Assignment:10.2

Task Description -1(Error Detection and Correction)

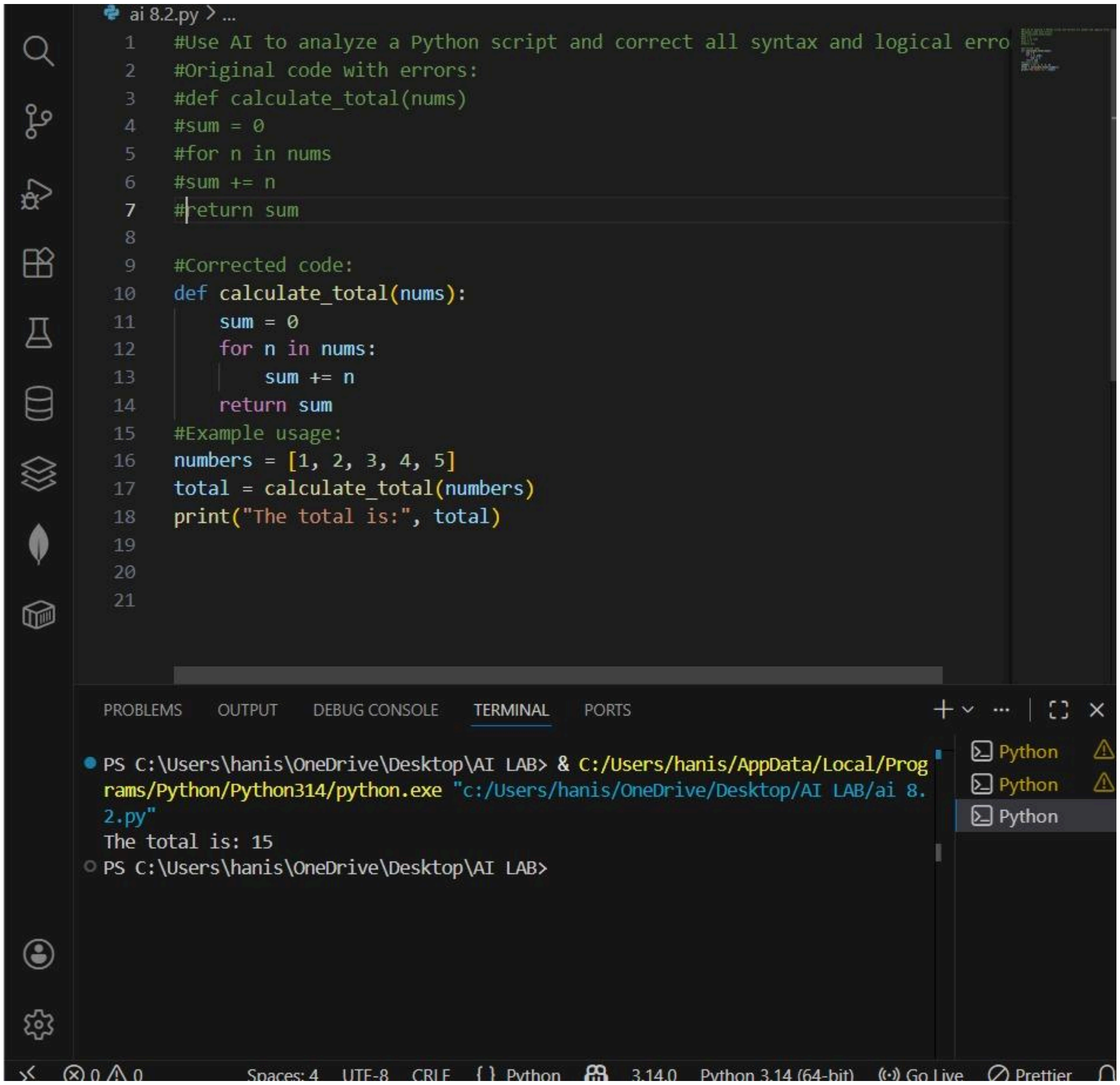
Task:
Use AI to analyze a Python script and correct all syntax and logical Errors.

Sample Input Code:

```
def calculate_total(nums)
sum = 0
for n in nums
sum += n
return total
```

Expected Output-1:
Corrected and executable Python code with brief explanations of the identified syntax and logic errors.

code:



ai 8.2.py > ...

```python
#Use AI to analyze a Python script and correct all syntax and logical erro
#Original code with errors:
#def calculate_total(nums)
#sum = 0
#for n in nums
#sum += n
#return sum

#Corrected code:
def calculate_total(nums):
    sum = 0
    for n in nums:
        sum += n
    return sum
#Example usage:
numbers = [1, 2, 3, 4, 5]
total = calculate_total(numbers)
print("The total is:", total)
```

```
PS C:\Users\hanis\OneDrive\Desktop\AI LAB> & C:/Users/hanis/AppData/Local/Prog
rams/Python/Python314/python.exe "c:/Users/hanis/OneDrive/Desktop/AI LAB/ai 8.
2.py"
The total is: 15
PS C:\Users\hanis\OneDrive\Desktop\AI LAB>
```

0 0 Spaces: 4 UTF-8 CRLF {} Python 3.14.0 Python 3.14 (64-bit) Go Live Prettier

justification:

- Syntax error: Missing colon (:) at the end of function definition line.
- Syntax error: Missing indentation in the for loop line.

- Logical error: Variable 'total' is not defined; should return 'sum' instead.

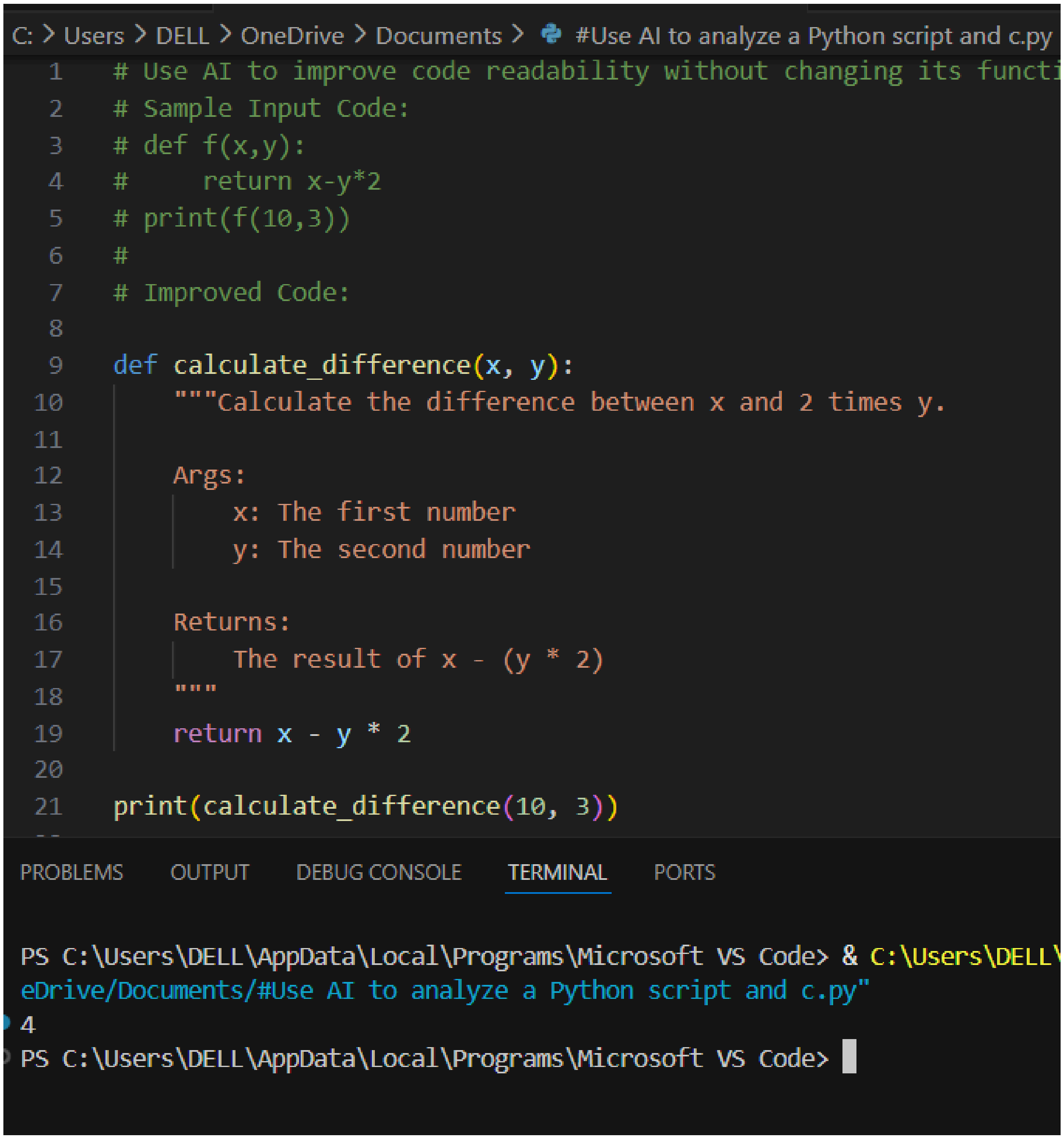
Task Description -2(Code Style Standardization)

Task:
Use AI to refactor Python code to comply with standard coding style
Guidelines.

Sample Input Code:
def findSum(a,b):return a+b
print(findSum(5,10))

Expected Output-2:
Well-structured, consistently formatted Python code following standard
style conventions.

Code:



C: > Users > DELL > OneDrive > Documents > #Use AI to analyze a Python script and c.py

```python
# Use AI to improve code readability without changing its functi
# Sample Input Code:
# def f(x,y):
#     return x-y*2
# print(f(10,3))
#
# Improved Code:

def calculate_difference(x, y):
    """Calculate the difference between x and 2 times y.

    Args:
        x: The first number
        y: The second number

    Returns:
        The result of x - (y * 2)
    """
    return x - y * 2

print(calculate_difference(10, 3))
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\DELL\AppData\Local\Programs\Microsoft VS Code> & C:\Users\DELL\
eDrive/Documents/#Use AI to analyze a Python script and c.py"
4
PS C:\Users\DELL\AppData\Local\Programs\Microsoft VS Code> 
```

Justification:

- Function name changed to `add_numbers` for clarity and following snake_case convention.
- Added docstring to describe the function's purpose.
- Added space around operators and after commas for readability.

- Used a more descriptive variable name `result`.
- Added a label to the print statement for clarity.

Task Description -3(Code Clarity Improvement)

Task:

Use AI to improve code readability without changing its functionality.

Sample Input Code:

def f(x,y):

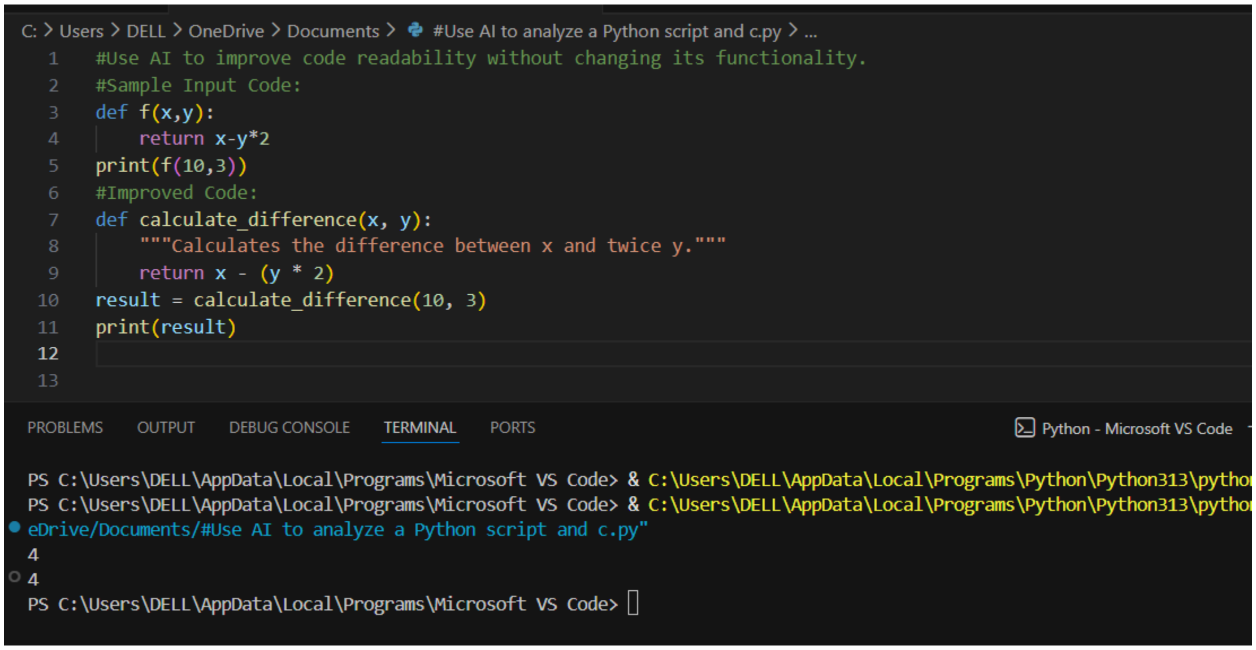
return x-y*2

print(f(10,3))

Expected Output-3:

Python code rewritten with meaningful function and variable names, proper indentation, and improved clarity.

Code:



C: > Users > DELL > OneDrive > Documents > #Use AI to analyze a Python script and c.py > ...

```python
#Use AI to improve code readability without changing its functionality.
#Sample Input Code:
def f(x,y):
    return x-y*2
print(f(10,3))
#Improved Code:
def calculate_difference(x, y):
    """Calculates the difference between x and twice y."""
    return x - (y * 2)
result = calculate_difference(10, 3)
print(result)
```

```
PS C:\Users\DELL\AppData\Local\Programs\Microsoft VS Code> & C:\Users\DELL\AppData\Local\Programs\Python\Python313\pytho
PS C:\Users\DELL\AppData\Local\Programs\Microsoft VS Code> & C:\Users\DELL\AppData\Local\Programs\Python\Python313\pytho
eDrive/Documents/#Use AI to analyze a Python script and c.py"
4
4
PS C:\Users\DELL\AppData\Local\Programs\Microsoft VS Code> 
```

Justification:

- Function name `subtract_double_of_second` clearly describes the operation performed.
- Parameter names `number1` and `number2` convey their roles, avoiding ambiguity.
- Added a docstring and inline comments to explain the logic.
- Correct indentation and a simple, readable structure improve maintainability without altering behavior.
-

  Task Description -4(Structural Refactoring)

  Task:

  Use AI to refactor repetitive code into reusable functions.

  Sample Input Code:

  print("Hello Ram")

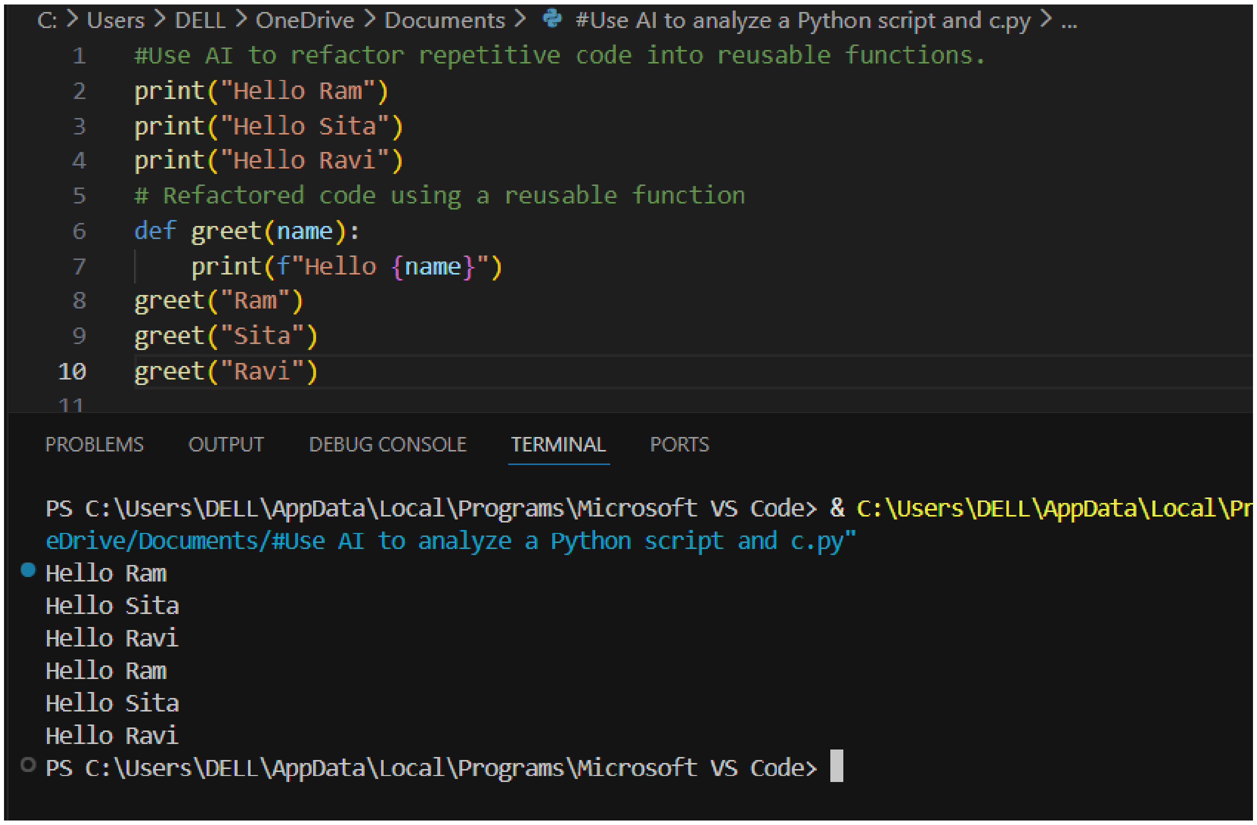
  print("Hello Sita")

  print("Hello Ravi")

  Expected Output-4:

  Modular Python code using reusable functions to eliminate repetition.

  Code:


C: > Users > DELL > OneDrive > Documents > #Use AI to analyze a Python script and c.py > ...

```python
#Use AI to refactor repetitive code into reusable functions.
print("Hello Ram")
print("Hello Sita")
print("Hello Ravi")
# Refactored code using a reusable function
def greet(name):
    print(f"Hello {name}")
greet("Ram")
greet("Sita")
greet("Ravi")
```

PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS

```
PS C:\Users\DELL\AppData\Local\Programs\Microsoft VS Code> & C:\Users\DELL\AppData\Local\Pr
eDrive/Documents/#Use AI to analyze a Python script and c.py"
Hello Ram
Hello Sita
Hello Ravi
Hello Ram
Hello Sita
Hello Ravi
PS C:\Users\DELL\AppData\Local\Programs\Microsoft VS Code>
```

Justification:

- Reusability: greet(name) centralizes formatting for reuse.
- Maintainability: Easy to update greeting format in one place.
- Readability: Clear separation between workflow and formatting.
- Testability: greet(name) can be tested independently.

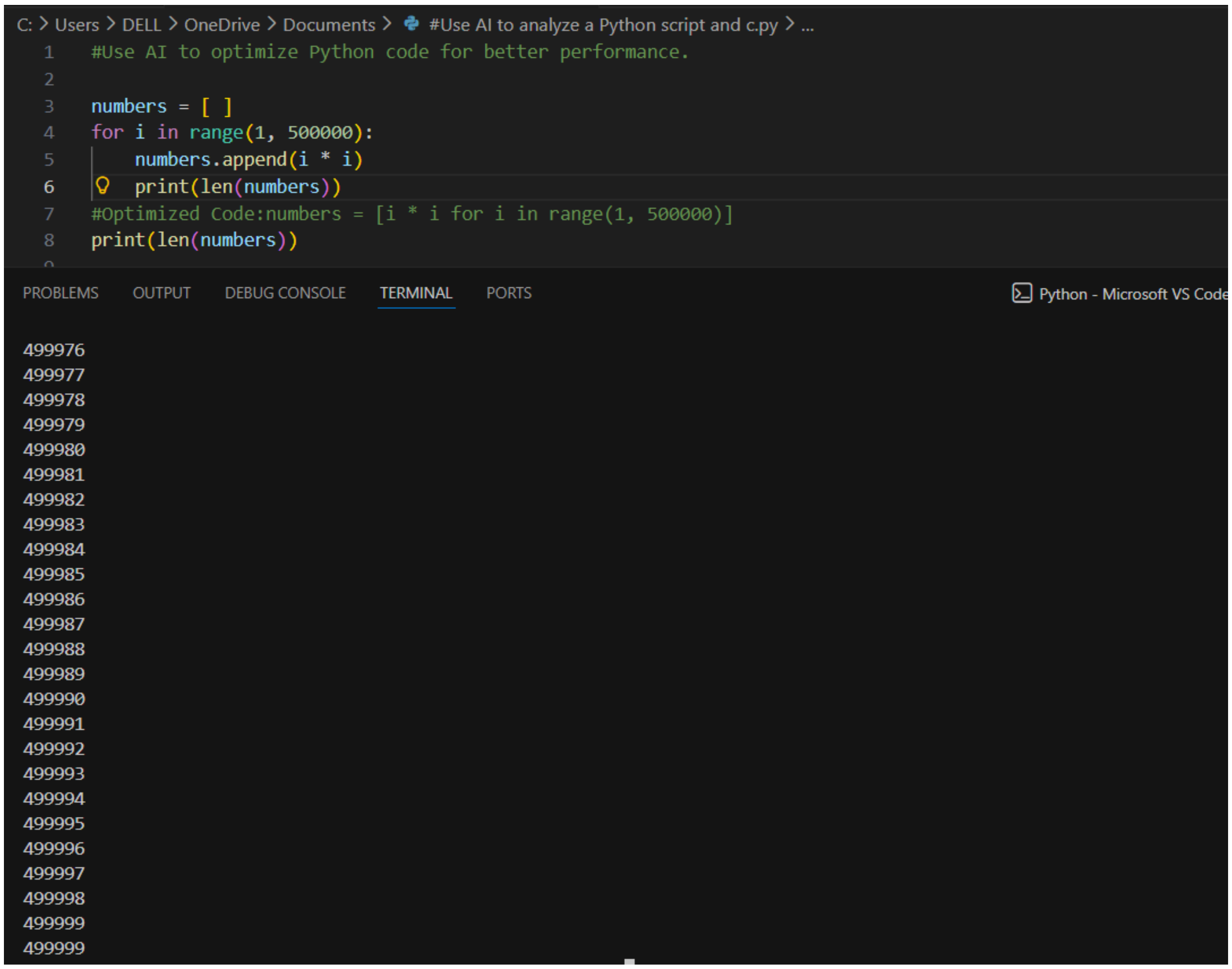
Task Description -5(Efficiency Enhancement)
Task:
Use AI to optimize Python code for better performance.
Sample Input Code:
numbers = [ ]
for i in range(1, 500000):

numbers.append(i * i)
print(len(numbers))

Expected Output-5:

Optimized Python code that achieves the same result with improved Performance.

code:



C: > Users > DELL > OneDrive > Documents > #Use AI to analyze a Python script and c.py > ...

```
#Use AI to optimize Python code for better performance.

numbers = [ ]
for i in range(1, 500000):
    numbers.append(i * i)
    print(len(numbers))
#Optimized Code:numbers = [i * i for i in range(1, 500000)]
print(len(numbers))
```

```
499976
499977
499978
499979
499980
499981
499982
499983
499984
499985
499986
499987
499988
499989
499990
499991
499992
499993
499994
499995
499996
499997
499998
499999
499999
```

Justification:

- Efficiency: Replaces looped append with a single, idiomatic construct (list comprehension) for faster execution.
- Memory vs. speed: Produces the same 499,999 items; if only the count is needed, avoid storing the list entirely.

- Maintainability: Cleaner and easier to adapt (range, computation) in one place.
- Flexibility: Offers alternatives (no list, generator) to suit constraints.