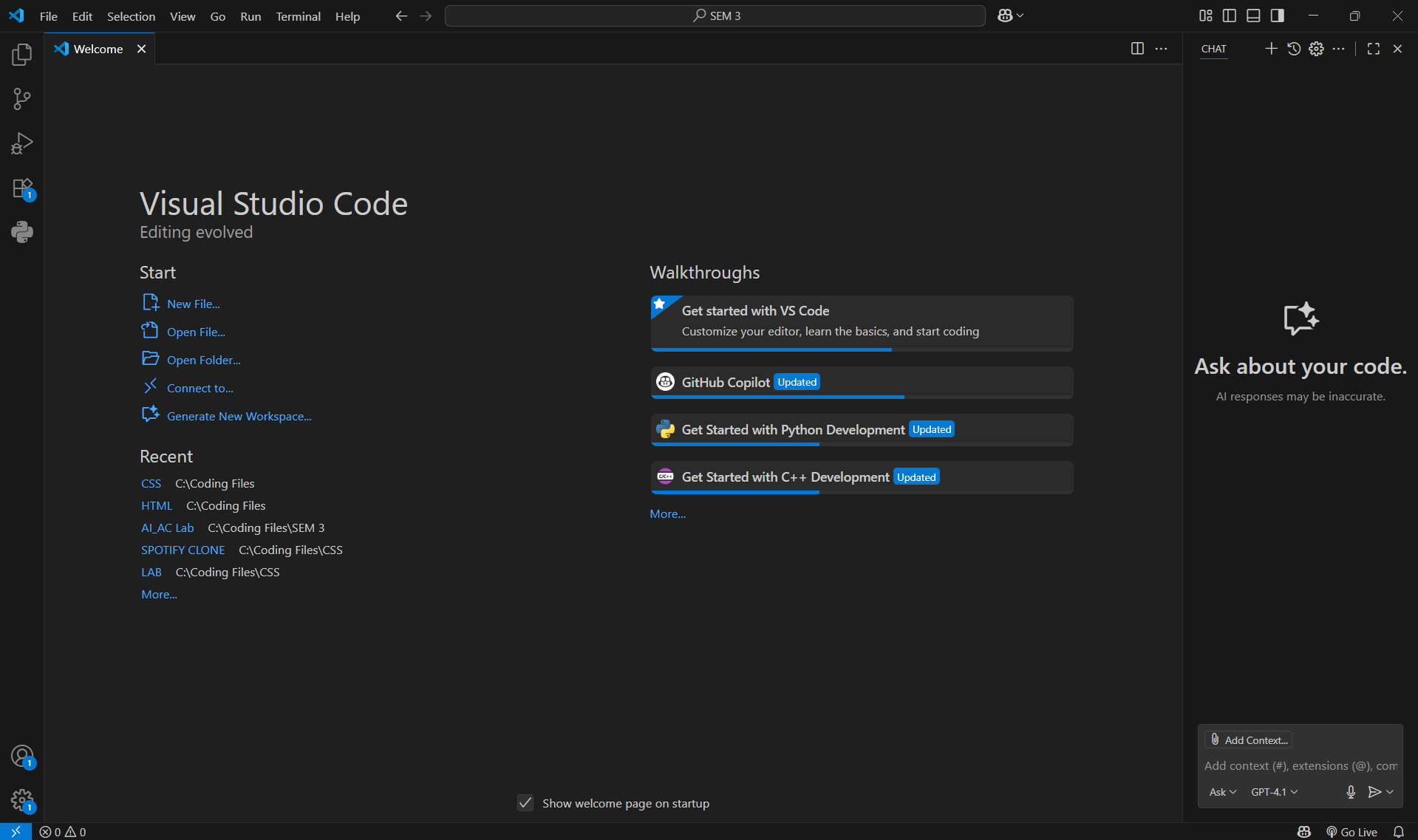
# AI Assisted Coding Lab Assignment – 1.1

2403A510E4- BATCH 05

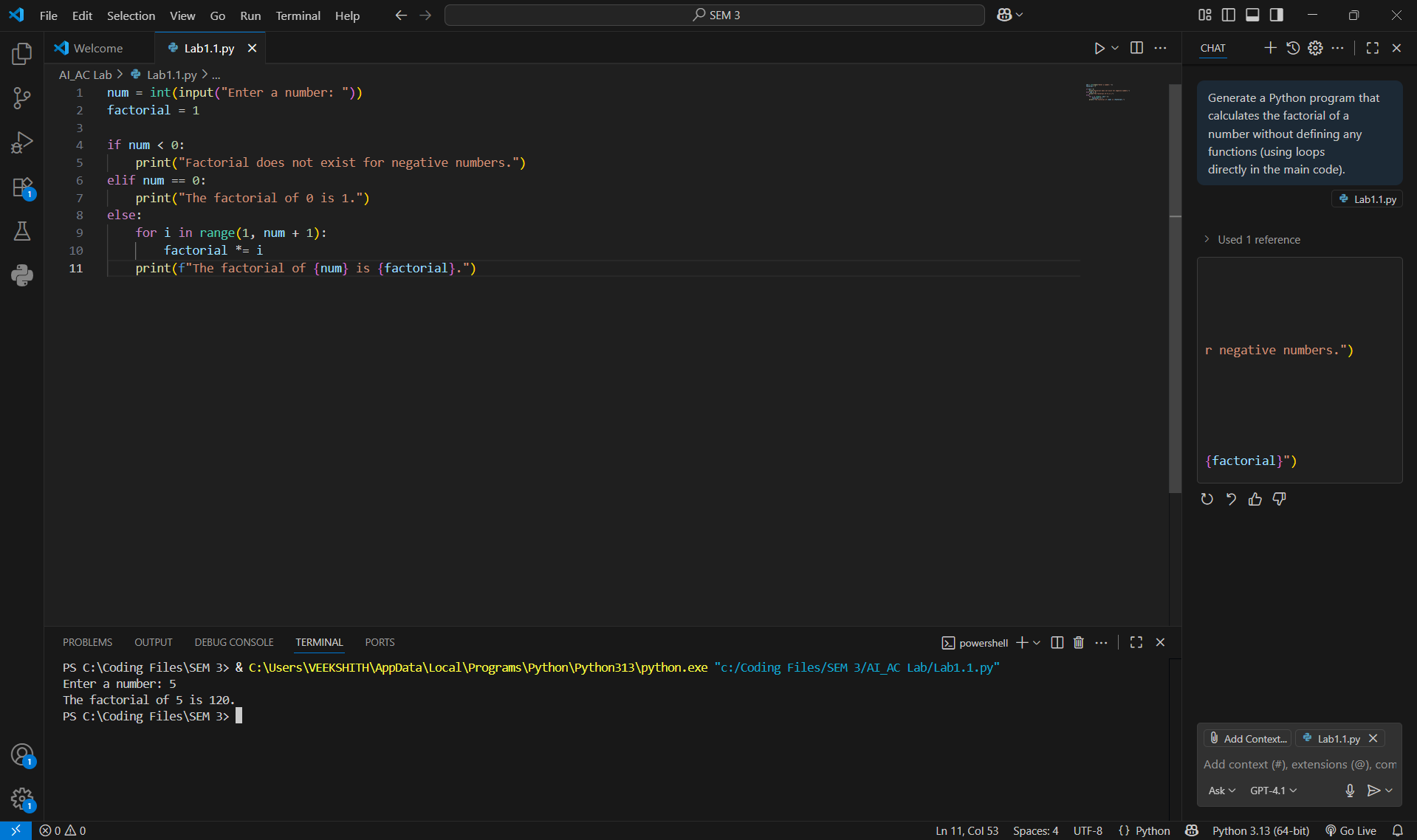
**#TASK 0:**

Install and configure GitHub Copilot in VS Code. Take screenshots of  
each step.



**#TASK 1:**

Use GitHub Copilot to generate a Python program that calculates the  
factorial of a number without defining any functions (using loops directly in the main code).



**#Task 2:**

Examine the Copilot-generated code from Task 1 and demonstrate how its efficiency can be improved (e.g., removing unnecessary variables, optimizing loops).

Co-pilot version:

num = int(input("Enter a number: "))

factorial = 1

if num < 0:

    print("Factorial does not exist for negative numbers.")

elif num == 0:

    print("The factorial of 0 is 1.")

else:

    for i in range(1, num + 1):

        factorial \*= i

    print(f"The factorial of {num} is {factorial}.")

Improved version:

num = int(input("Enter a number: "))

if num < 0:

print("Factorial does not exist for negative numbers.")

else:

factorial = 1

for i in range(2, num + 1): # start from 2 instead of 1

factorial \*= i

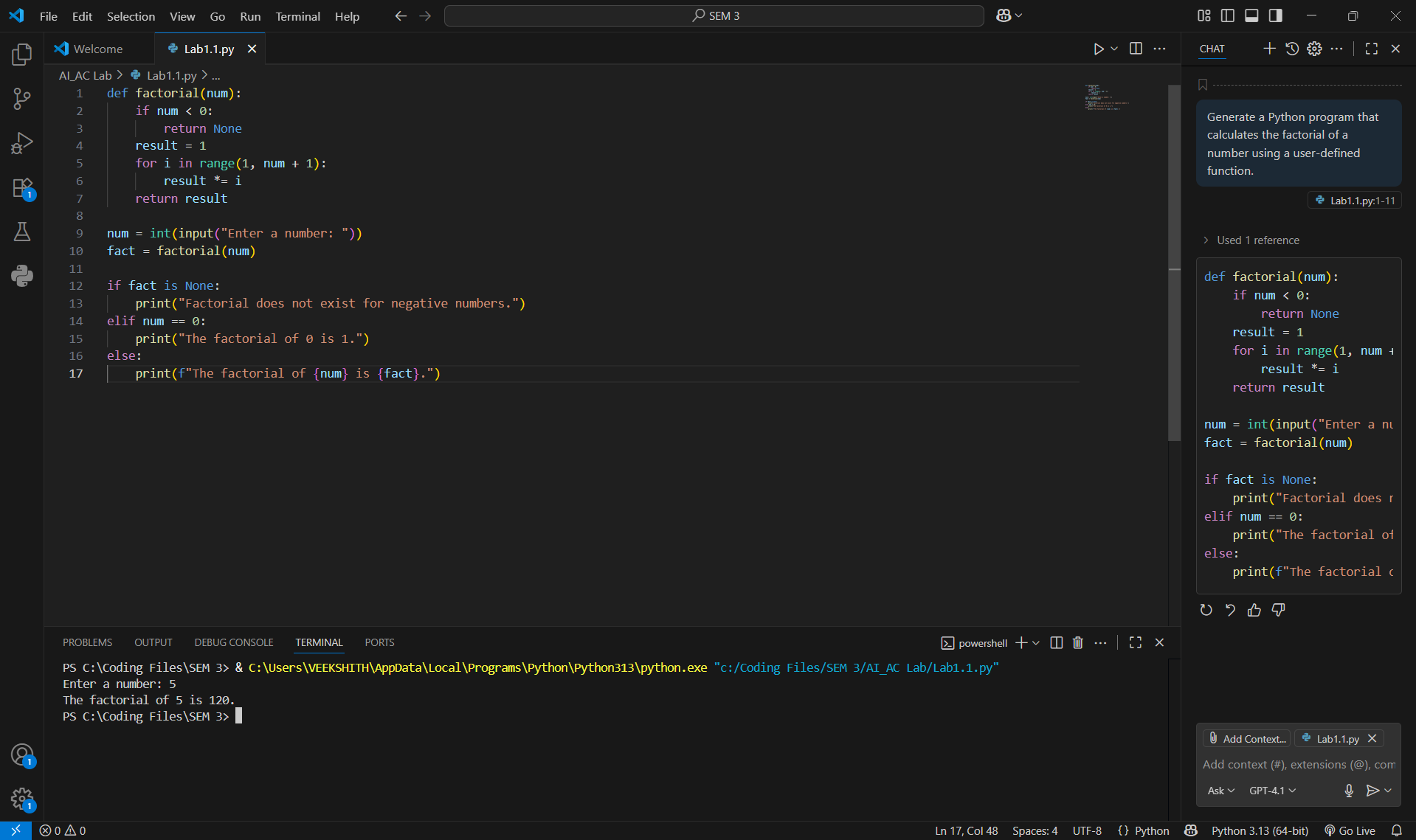
print(f"The factorial of {num} is {factorial}.")

Explanation:

The improved version skips unnecessary multiplication by 1 and also removed the redundant elif statement.

**#TASK 3:**

Use GitHub Copilot to generate a Python program that calculates the factorial of a number using a user-defined function



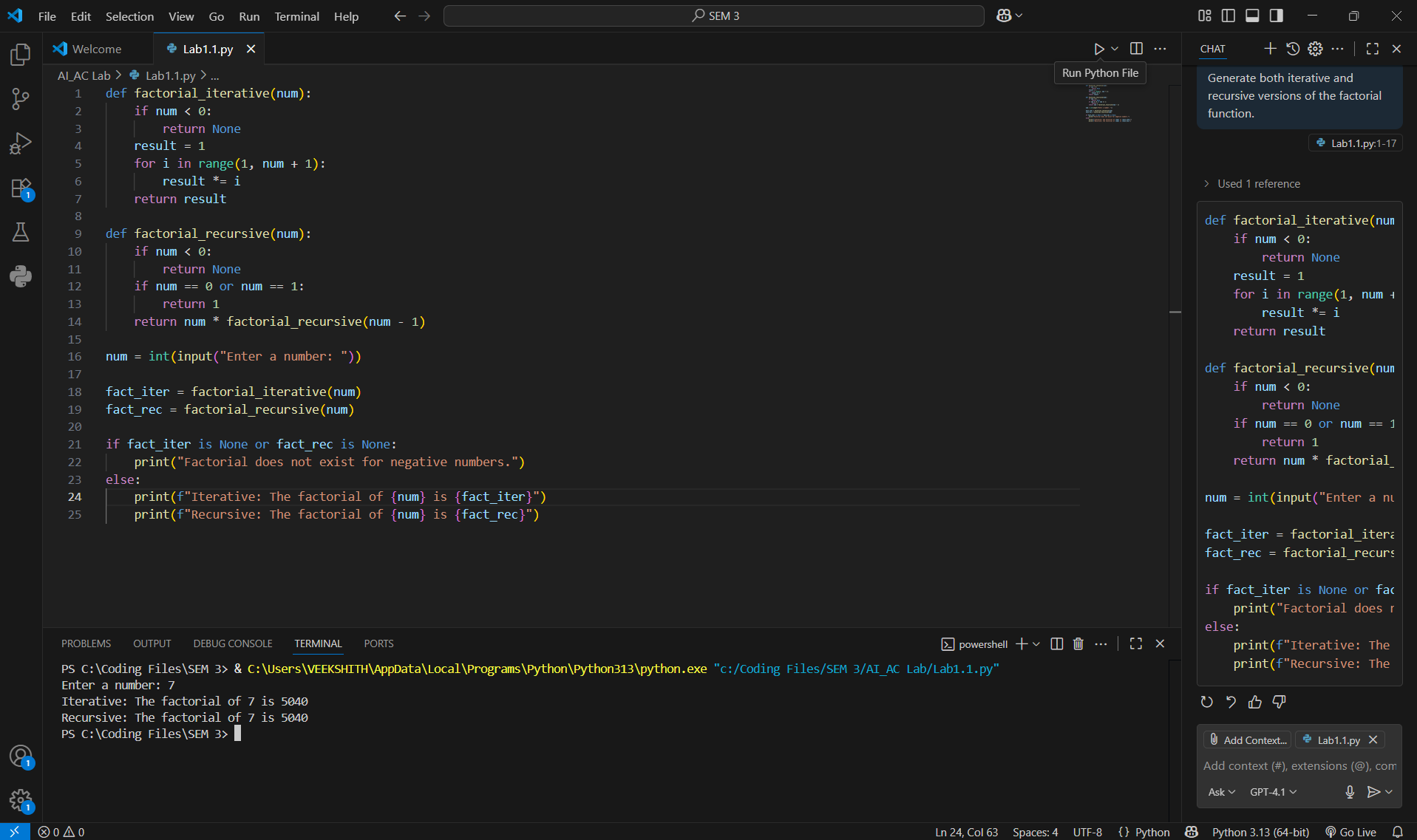
**#TASK 4:**

Differentiate between the Copilot-generated factorial program with functions and without functions in terms of logic, reusability, and execution.

Short difference table:

| **Aspect** | **Without Function (first code)** | **With Function (second code)** |
| --- | --- | --- |
| **Logic** | Factorial logic is written inline inside the main program. | Factorial logic is wrapped inside a separate factorial() function. |
| **Reusability** | Cannot be reused — you’d need to rewrite the loop if factorial is needed elsewhere. | Reusable — you can call factorial(num) multiple times in different parts of the program. |
| **Execution** | Slightly faster because no function call overhead. | Slightly slower due to function call, but negligible for normal inputs. |
| **Readability** | Simple and direct, good for very small scripts. | More structured, cleaner, and easier to maintain. |
| **Error Handling** | Negative check and logic all mixed together. | Negative check is handled inside the function, separating concerns. |

**#TASK 5:**

Prompt GitHub Copilot to generate both iterative and recursive versions of the factorial function. 

Comparison table:

| **Aspect** | **Iterative Approach** | **Recursive Approach** |
| --- | --- | --- |
| **Logic** | Uses a for loop to multiply numbers from 1 to n. | Function calls itself with n-1 until base case (0 or 1). |
| **Performance** | Faster, less overhead (just a loop). | Slower due to repeated function calls and stack usage. |
| **Execution Flow** | Straightforward: single loop runs n times. | Function keeps calling itself until base case, then unwinds (stack unwinding). |
| **Memory Usage** | Constant (O(1) space). | Requires call stack of depth n → O(n) space. |
| **Readability** | Easy to follow, especially for beginners. | Elegant and mathematical, but harder to trace for large n. |
| **Limitations** | None for reasonable n. | Risk of **RecursionError** if n is too large (Python’s recursion depth limit). |