range(1, n):
        result = result * i
    return result

print(factorial(5))

#identify the logical bug in the code .
#explain why the bug occurs .
#provide a corrected version of the code.


#EXPLANATION:
#The logical bug in the code is that the loop runs from 1 to n-1 instead of 1 to n. This means that the function will not include n in the multiplication
# which is necessary for calculating the factorial.
#The bug occurs because the range function in Python is exclusive of the end value, so when it is set to n, it will only go up to n-1.
#Here is the corrected version of the code:
def factorial(n):
    result = 1
    for i in range(1, n + 1):
        result = result * i
    return result
print(factorial(5))
```

```
. : The cloud file provider is not running.
At line:1 char:3
+ . 'C:\Users\gbsgu\OneDrive\Documents\WindowsPowerShell\profile.ps1'
+   ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : ObjectNotFound: (:String) [], CommandNotFoundException
    + FullyQualifiedErrorId : CommandNotFoundException

PS C:\Users\gbsgu\OneDrive\Documents\Desktop\Assisted coding> & C:/Users/gbsgu/AppData/Local/Programs/Python/Python312/python.exe "c:/Users/gbsgu/OneDrive/Documents/Desktop/As
).py"
24
120
```

Comparison :The normal code almost works but forgets the last number.

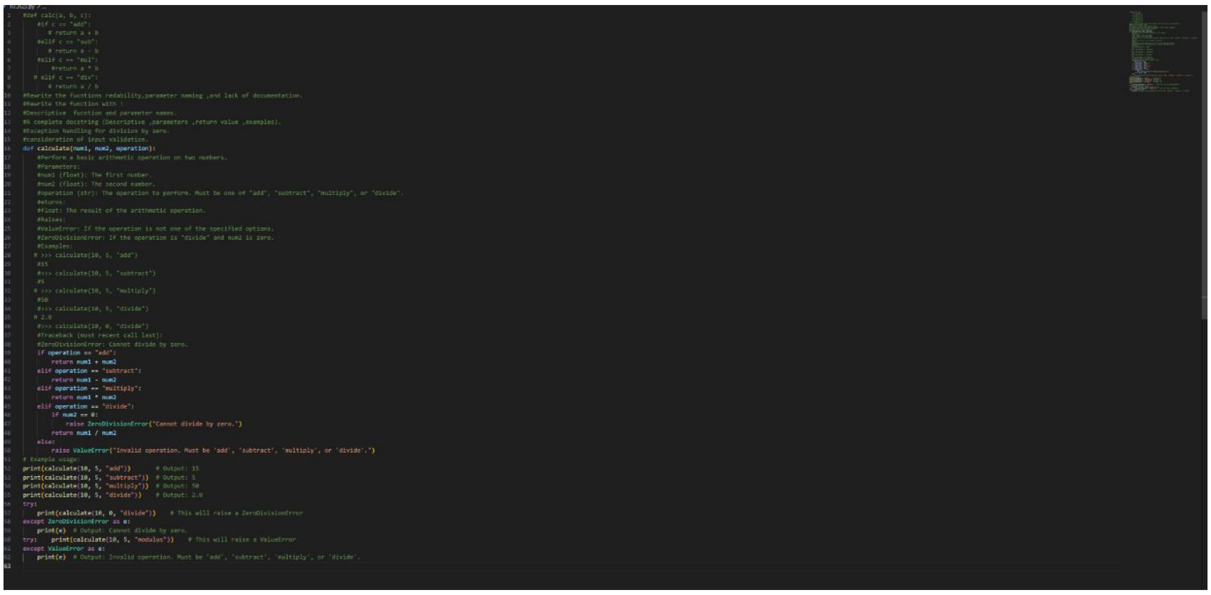
The AI version fixes that small mistake and makes the factorial calculation correct.

## Problem Statement 2: Task 2 — Improving Readability & Documentation

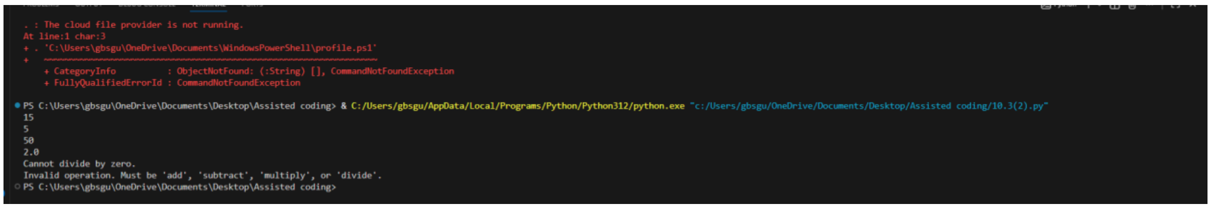
**Scenario:**The following code works but is poorly written:

.

```
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":
```



```
#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
#Rewrite the fucntions redability,parameter naming ,and lack of documentation.
#Rewrite the function with :
#Descriptive  fucntion and parameter names.
#A complete docstring (Descriptive ,parameters ,return value ,examples).
#Exception handling for division by zero.
#consideration of input validation.
def calculate(num1, num2, operation):
    #Perform a basic arithmetic operation on two numbers.
    #Parameters:
    #num1 (float): The first number.
    #num2 (float): The second number.
    #operation (str): The operation to perform. Must be one of "add", "subtract", "multiply", or "divide".
    #eturns:
    #float: The result of the arithmetic operation.
    #Raises:
    #ValueError: If the operation is not one of the specified options.
    #ZeroDivisionError: If the operation is "divide" and num2 is zero.
    #Examples:
   # >>> calculate(10, 5, "add")
    #15
    #>>> calculate(10, 5, "subtract")
    #5
   # >>> calculate(10, 5, "multiply")
    #50
    #>>> calculate(10, 5, "divide")
   # 2.0
    #>>> calculate(10, 0, "divide")
    #Traceback (most recent call last):
    #ZeroDivisionError: Cannot divide by zero.
    if operation == "add":
        return num1 + num2
    elif operation == "subtract":
        return num1 - num2
    elif operation == "multiply":
        return num1 * num2
    elif operation == "divide":
        if num2 == 0:
            raise ZeroDivisionError("Cannot divide by zero.")
        return num1 / num2
    else:
        raise ValueError("Invalid operation. Must be 'add', 'subtract', 'multiply', or 'divide'.")
# Example usage:
print(calculate(10, 5, "add"))       # Output: 15
print(calculate(10, 5, "subtract"))  # Output: 5
print(calculate(10, 5, "multiply"))  # Output: 50
print(calculate(10, 5, "divide"))    # Output: 2.0
try:
    print(calculate(10, 0, "divide"))    # This will raise a ZeroDivisionError
except ZeroDivisionError as e:
    print(e)  # Output: Cannot divide by zero.
try:    print(calculate(10, 5, "modulus"))    # This will raise a ValueError
except ValueError as e:
    print(e)  # Output: Invalid operation. Must be 'add', 'subtract', 'multiply', or 'divide'.
```



```
. : The cloud file provider is not running.
At line:1 char:3
+ . 'C:\Users\gbsgu\OneDrive\Documents\WindowsPowerShell\profile.ps1'
+   ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : ObjectNotFound: (:String) [], CommandNotFoundException
    + FullyQualifiedErrorId : CommandNotFoundException

PS C:\Users\gbsgu\OneDrive\Documents\Desktop\Assisted coding> & C:/Users/gbsgu/AppData/Local/Programs/Python/Python312/python.exe "c:/Users/gbsgu/OneDrive/Documents/Desktop/Assisted coding/10.3(2).py"
15
5
50
2.0
Cannot divide by zero.
Invalid operation. Must be 'add', 'subtract', 'multiply', or 'divide'.
PS C:\Users\gbsgu\OneDrive\Documents\Desktop\Assisted coding>
```

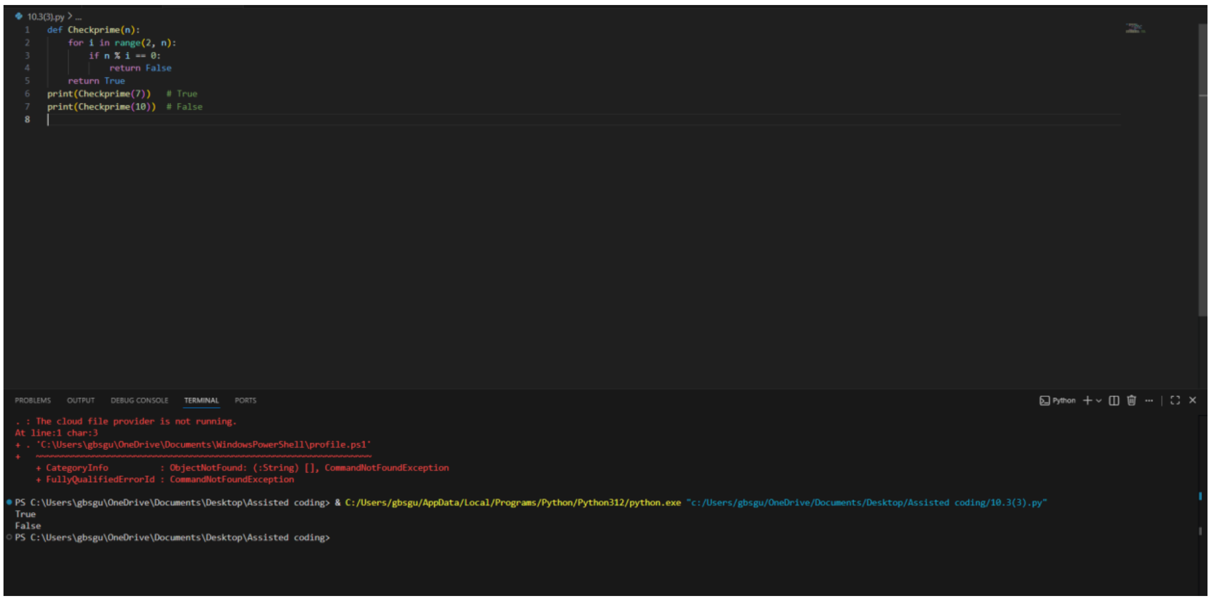
Explanation: The original function works, but it is not very clear. The function name calc and parameters a, b, c are confusing, and there is no documentation explaining what the function does. It also does not handle errors like division by zero or invalid operations.

The AI-improved version uses clear names, adds a proper docstring, and includes input validation and exception handling. This makes the function more readable, safer, and more professional.

**Problem Statement 3: Enforcing Coding Standards**

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

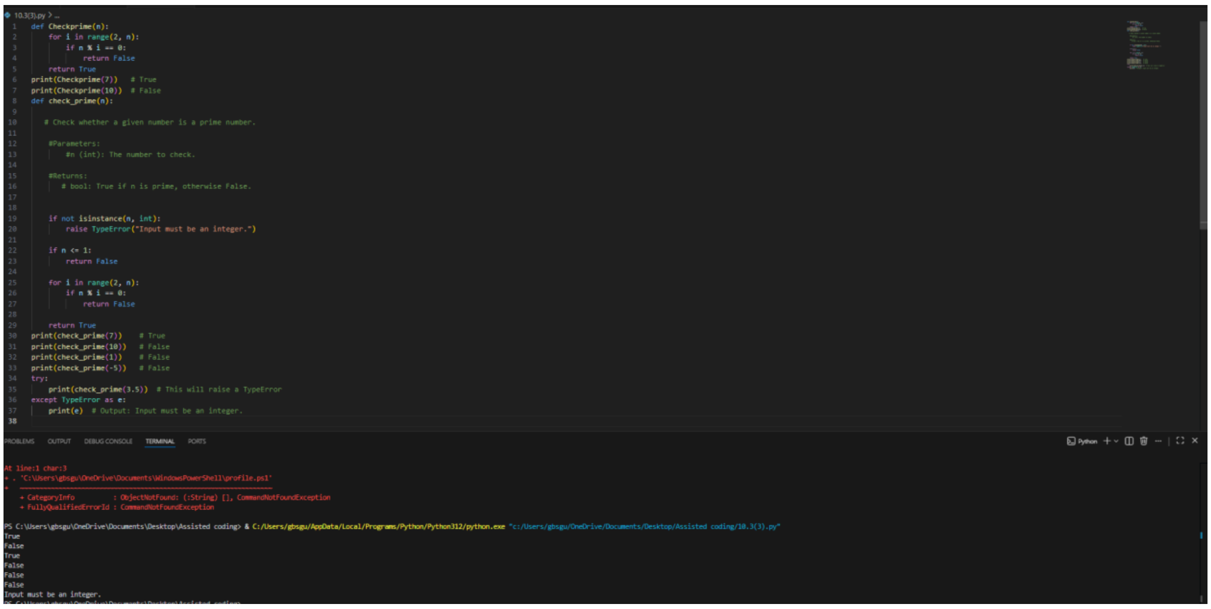
```
def Checkprime(n):
    for i in range(2, n):
        if n % i == 0:
            return False
    return True
```



```
def Checkprime(n):
    for i in range(2, n):
        if n % i == 0:
            return False
    return True
print(Checkprime(7))   # True
print(Checkprime(10))  # False
```

```
. : The cloud file provider is not running.
At line:1 char:3
+ . 'C:\Users\gbsgu\OneDrive\Documents\WindowsPowerShell\profile.ps1'
+   ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : ObjectNotFound: (:String) [], CommandNotFoundException
    + FullyQualifiedErrorId : CommandNotFoundException

PS C:\Users\gbsgu\OneDrive\Documents\Desktop\Assisted coding> & C:/Users/gbsgu/AppData/Local/Programs/Python/Python312/python.exe "c:/Users/gbsgu/OneDrive/Documents/Desktop/Assisted coding/10.3(3).py"
True
False
PS C:\Users\gbsgu\OneDrive\Documents\Desktop\Assisted coding>
```



10.3(3).py > ...

```python
def Checkprime(n):
    for i in range(2, n):
        if n % i == 0:
            return False
    return True
print(Checkprime(7))   # True
print(Checkprime(10))  # False
def check_prime(n):

   # Check whether a given number is a prime number.

    #Parameters:
        #n (int): The number to check.

    #Returns:
      # bool: True if n is prime, otherwise False.


    if not isinstance(n, int):
        raise TypeError("Input must be an integer.")

    if n <= 1:
        return False

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

    return True
print(check_prime(7))    # True
print(check_prime(10))   # False
print(check_prime(1))    # False
print(check_prime(-5))   # False
try:
    print(check_prime(3.5))  # This will raise a TypeError
except TypeError as e:
    print(e)  # Output: Input must be an integer.
```

```
At line:1 char:3
+ . 'C:\Users\gbsgu\OneDrive\Documents\WindowsPowerShell\profile.ps1'
+   ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : ObjectNotFound: (:String) [], CommandNotFoundException
    + FullyQualifiedErrorId : CommandNotFoundException

PS C:\Users\gbsgu\OneDrive\Documents\Desktop\Assisted coding> & C:/Users/gbsgu/AppData/Local/Programs/Python/Python312/python.exe "c:/Users/gbsgu/OneDrive/Documents/Desktop/Assisted coding/10.3(3).py"
True
False
True
False
False
False
Input must be an integer.
```

Automated AI reviews can significantly streamline code reviews in large teams. AI tools can instantly detect PEP8 violations, naming issues, missing documentation, and logical edge cases. This reduces manual effort, improves consistency, and allows developers to focus more on logic and design rather than formatting issues.

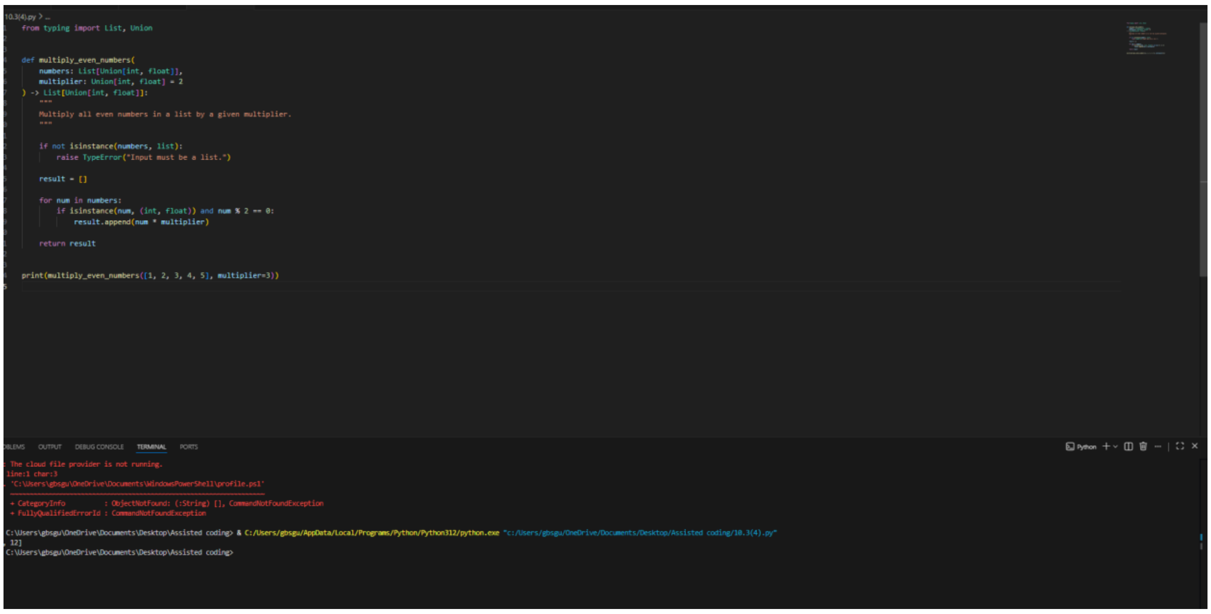
AI-assisted reviews make the development process faster, cleaner, and more standardized.

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

**Scenario:**

In a GitHub project, a teammate submits:

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



10.3(4).py > ...

```python
from typing import List, Union


def multiply_even_numbers(
    numbers: List[Union[int, float]],
    multiplier: Union[int, float] = 2
) -> List[Union[int, float]]:
    """
    Multiply all even numbers in a list by a given multiplier.
    """

    if not isinstance(numbers, list):
        raise TypeError("Input must be a list.")

    result = []

    for num in numbers:
        if isinstance(num, (int, float)) and num % 2 == 0:
            result.append(num * multiplier)

    return result


print(multiply_even_numbers([1, 2, 3, 4, 5], multiplier=3))
```

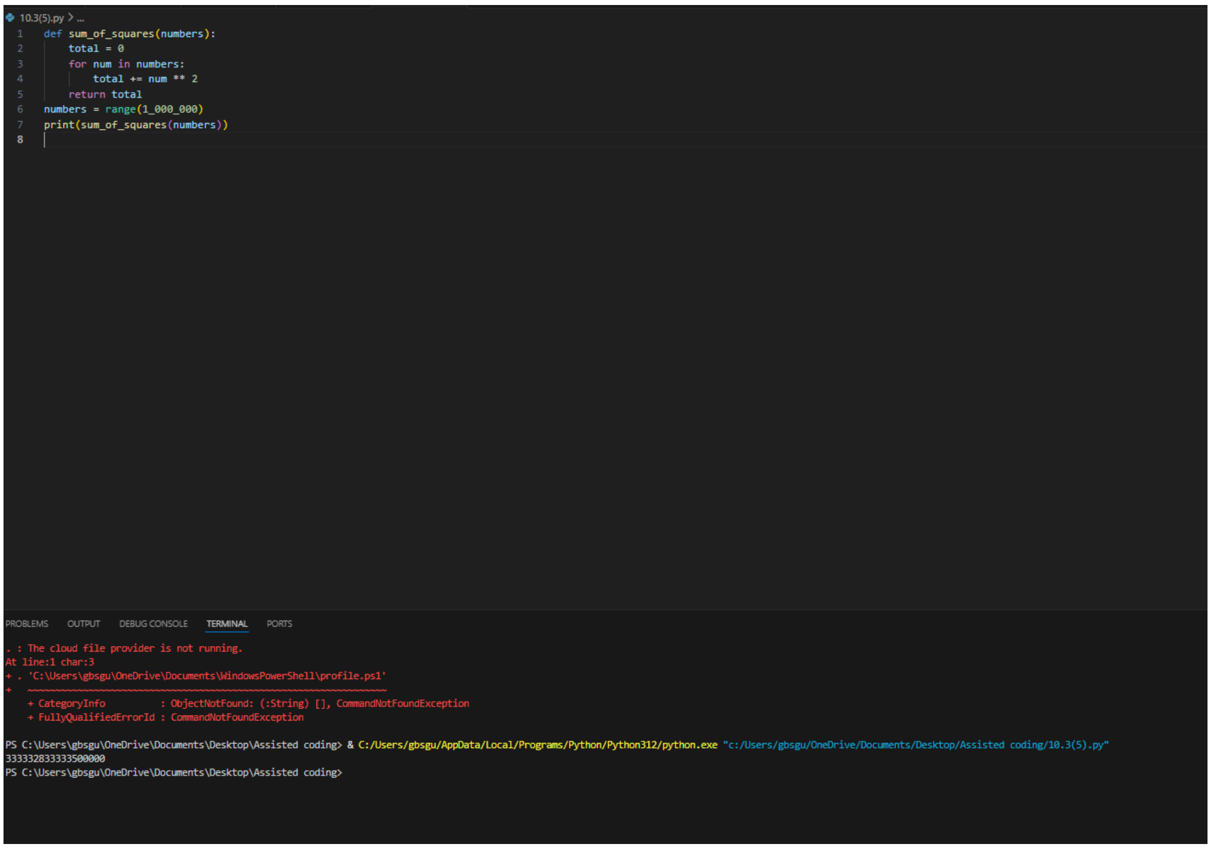
```
: The cloud file provider is not running.
 line:1 char:3
. 'C:\Users\gbsgu\OneDrive\Documents\WindowsPowerShell\profile.ps1'
  ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
  + CategoryInfo          : ObjectNotFound: (:String) [], CommandNotFoundException
  + FullyQualifiedErrorId : CommandNotFoundException

 C:\Users\gbsgu\OneDrive\Documents\Desktop\Assisted coding> & C:/Users/gbsgu/AppData/Local/Programs/Python/Python312/python.exe "c:/Users/gbsgu/OneDrive/Documents/Desktop/Assisted coding/10.3(4).py"
, 12]
 C:\Users\gbsgu\OneDrive\Documents\Desktop\Assisted coding>
```

**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:

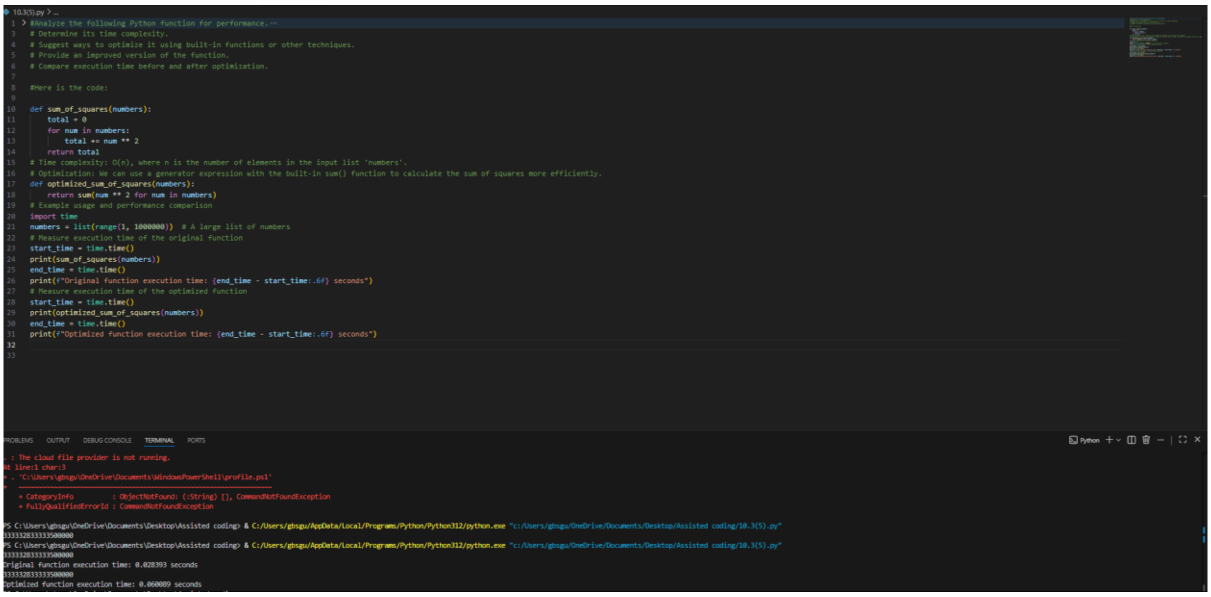
```
def sum_of_squares(numbers):
    total = 0
    for num in numbers:
        total += num ** 2
    return total
```



```python
def sum_of_squares(numbers):
    total = 0
    for num in numbers:
        total += num ** 2
    return total
numbers = range(1_000_000)
print(sum_of_squares(numbers))
```

```
. : The cloud file provider is not running.
At line:1 char:3
+ . 'C:\Users\gbsgu\OneDrive\Documents\WindowsPowerShell\profile.ps1'
+   ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : ObjectNotFound: (:String) [], CommandNotFoundException
    + FullyQualifiedErrorId : CommandNotFoundException

PS C:\Users\gbsgu\OneDrive\Documents\Desktop\Assisted coding> & C:/Users/gbsgu/AppData/Local/Programs/Python/Python312/python.exe "c:/Users/gbsgu/OneDrive/Documents/Desktop/Assisted coding/10.3(5).py"
333332833333500000
PS C:\Users\gbsgu\OneDrive\Documents\Desktop\Assisted coding>
```



```python
#Analyze the following Python function for performance.
# Determine its time complexity.
# Suggest ways to optimize it using built-in functions or other techniques.
# Provide an improved version of the function.
# Compare execution time before and after optimization.

#Here is the code:

def sum_of_squares(numbers):
    total = 0
    for num in numbers:
        total += num ** 2
    return total
# Time complexity: O(n), where n is the number of elements in the input list 'numbers'.
# Optimization: We can use a generator expression with the built-in sum() function to calculate the sum of squares more efficiently.
def optimized_sum_of_squares(numbers):
    return sum(num ** 2 for num in numbers)
# Example usage and performance comparison
import time
numbers = list(range(1, 1000000))  # A large list of numbers
# Measure execution time of the original function
start_time = time.time()
print(sum_of_squares(numbers))
end_time = time.time()
print(f"Original function execution time: {end_time - start_time:.6f} seconds")
# Measure execution time of the optimized function
start_time = time.time()
print(optimized_sum_of_squares(numbers))
end_time = time.time()
print(f"Optimized function execution time: {end_time - start_time:.6f} seconds")
```

```
. : The cloud file provider is not running.
At line:1 char:3
+ . 'C:\Users\gbsgu\OneDrive\Documents\WindowsPowerShell\profile.ps1'
+   ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : ObjectNotFound: (:String) [], CommandNotFoundException
    + FullyQualifiedErrorId : CommandNotFoundException

PS C:\Users\gbsgu\OneDrive\Documents\Desktop\Assisted coding> & C:/Users/gbsgu/AppData/Local/Programs/Python/Python312/python.exe "c:/Users/gbsgu/OneDrive/Documents/Desktop/Assisted coding/10.3(5).py"
333332833333500000
PS C:\Users\gbsgu\OneDrive\Documents\Desktop\Assisted coding> & C:/Users/gbsgu/AppData/Local/Programs/Python/Python312/python.exe "c:/Users/gbsgu/OneDrive/Documents/Desktop/Assisted coding/10.3(5).py"
333332833333500000
Original function execution time: 0.028393 seconds
333332833333500000
Optimized function execution time: 0.060089 seconds
```

Explanation:

The original function calculates the sum of squares using a manual loop. Its time complexity is **O(n)** because it processes each element once. While the logic is correct, it can be slightly improved by using Python's built-in sum() function with a generator expression. This version is more readable, more Pythonic, and usually a bit faster because sum() is implemented in optimized C code. The overall time complexity remains **O(n)**, but performance and clarity improve.