**LAB ASSIGNMENT – 01**

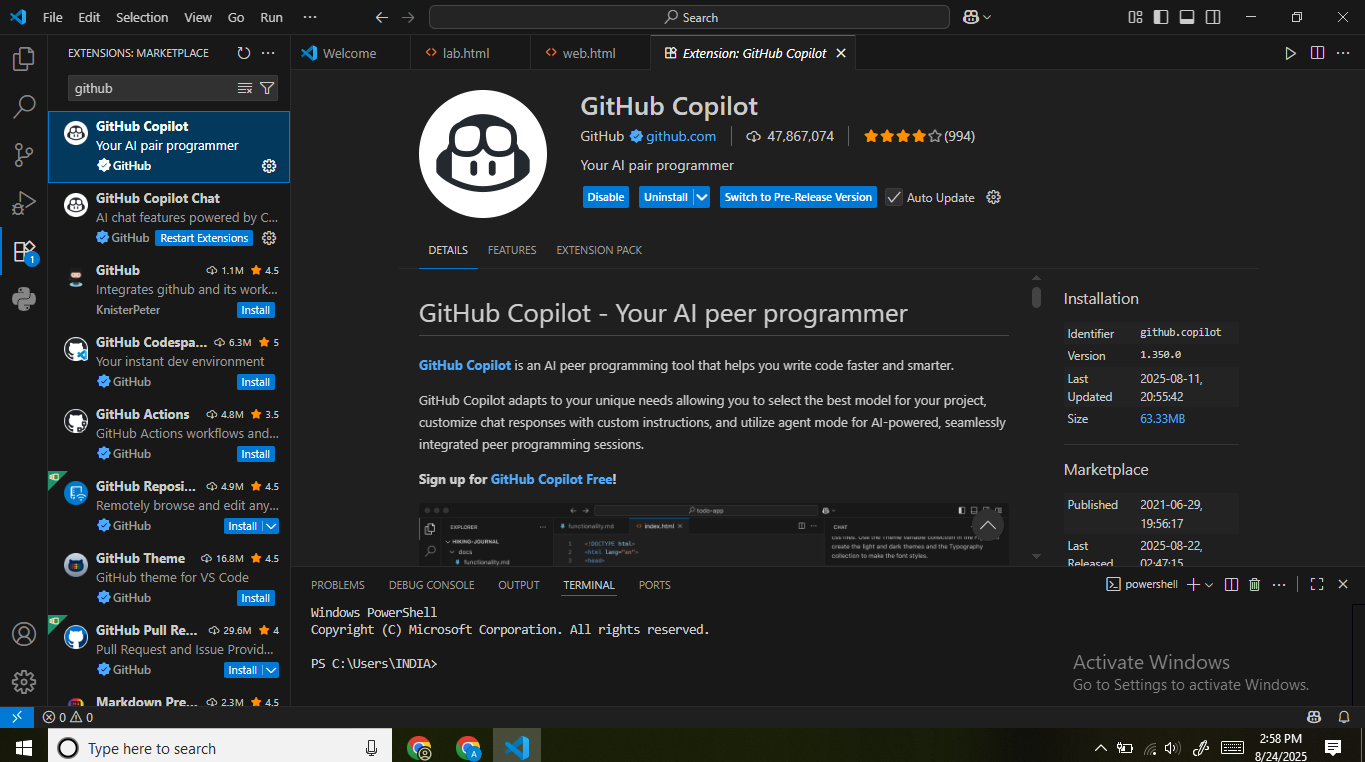
NAME: Bhavaneesh.A

BATCH NO: AI 13

ROLL NO: 2403A52350

Task – 0:

Install and configure GitHub Copilot in VS Code



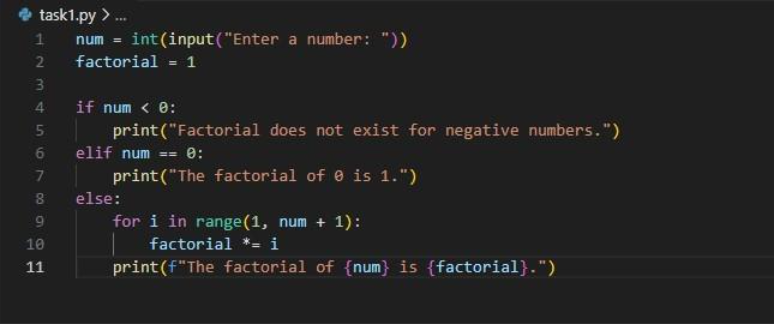
**Task – 1:**

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).

**Prompt - 1:**

Generate a python Program that calculates the factorial of a number without defining any functions (using loops directly in the main code)

**Code:**



**Output:**



**Explanation:**

# **Factorial calculation without functions (using loop directly)**

# **Step 1: Take input from the user**

n = int(input("Enter a number: ")) # Example: if user enters 5, n = 5

# **Step 2: Initialize factorial as 1 (since multiplying by 0 makes everything 0)**

factorial = 1

# **Step 3: Use a loop to multiply numbers from 1 to n**

for i in range(1, n + 1): # Loop runs from 1 up to n (inclusive) factorial \*= i # Multiply factorial by i (factorial = factorial \* i)

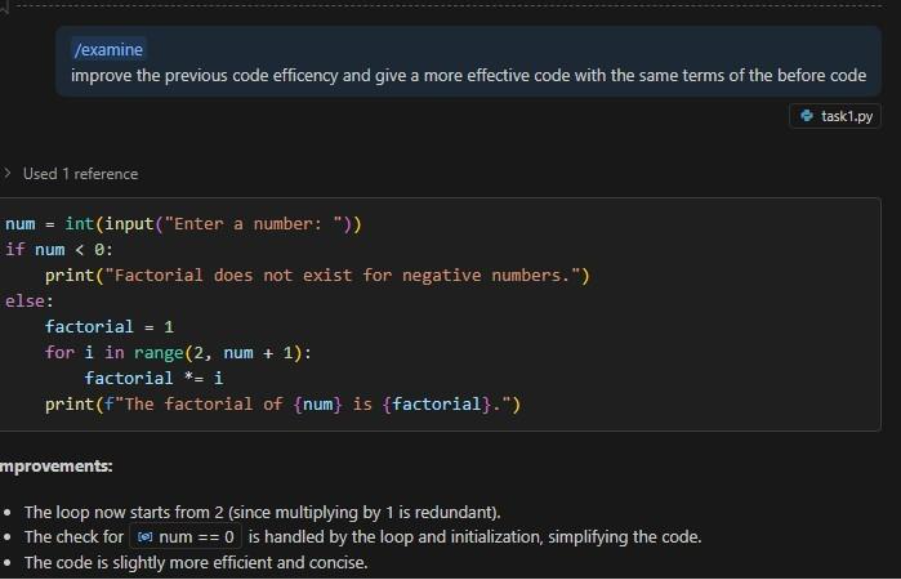
# **Step 4: Print the final result**

print("Factorial of", n, "is:", factorial)

**Task – 2:**

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)

**Code:**



**Output:**



**Differences:**

* The loop now starts from 2 (since multiplying by 1 is redundant).
* The check for [num == 0](https://vscode-file//vscode-app/c:/Users/jithendra%20maddisetty/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html) is handled by the loop and initialization, simplifying the code.
* The code is slightly more efficient and concise.

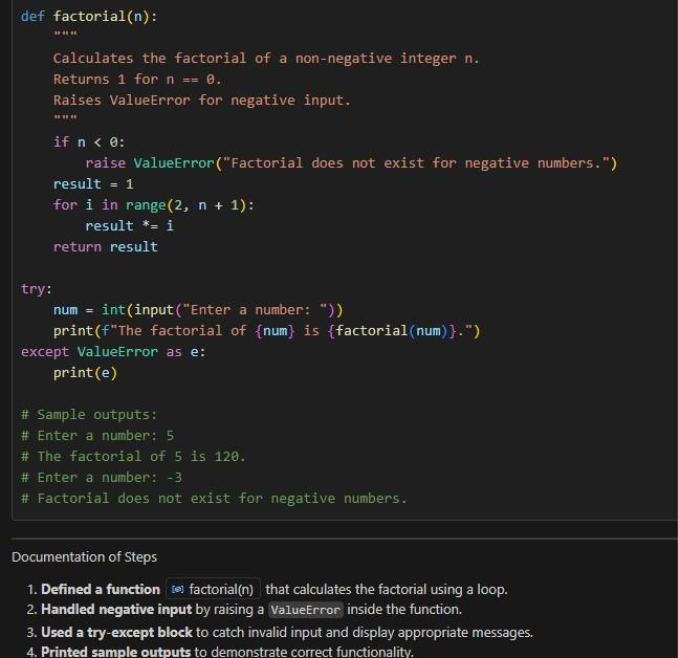
**Task – 3:**

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

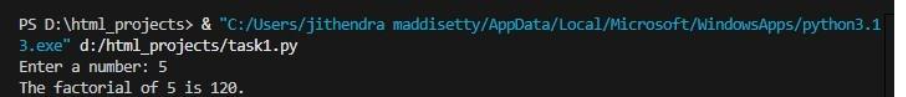
**Prompt – 3:**

Generate a Python program that calculates the factorial of a number using a user-defined function.

**Code:**



**Output:**



**Explanation:**

Defined a function factorial(n) that calculates the factorial using a loop. Handled negative input by raising a ValueError inside the function.

Used a try-except block to catch invalid input and display appropriate messages. Printed sample outputs to demonstrate correct functionalitites

**Task – 4:**

Comparative Analysis – With vs Without Functions

|  |  |  |
| --- | --- | --- |
| Aspect | Without  Functions | With  Functions |
| **Logic** | Logic is written directly in the main script; control flow is sequential | Logic is encapsulated in a separate function (factorial(n)) and invoked when needed. |
| **Reusability** | Cannot be reused without copying the code into another program. | Code is reusable—call  factorial(n) from anywhere in the same or another program |
| **Readability s Structure** | All logic is in one block; larger programs become harder to follow. | Code is modular—core logic is separate from input/output, improving clarity. |
| **Error Handling** | Minimal error handling, usually tied directly to input validation | Robust error handling can be built into the function (e.g., raise ValueError) for better  control. |
| **Execution Flow** | Program executes line-by-line from input to output without modular jumps. | Main section handles input/output, calls the  function, receives the result,  and displays it. |
| **Maintainability** | Updating logic requires editing it everywhere it appears | Updating logic in one function automatically updates all uses across the program |
| **Testing** | Logic is hard to test separately from user input/output. | Function can be tested independently without  involving user interaction |

**Task – 5:**

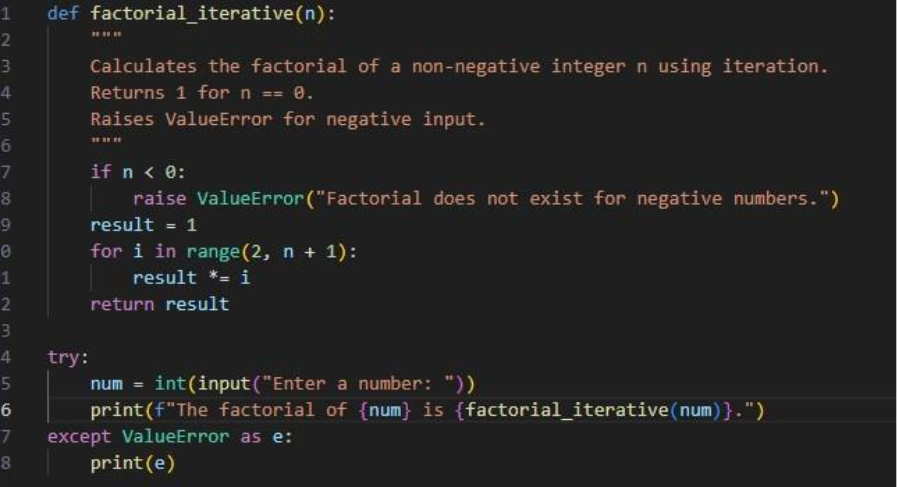
Iterative vs Recursive Factorial

**Prompt – 5:**

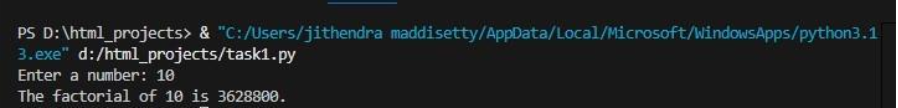
Please generate the iterative vs Recursive versions of the code separately with maximum care SX.

**Iterative:**

**Code:**

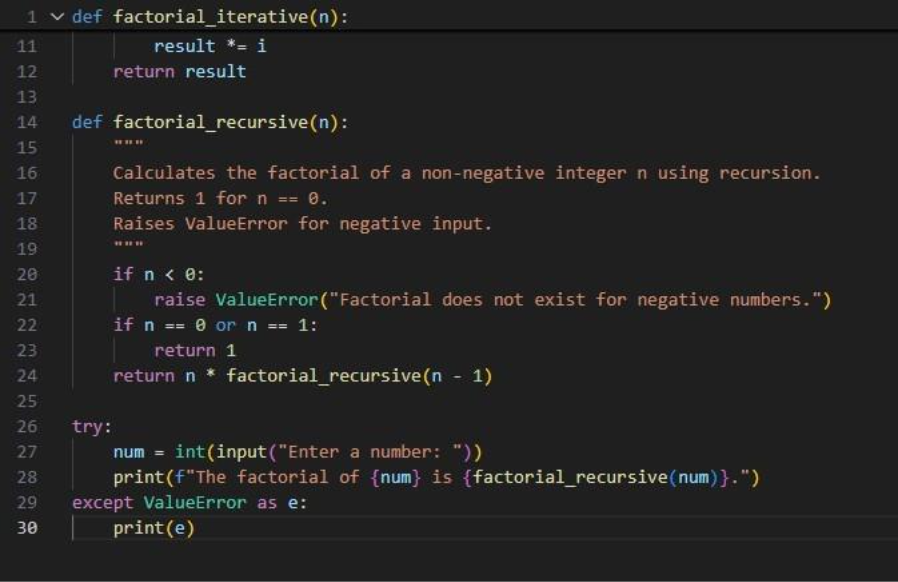


**Output:**

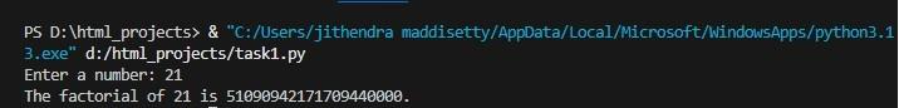


**Recursive:**

**Code:**



**Output:**



**Explanation and Comparision:**

|  |  |  |
| --- | --- | --- |
| Feature | Iterative | Recursive |
| **logic** | Uses a loop to multiply numbers from 2 to n | Calls itself with n-1 until base case reached |
| **Time Complexity** | O(n) | O(n) |
| **Space Complexity** | O(1) | O(n) (due to call stack) |
| **Memory Usage** | Low | Higher |
| **Risk** | No stack overflow | Risk of RecursionError for large n |
| **Readability** | Less mathematical, but straightforward | Matches mathematical definition, more elegant |
| **Performance** | Faster for large n | Slower due to function call overhead |