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: Dr. Rishabh Mittal	**Instructor(s) Name:**	
Course Code: 23CS002PC304	**Course Title:** AI Assisted Coding	**Year/Sem:** III/II
Regulation: R23	**Date and Day of Assignment:** Week3 – Wednesday	**Duration:** 2 Hours
Time(s): 23CSBTB01 To 23CSBTB52	**Applicable to Batches:** All batches	
Assignment Number: 9.3 (Present assignment number) / **24** (Total number of assignments)		

ASSIGNMENT-9.3

Name:	P.Manikanta
HT. No:	2303A51271
Batch:	8

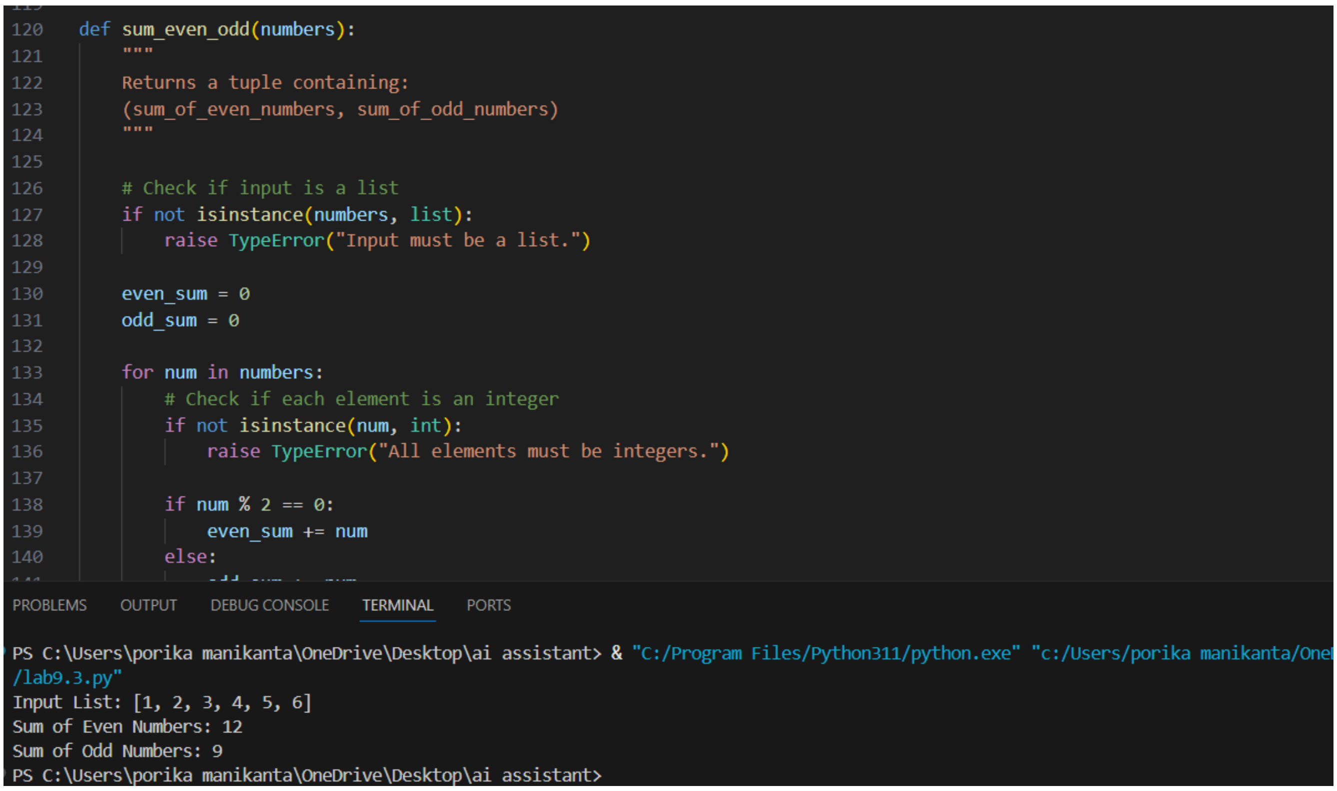
Lab 9: Documentation Generation – Automatic Documentation and Code Comments

Task 1: Basic Docstring Generation

Scenario: You are developing a utility function that processes numerical lists and must be properly documented for future maintenance.

Prompt: # Write a Python function to return the sum of even numbers and sum of odd numbers in a given list

Code:



```python
def sum_even_odd(numbers):
    """
    Returns a tuple containing:
    (sum_of_even_numbers, sum_of_odd_numbers)
    """

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

    even_sum = 0
    odd_sum = 0

    for num in numbers:
        # Check if each element is an integer
        if not isinstance(num, int):
            raise TypeError("All elements must be integers.")

        if num % 2 == 0:
            even_sum += num
        else:
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\porika manikanta\OneDrive\Desktop\ai assistant> & "C:/Program Files/Python311/python.exe" "c:/Users/porika manikanta/One
/lab9.3.py"
Input List: [1, 2, 3, 4, 5, 6]
Sum of Even Numbers: 12
Sum of Odd Numbers: 9
PS C:\Users\porika manikanta\OneDrive\Desktop\ai assistant>
```

Result:

The function sum_even_odd() successfully returns a tuple containing the sum of even and odd numbers. For input [1, 2, 3, 4, 5, 6]: Sum of Even Numbers: 12, Sum of Odd Numbers: 9.

Observation:

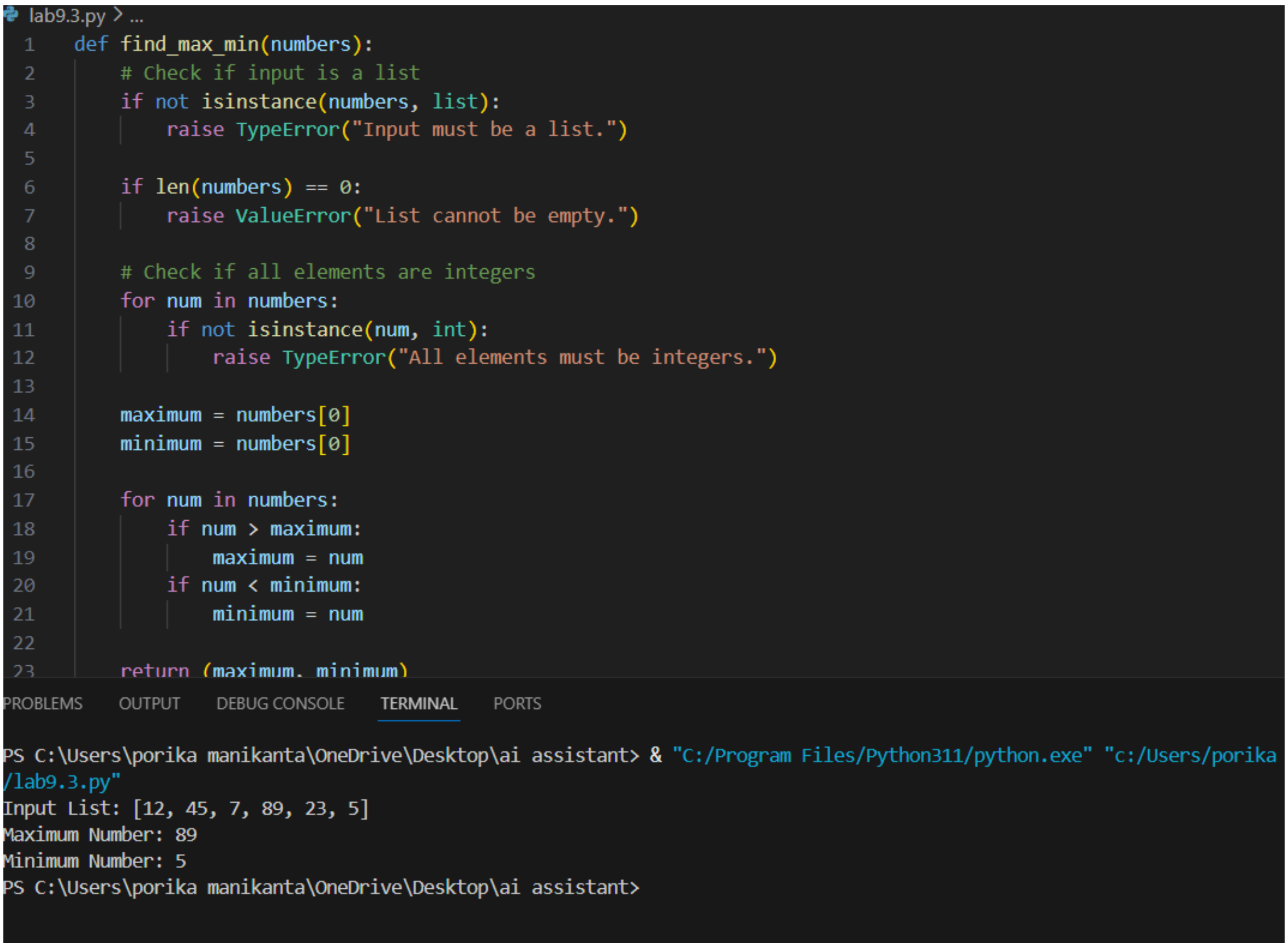
The AI-assisted tool generated a Google-style docstring that clearly describes the function parameters and return values. The function validates input type, separates even and odd numbers using modulo, and returns both sums as a tuple. The AI-generated docstring was more detailed and structured compared to a manually written one, covering edge cases like TypeError for non-integer elements.

Task 2: Automatic Inline Comments

Scenario: You are developing a student management module that must be easy to understand for new developers.

Prompt: # Write a Python function to find maximum and minimum from a list

Code:


lab9.3.py > ...

```python
def find_max_min(numbers):
    # Check if input is a list
    if not isinstance(numbers, list):
        raise TypeError("Input must be a list.")

    if len(numbers) == 0:
        raise ValueError("List cannot be empty.")

    # Check if all elements are integers
    for num in numbers:
        if not isinstance(num, int):
            raise TypeError("All elements must be integers.")

    maximum = numbers[0]
    minimum = numbers[0]

    for num in numbers:
        if num > maximum:
            maximum = num
        if num < minimum:
            minimum = num

    return (maximum, minimum)
```

```
PS C:\Users\porika manikanta\OneDrive\Desktop\ai assistant> & "C:/Program Files/Python311/python.exe" "c:/Users/porika
/lab9.3.py"
Input List: [12, 45, 7, 89, 23, 5]
Maximum Number: 89
Minimum Number: 5
PS C:\Users\porika manikanta\OneDrive\Desktop\ai assistant>
```

Result:

For input [12, 45, 7, 89, 23, 5]: Maximum Number: 89, Minimum Number: 5.

Observation:

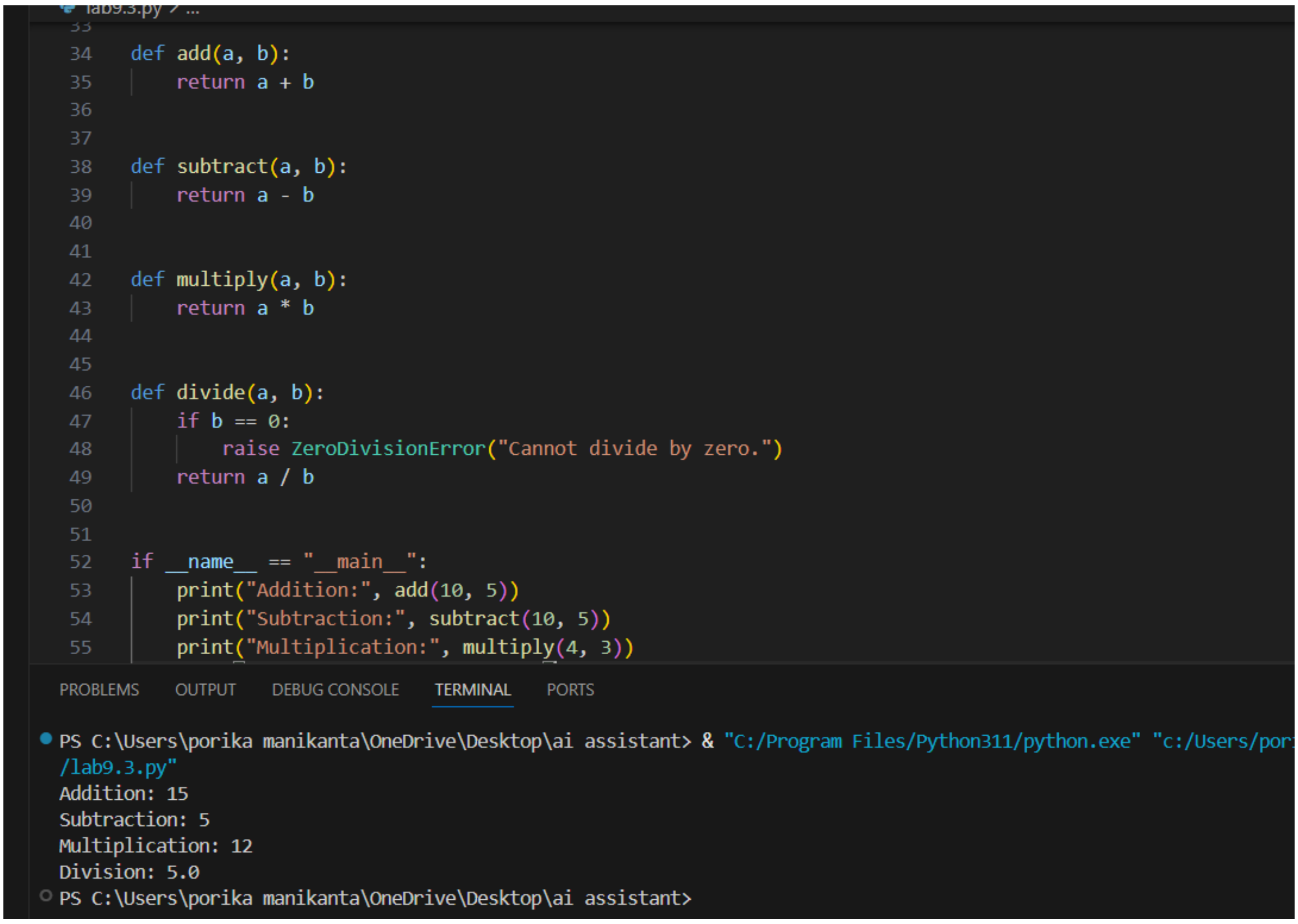
The find_max_min() function was generated with proper inline comments by the AI tool. It validates that the input is a non-empty list of integers, then iterates through to find maximum and minimum values. The AI-generated comments were accurate and described each logical block clearly, making the code easy to understand for new developers.

Task 3: Module-Level and Function-Level Documentation

Scenario: You are building a small calculator module that will be shared across multiple projects and requires structured documentation.

Prompt: # Write Python functions for add, subtract, multiply, and divide with NumPy style docstrings

Code:


lab9.3.py > ...

```python
def add(a, b):
    return a + b


def subtract(a, b):
    return a - b


def multiply(a, b):
    return a * b


def divide(a, b):
    if b == 0:
        raise ZeroDivisionError("Cannot divide by zero.")
    return a / b


if __name__ == "__main__":
    print("Addition:", add(10, 5))
    print("Subtraction:", subtract(10, 5))
    print("Multiplication:", multiply(4, 3))
```

```
PS C:\Users\porika manikanta\OneDrive\Desktop\ai assistant> & "C:/Program Files/Python311/python.exe" "c:/Users/por
/lab9.3.py"
Addition: 15
Subtraction: 5
Multiplication: 12
Division: 5.0
PS C:\Users\porika manikanta\OneDrive\Desktop\ai assistant>
```

Result:

Addition: 15, Subtraction: 5, Multiplication: 12, Division: 5.0

Observation:

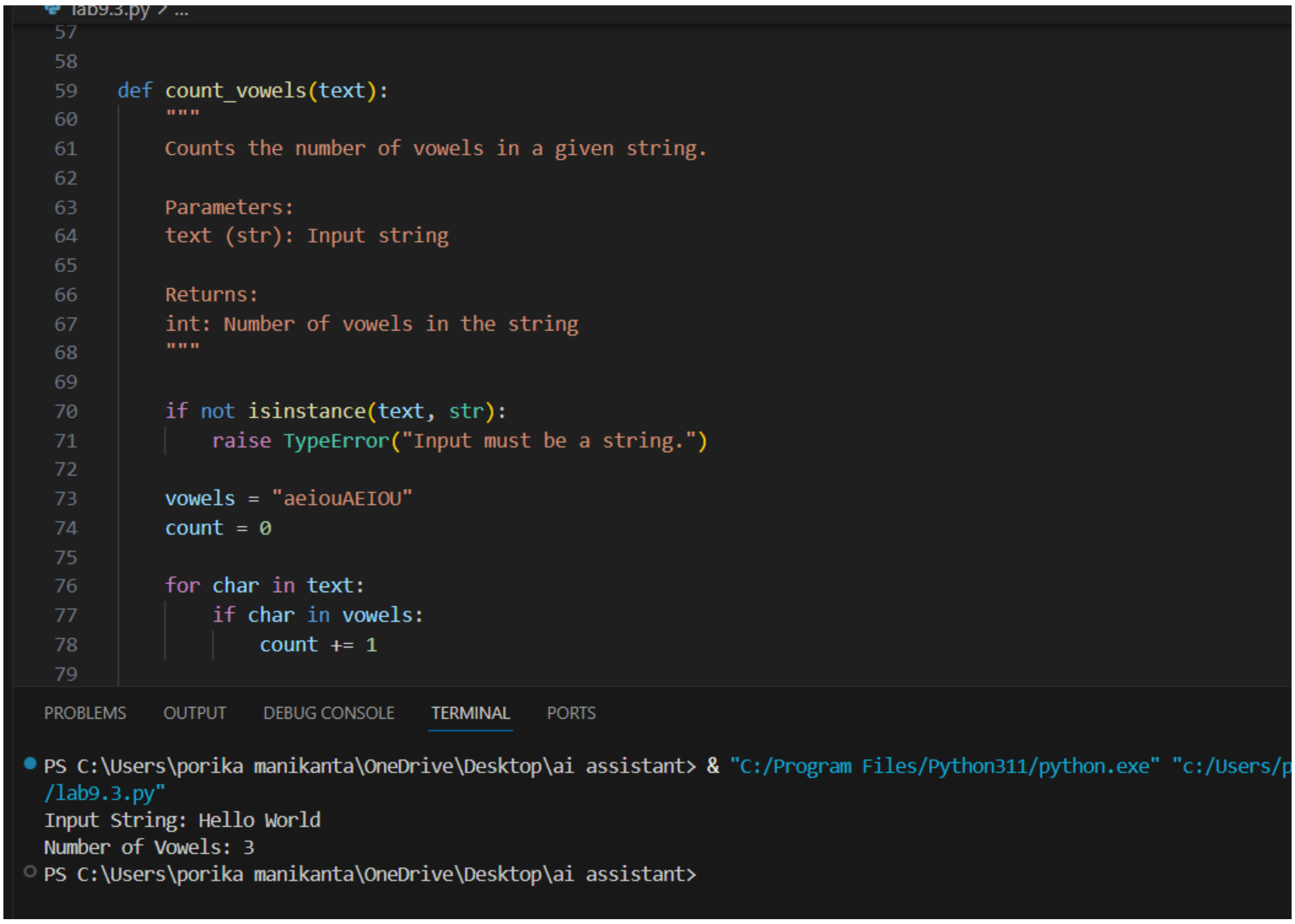
The calculator module includes four functions: add, subtract, multiply, and divide. The divide function includes a ZeroDivisionError guard. AI assistance generated a module-level docstring and NumPy-style function-level docstrings that were more structured and complete than manually written ones. The AI accurately captured parameter types and return value descriptions.

Additional Tasks: count_vowels and reverse_string Functions

Task: count_vowels Function

Prompt: # Write a Python function to count the number of vowels in a given string

Code:



```python

def count_vowels(text):
    """
    Counts the number of vowels in a given string.

    Parameters:
    text (str): Input string

    Returns:
    int: Number of vowels in the string
    """

    if not isinstance(text, str):
        raise TypeError("Input must be a string.")

    vowels = "aeiouAEIOU"
    count = 0

    for char in text:
        if char in vowels:
            count += 1

```

```
PS C:\Users\porika manikanta\OneDrive\Desktop\ai assistant> & "C:/Program Files/Python311/python.exe" "c:/Users/p
/lab9.3.py"
Input String: Hello World
Number of Vowels: 3
PS C:\Users\porika manikanta\OneDrive\Desktop\ai assistant>
```

Result:

Input String: Hello World — Number of Vowels: 3

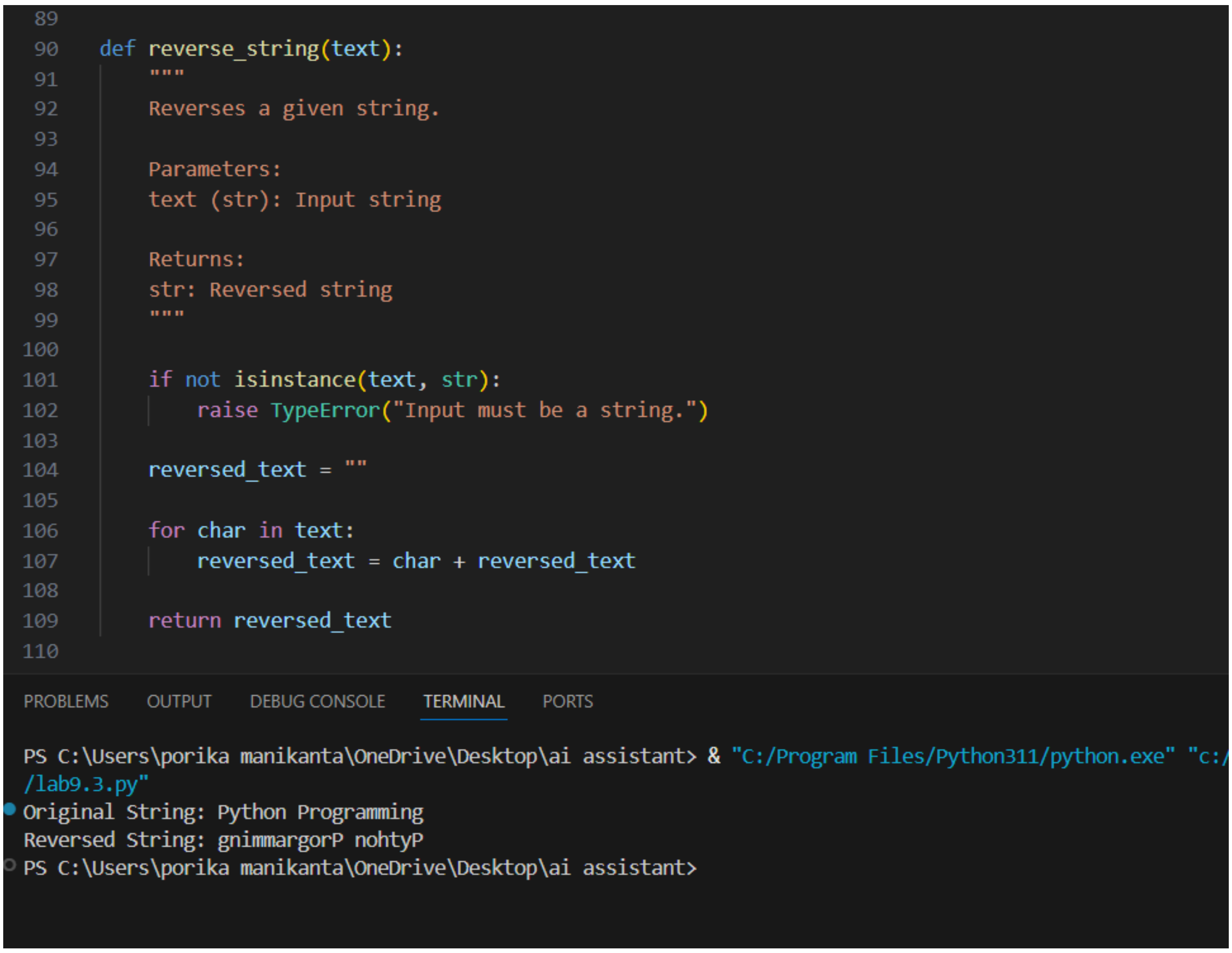
Observation:

The count_vowels() function uses a comprehensive vowel string (both uppercase and lowercase) and counts matching characters in the input. The AI-generated docstring included parameters, return types, and raises section. The inline comments helped trace the logic clearly for any new developer reviewing the code.

Task: reverse_string Function

Prompt: # Write a Python function to reverse a given string

Code:



```python

def reverse_string(text):
    """
    Reverses a given string.

    Parameters:
    text (str): Input string

    Returns:
    str: Reversed string
    """

    if not isinstance(text, str):
        raise TypeError("Input must be a string.")

    reversed_text = ""

    for char in text:
        reversed_text = char + reversed_text

    return reversed_text

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\porika manikanta\OneDrive\Desktop\ai assistant> & "C:/Program Files/Python311/python.exe" "c:/
/lab9.3.py"
Original String: Python Programming
Reversed String: gnimmargorP nohtyP
PS C:\Users\porika manikanta\OneDrive\Desktop\ai assistant>
```

Result:

Original String: Python Programming — Reversed String: gnimmargorP nohtyP

Observation:

The reverse_string() function reverses a string by prepending each character to an initially empty string. The AI-generated docstring correctly captured the function intent, parameter descriptions, and return type. Inline comments were added for each logical step. Compared to manual documentation, the AI-generated version was consistent in style and covered the TypeError guard as well.