hall ticket: 2403A52385

name: Charala Rohith Sai

batch: 24BTCAICSB18

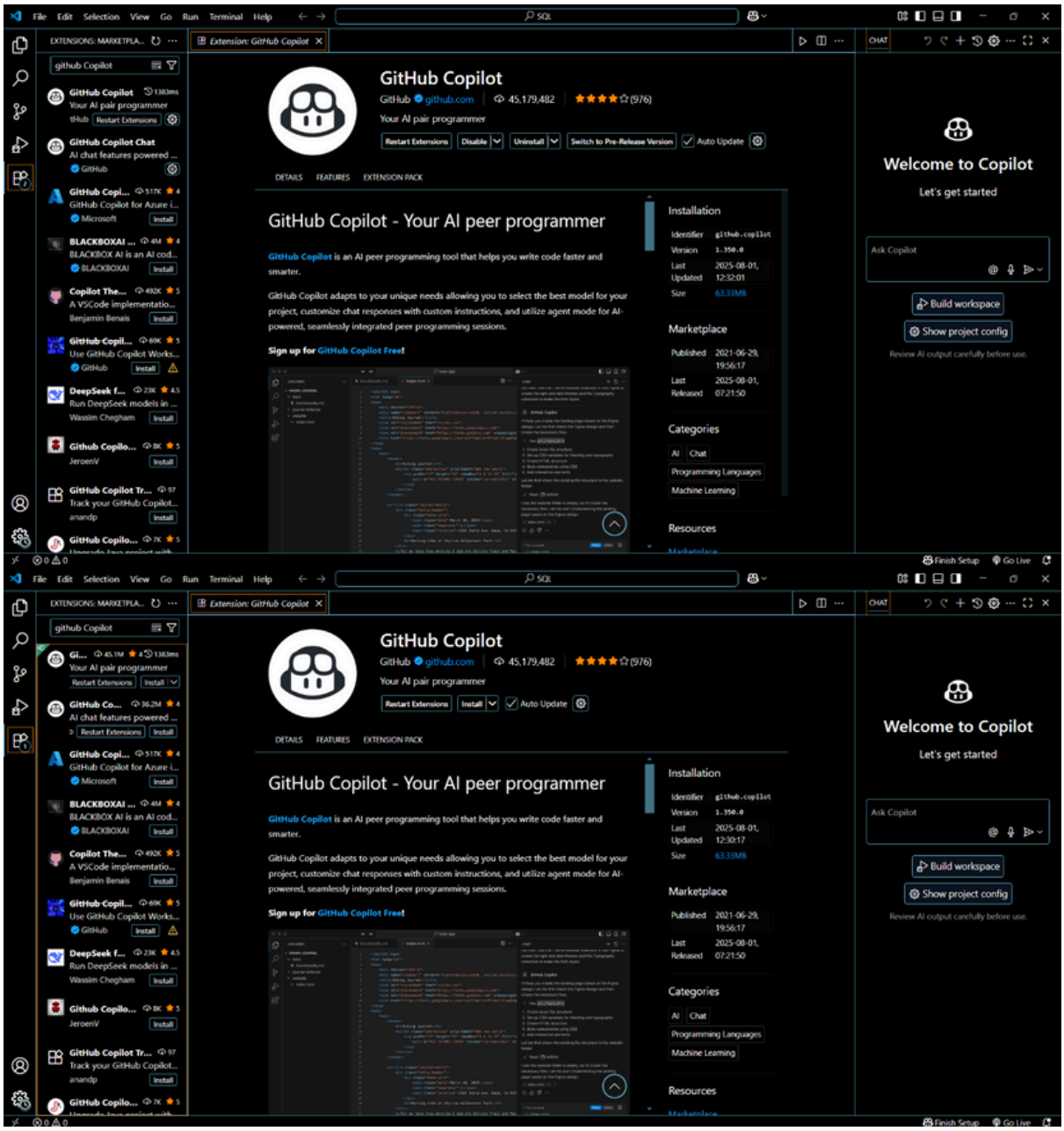
Lab 1: Environment Setup – GitHub Copilot and VS Code Integration

Task 1: Installation and configuration of GitHub Copilot in Vs Code.

Installation of GitHub Copilot in Vs Code.

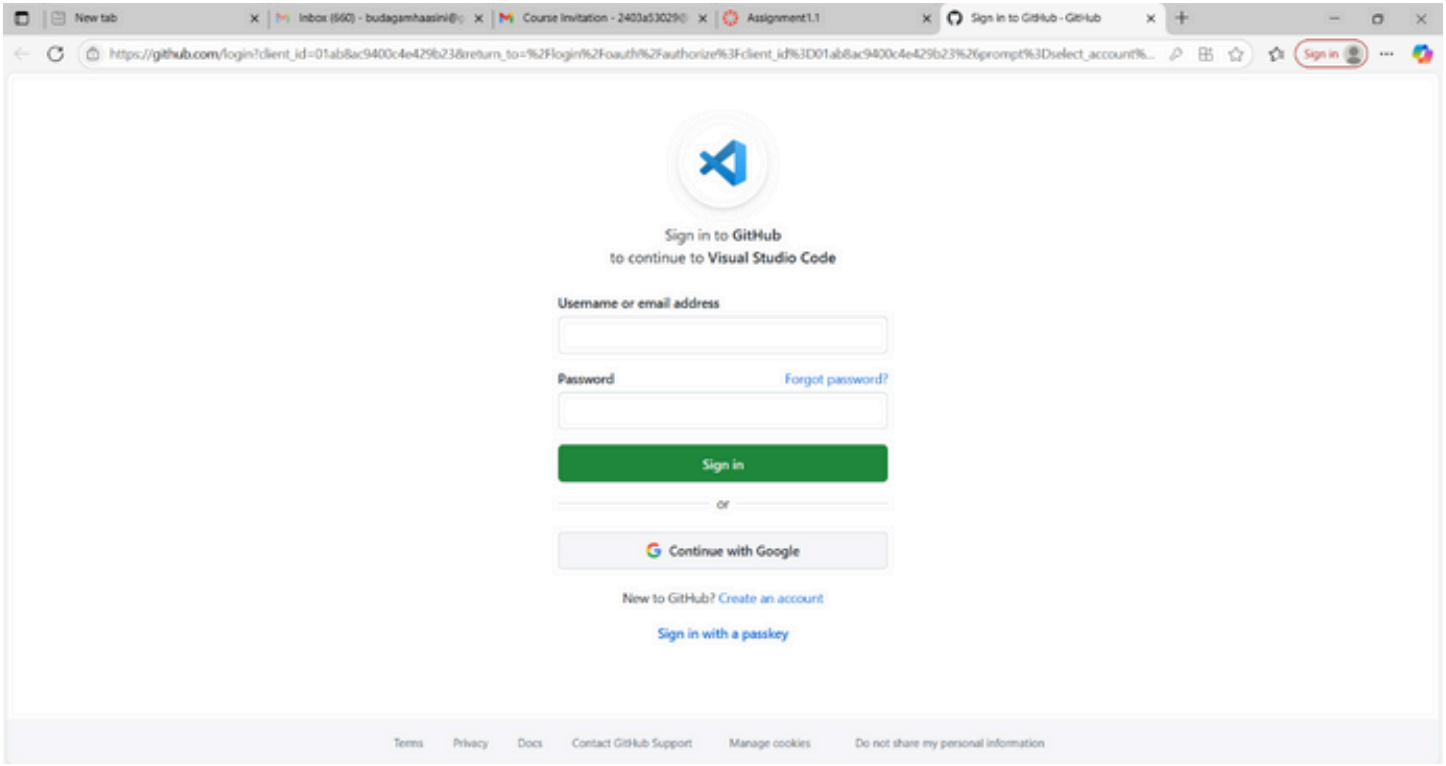
Here, we have to search for the GitHub Copilot extension in Vs Code and install it.

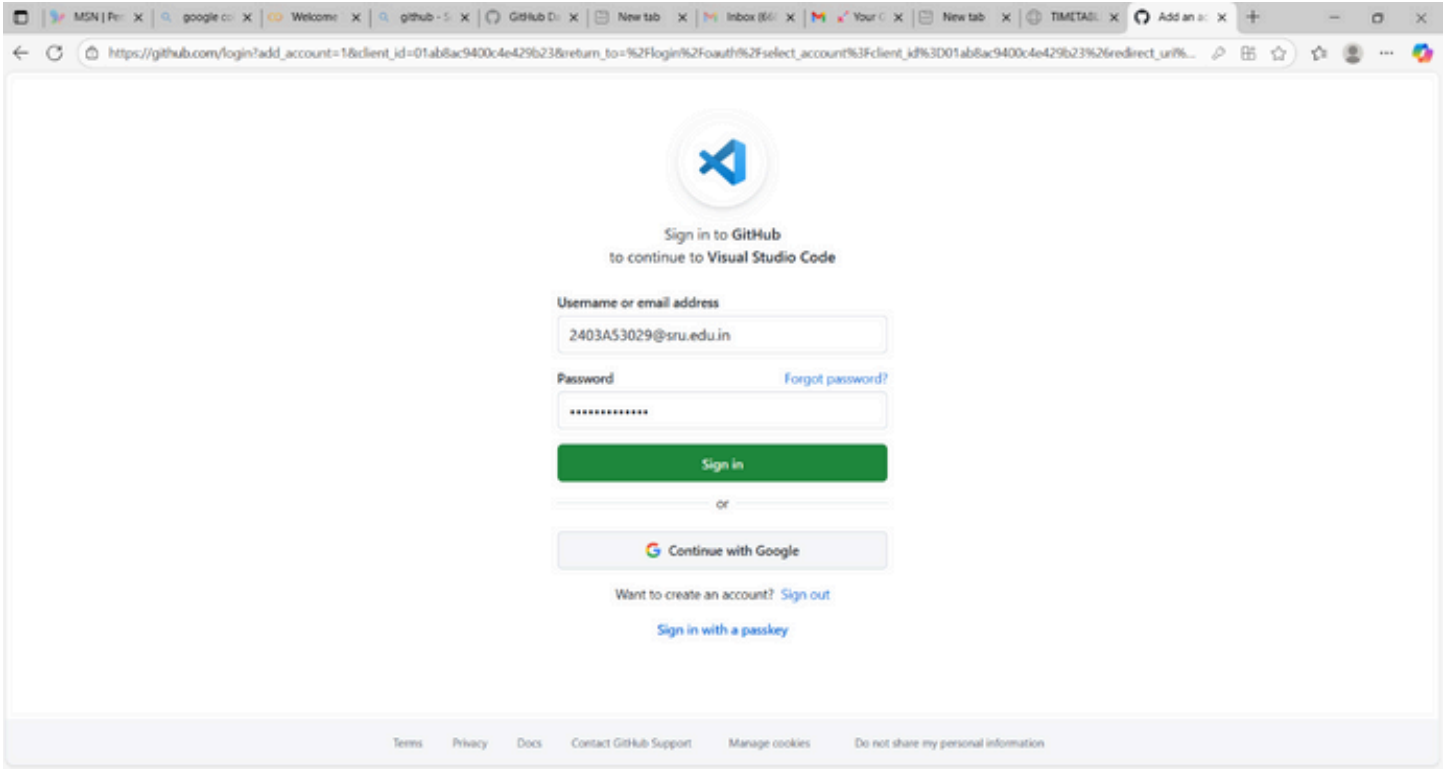
And it will direct to sign in page.



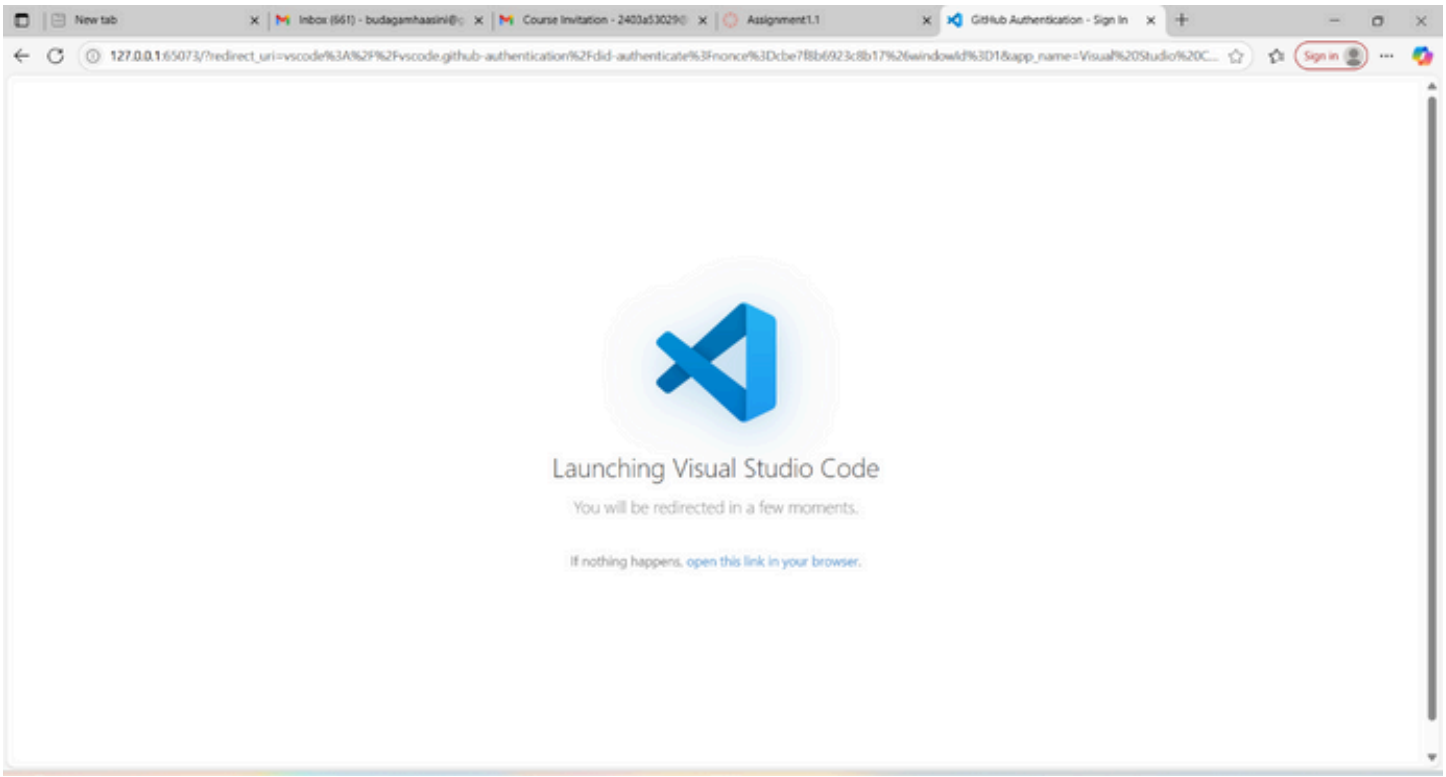
Signing into GitHub

Here, I am entering the Mail and password.





Launching of GitHub in vscode.

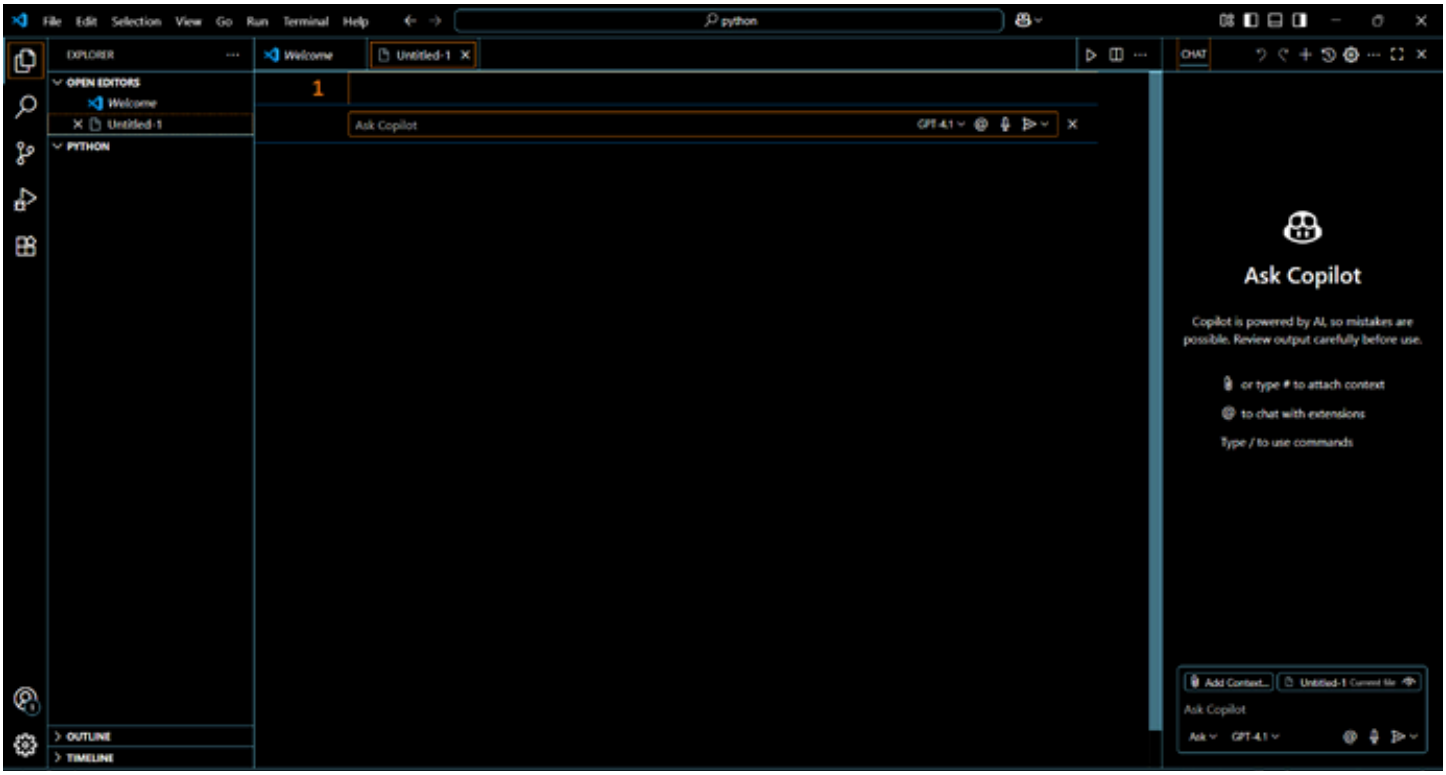


This connects GitHub Copilot to the Vs Code through installation of GitHub Copilot extension. By selecting the language, we can start using GitHub Copilot.

Ctrl + I Is the command for starting GitHub Copilot.

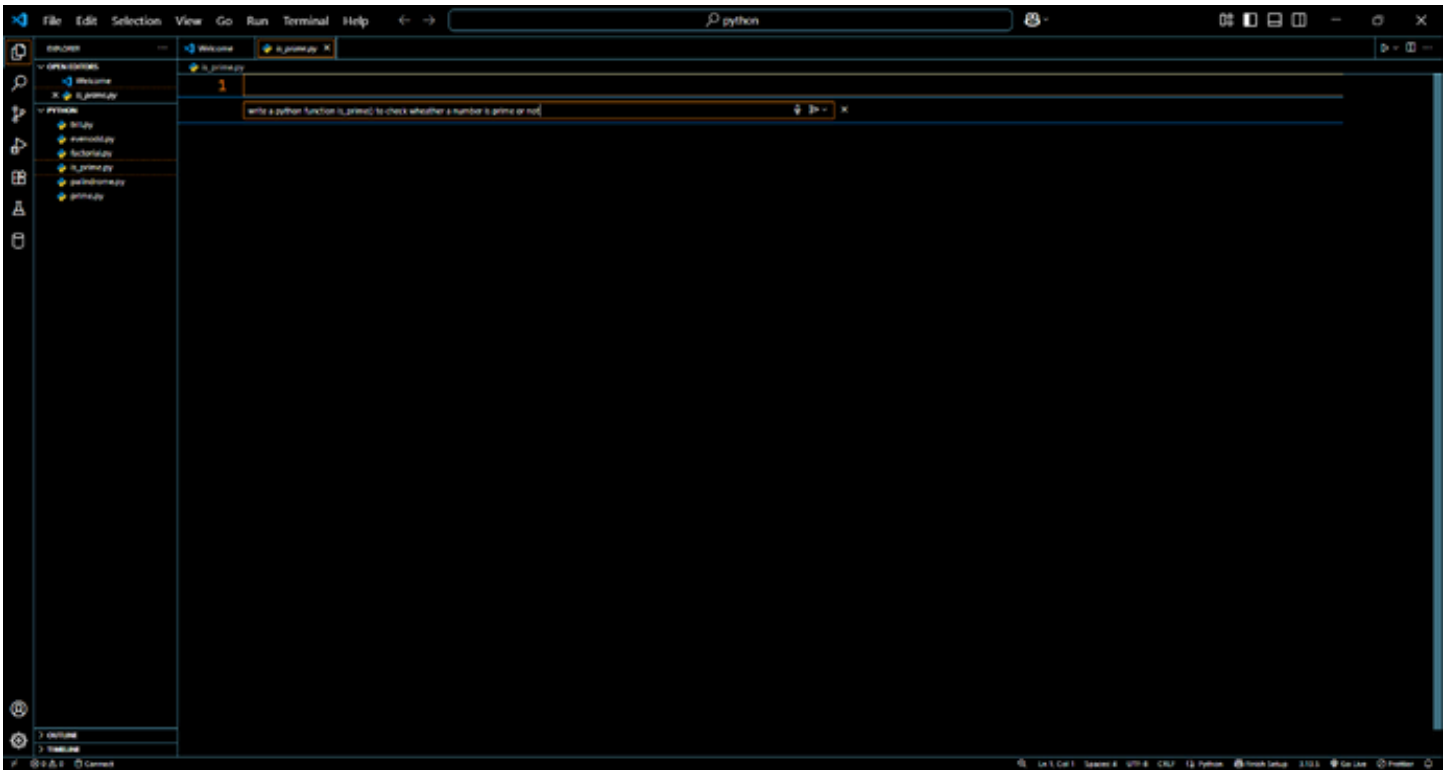
Task 2: is prime or not a prime number with function

- Ctrl + I for GitHub Copilot



- **Asking the GitHub Copilot to give the prime or not prime of a given number with function.**

Prompt: write a python program that check a number is prime or not.



- After that the GitHub Copilot takes the prompt and gives the code.

```python
def is_prime(n):
    if n <= 1:
        return False
    if n == 2:
        return True
    if n % 2 == 0:
        return False
    for i in range(3, int(n ** 0.5) + 1, 2):
        if n % i == 0:
            return False
    return True

if __name__ == "__main__":
    num = int(input("Enter a number: "))
    if is_prime(num):
        print(f"{num} is a prime number.")
    else:
        print(f"{num} is not a prime number.")
```

- Output:

-

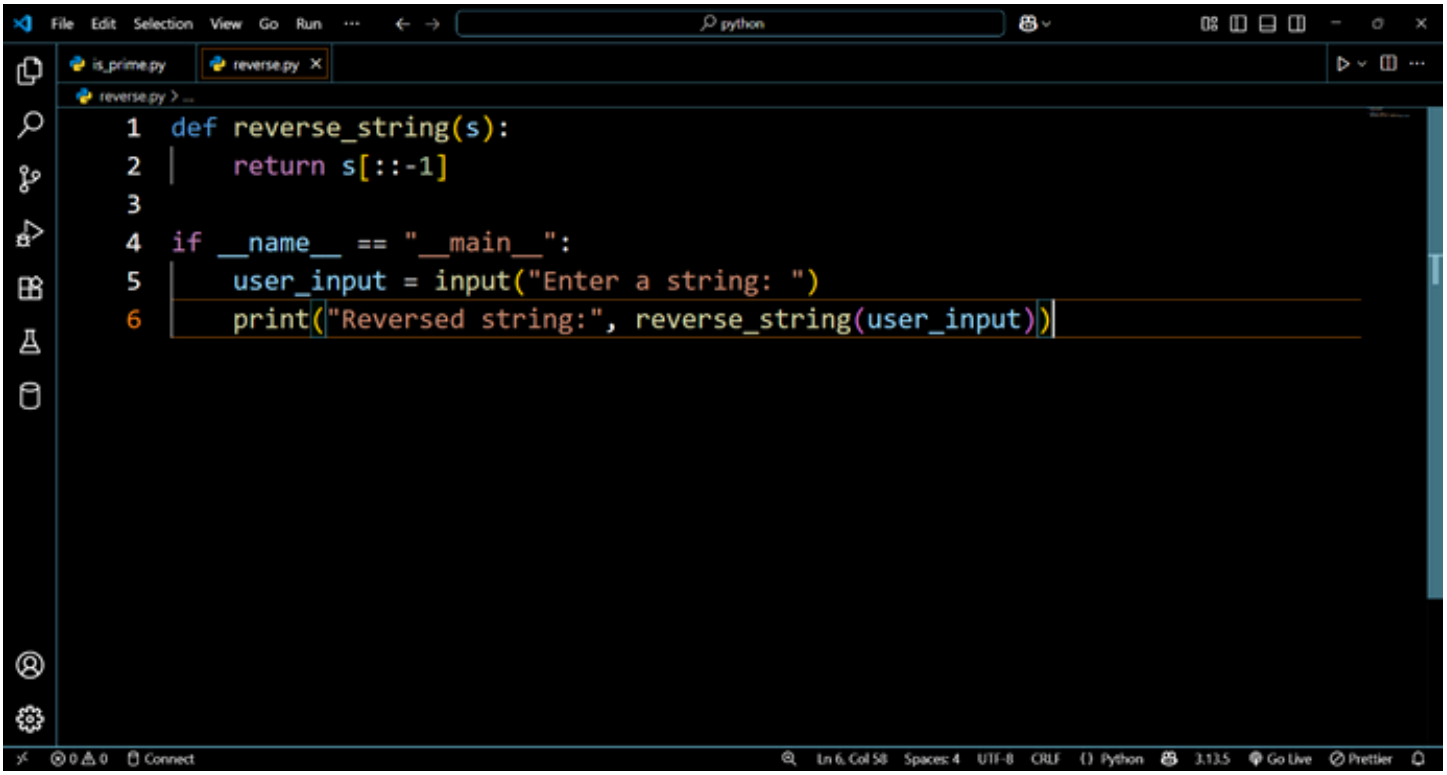
The above code is the simple code for checking a number is prime or not.

Task 3: Description: function to find reverse of a string

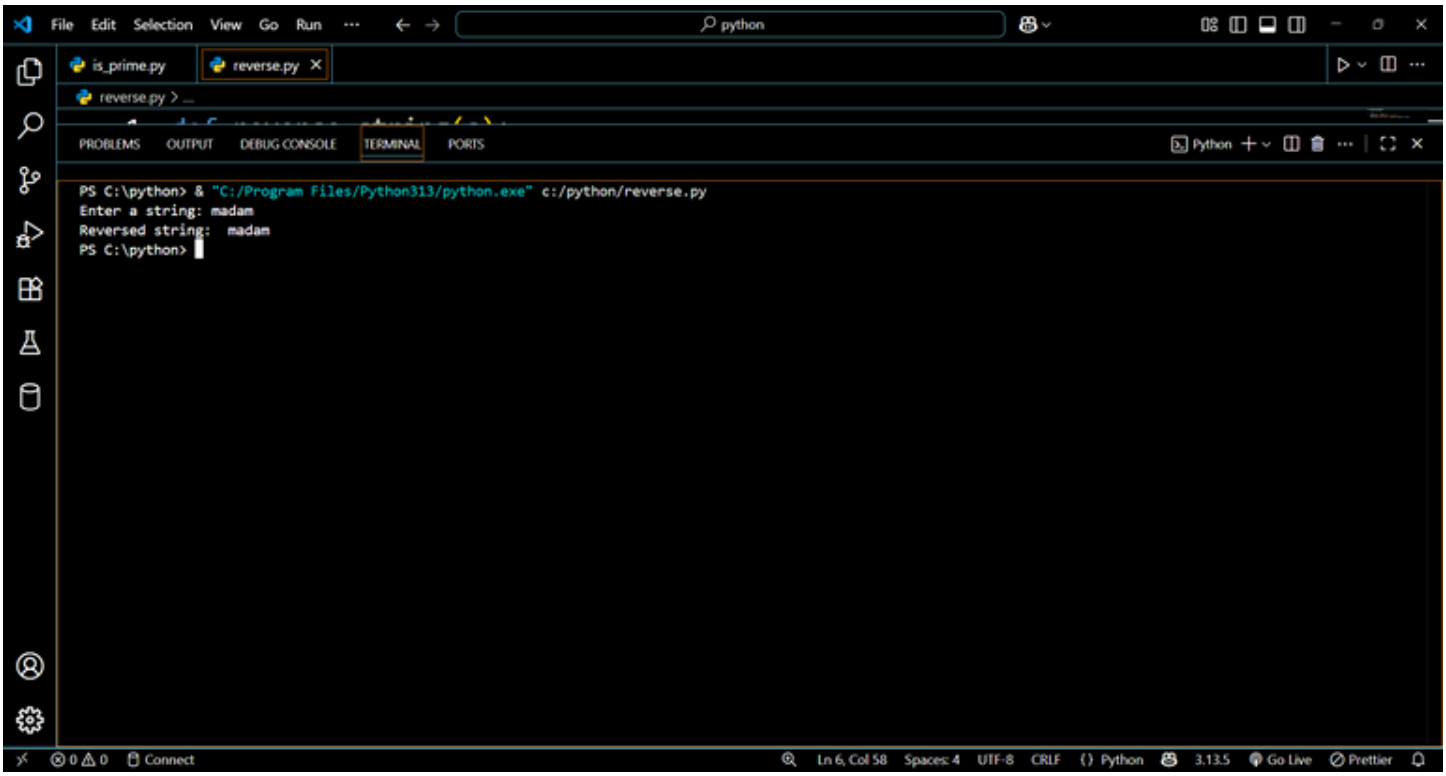
Prompt: write a python function to reverse a string.


File Edit Selection View Go Run
python
is_prime.py
reverse.py
reverse.py
1
writre a python function to reverse a string
GPT-4.1
Ln 1, Col 1 Spaces: 4 UTF-8 CRLF Python 3.13.5 Go Live Prettier

code:


File Edit Selection View Go Run
python
is_prime.py
reverse.py
reverse.py > ...
1 def reverse_string(s):
2 return s[::-1]
3
4 if __name__ == "__main__":
5 user_input = input("Enter a string: ")
6 print("Reversed string:", reverse_string(user_input))
Ln 6, Col 58 Spaces: 4 UTF-8 CRLF Python 3.13.5 Go Live Prettier

output:


File Edit Selection View Go Run
python
is_prime.py
reverse.py
reverse.py > ...
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Python
PS C:\python> & "C:/Program Files/Python313/python.exe" c:/python/reverse.py
Enter a string: madam
Reversed string: madam
PS C:\python>
0 0 Connect
Ln 6, Col 58 Spaces: 4 UTF-8 CRLF {} Python 3.13.5 Go Live Prettier

Task 4:

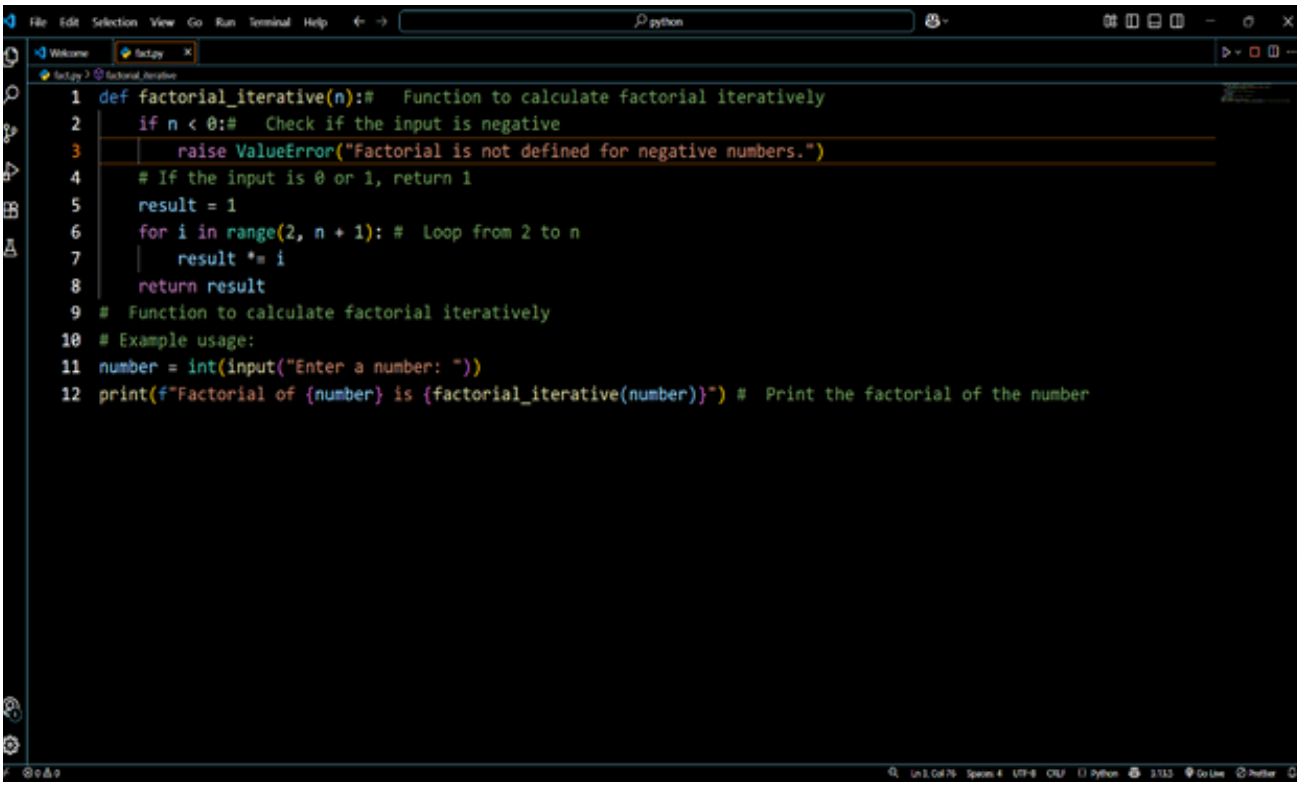
Iterative vs Recursive Factorial

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

Iterative function:

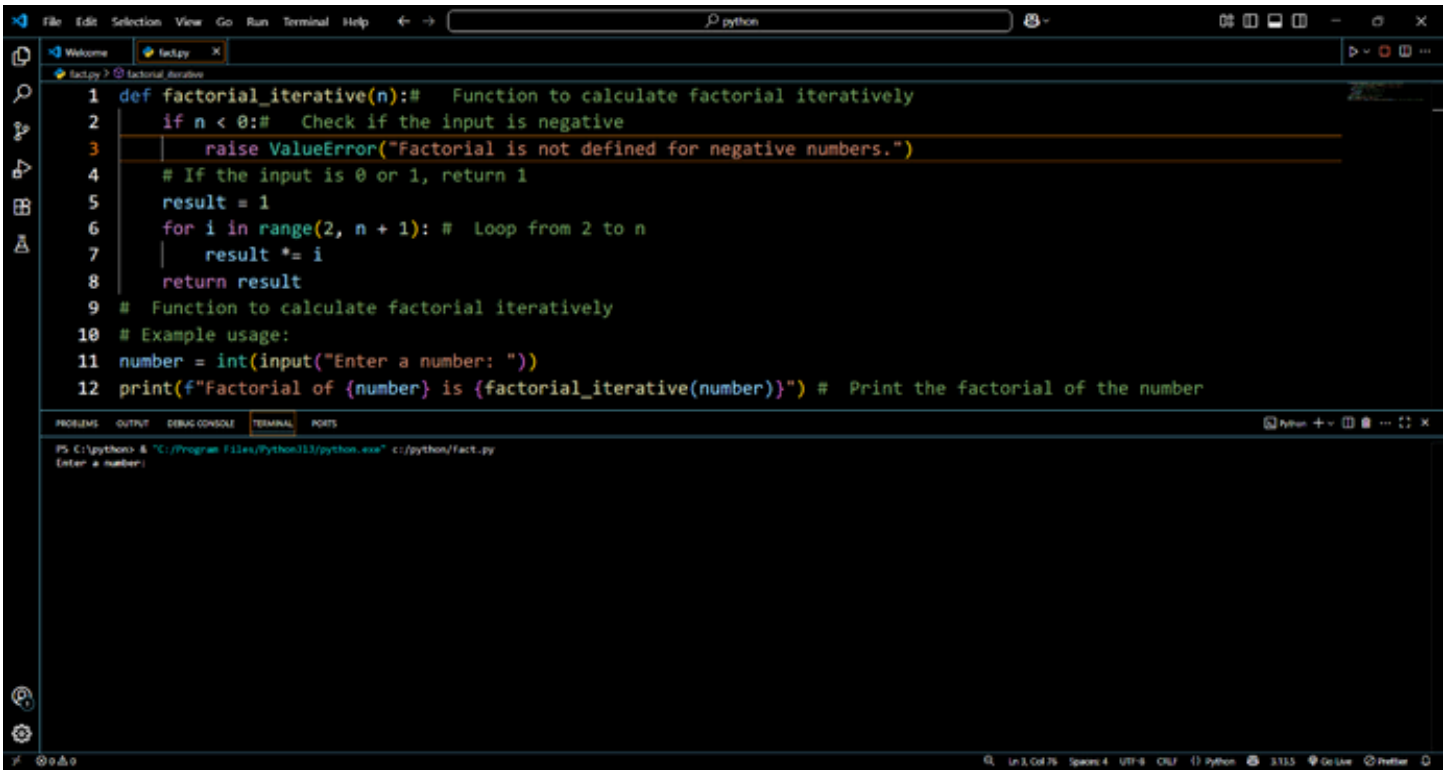
Prompt: write the python program for finding the factorial of number by Iterative function.

Code:



```python
def factorial_iterative(n):#   Function to calculate factorial iteratively
    if n < 0:#   Check if the input is negative
        raise ValueError("Factorial is not defined for negative numbers.")
    # If the input is 0 or 1, return 1
    result = 1
    for i in range(2, n + 1): #  Loop from 2 to n
        result *= i
    return result
#  Function to calculate factorial iteratively
# Example usage:
number = int(input("Enter a number: "))
print(f"Factorial of {number} is {factorial_iterative(number)}") #  Print the factorial of the number
```

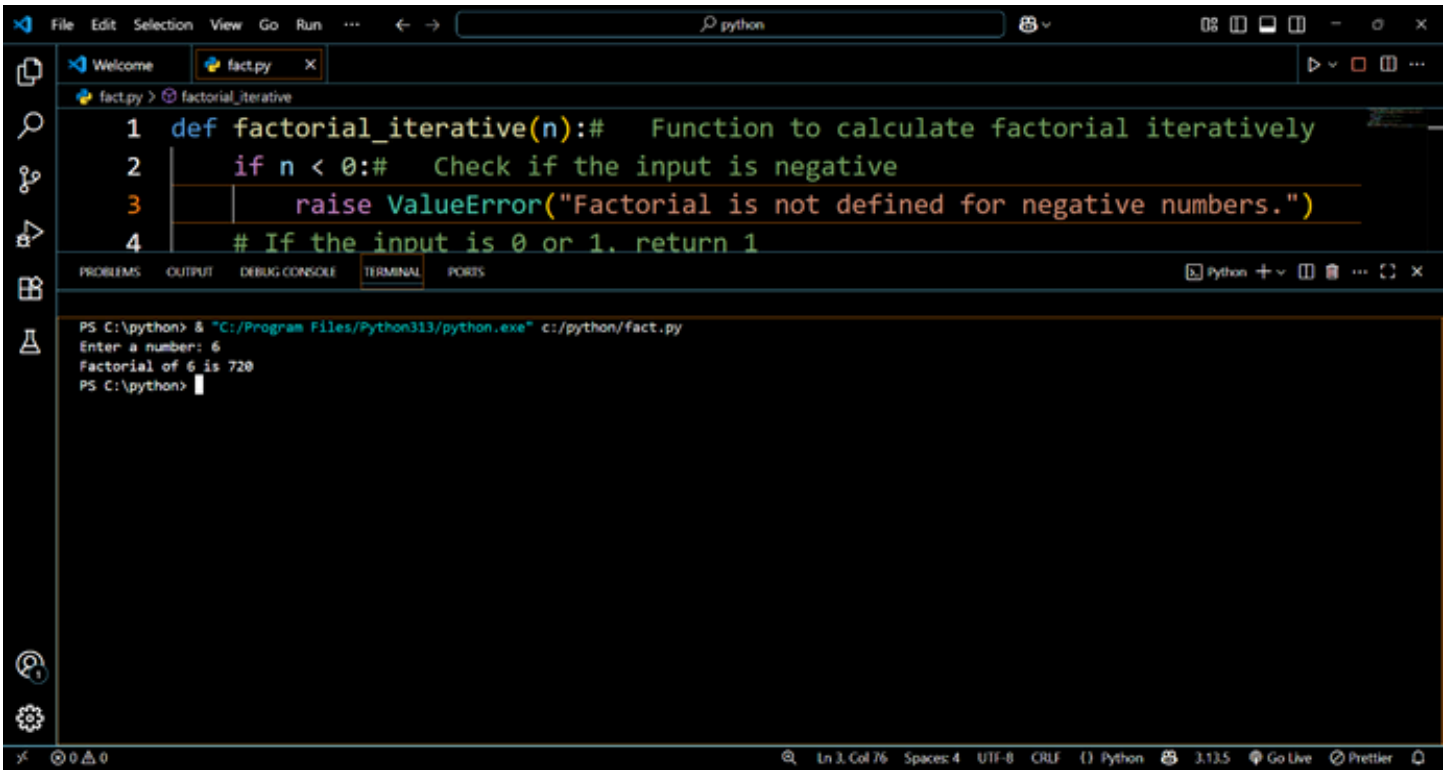
Asking for input:



```python
def factorial_iterative(n):#   Function to calculate factorial iteratively
    if n < 0:#   Check if the input is negative
        raise ValueError("Factorial is not defined for negative numbers.")
    # If the input is 0 or 1, return 1
    result = 1
    for i in range(2, n + 1): #  Loop from 2 to n
        result *= i
    return result
#  Function to calculate factorial iteratively
# Example usage:
number = int(input("Enter a number: "))
print(f"Factorial of {number} is {factorial_iterative(number)}") #  Print the factorial of the number
```

```
PS C:\python> & "C:/Program Files/Python313/python.exe" c:/python/Fact.py
Enter a number:
```

Output:

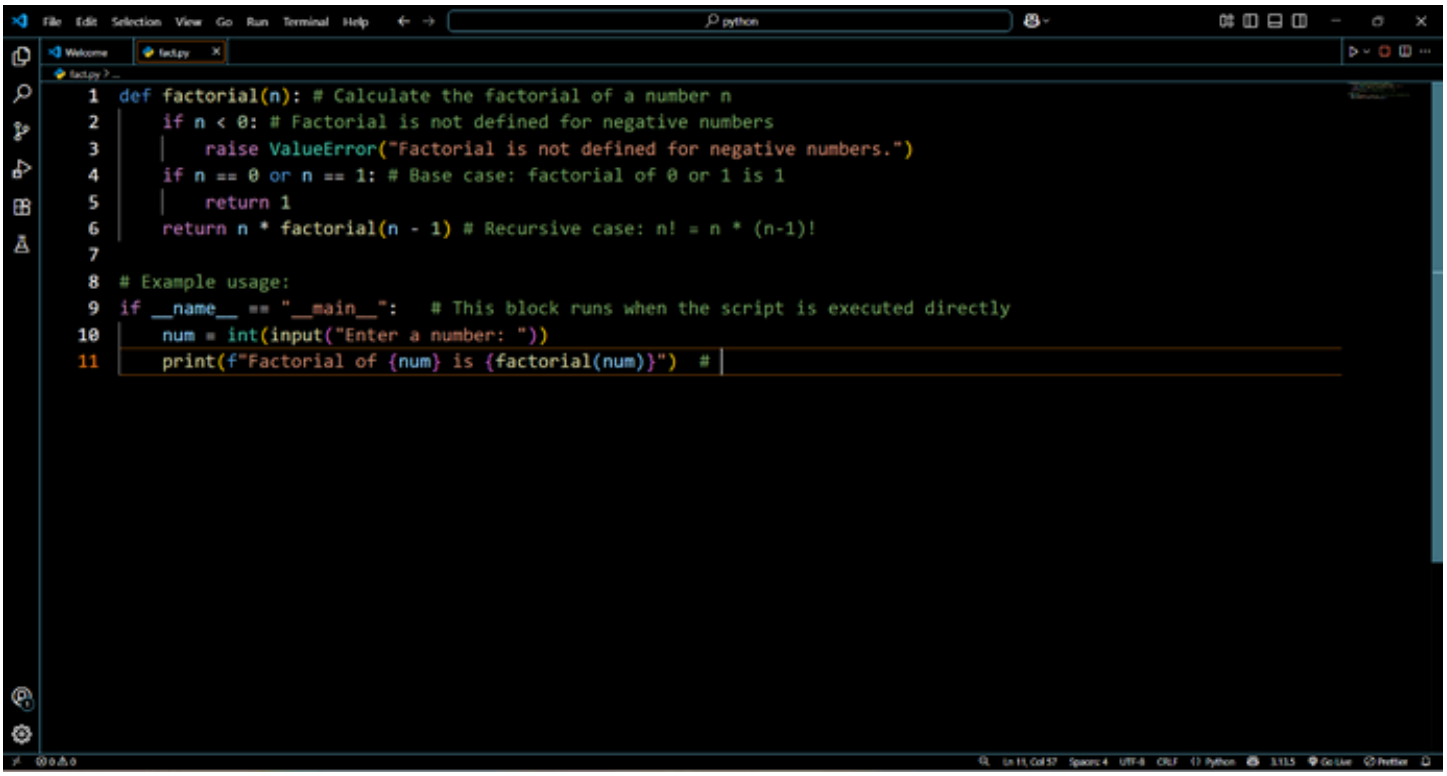


Recursive function for finding the Factorial of a number:

Prompt: write the python program for finding the factorial of a number in recursive function.

Copilot generating the Code:

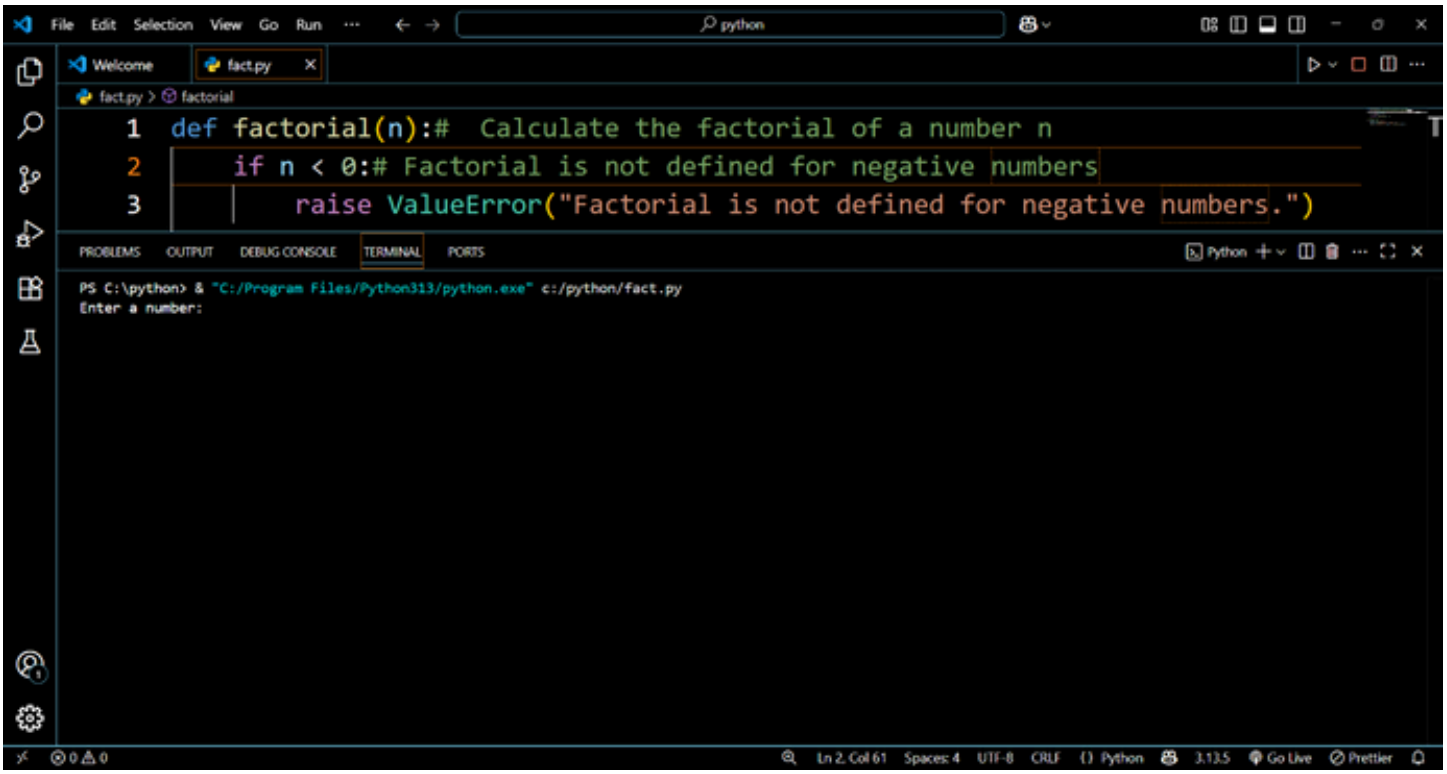
Code:



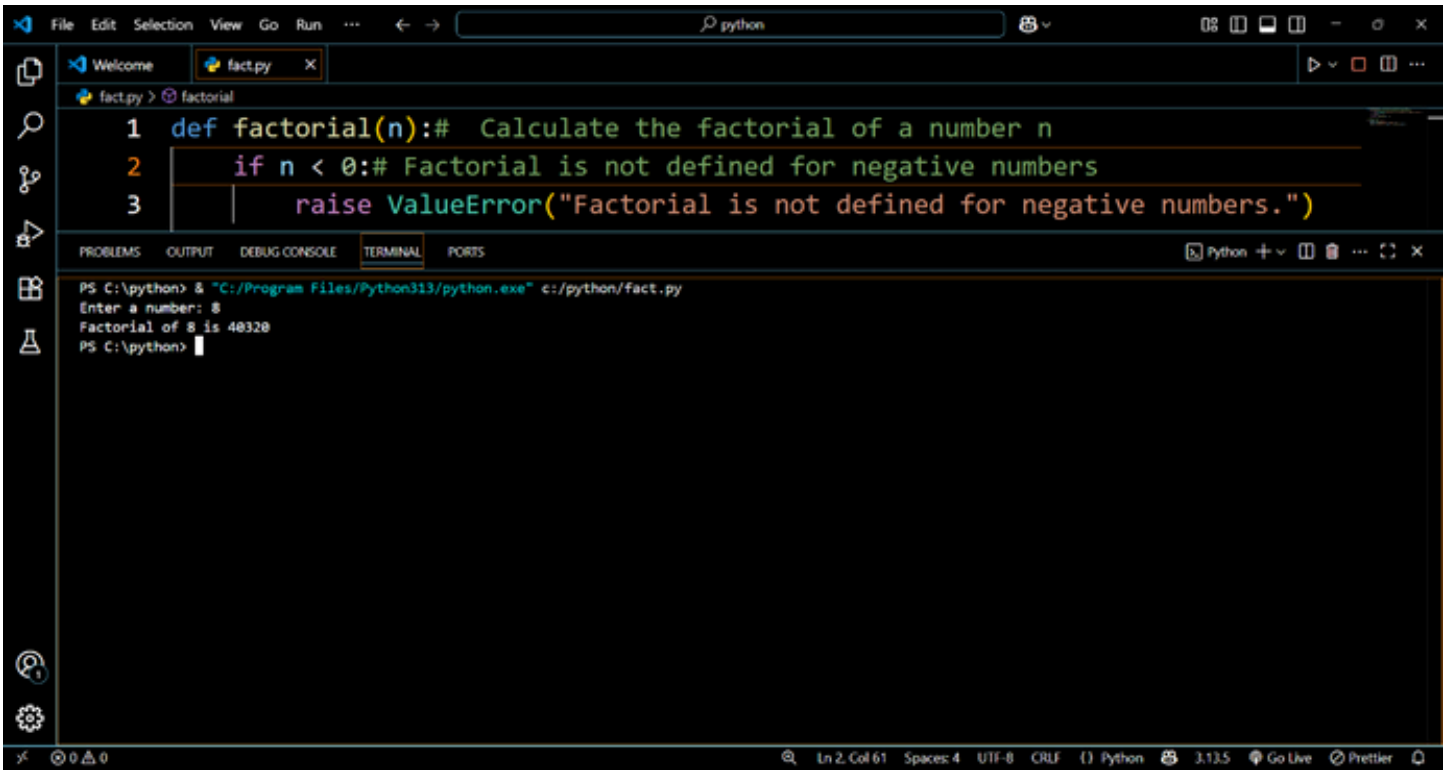
```python
def factorial(n): # Calculate the factorial of a number n
    if n < 0: # Factorial is not defined for negative numbers
        raise ValueError("Factorial is not defined for negative numbers.")
    if n == 0 or n == 1: # Base case: factorial of 0 or 1 is 1
        return 1
    return n * factorial(n - 1) # Recursive case: n! = n * (n-1)!

# Example usage:
if __name__ == "__main__":   # This block runs when the script is executed directly
    num = int(input("Enter a number: "))
    print(f"Factorial of {num} is {factorial(num)}")  #
```

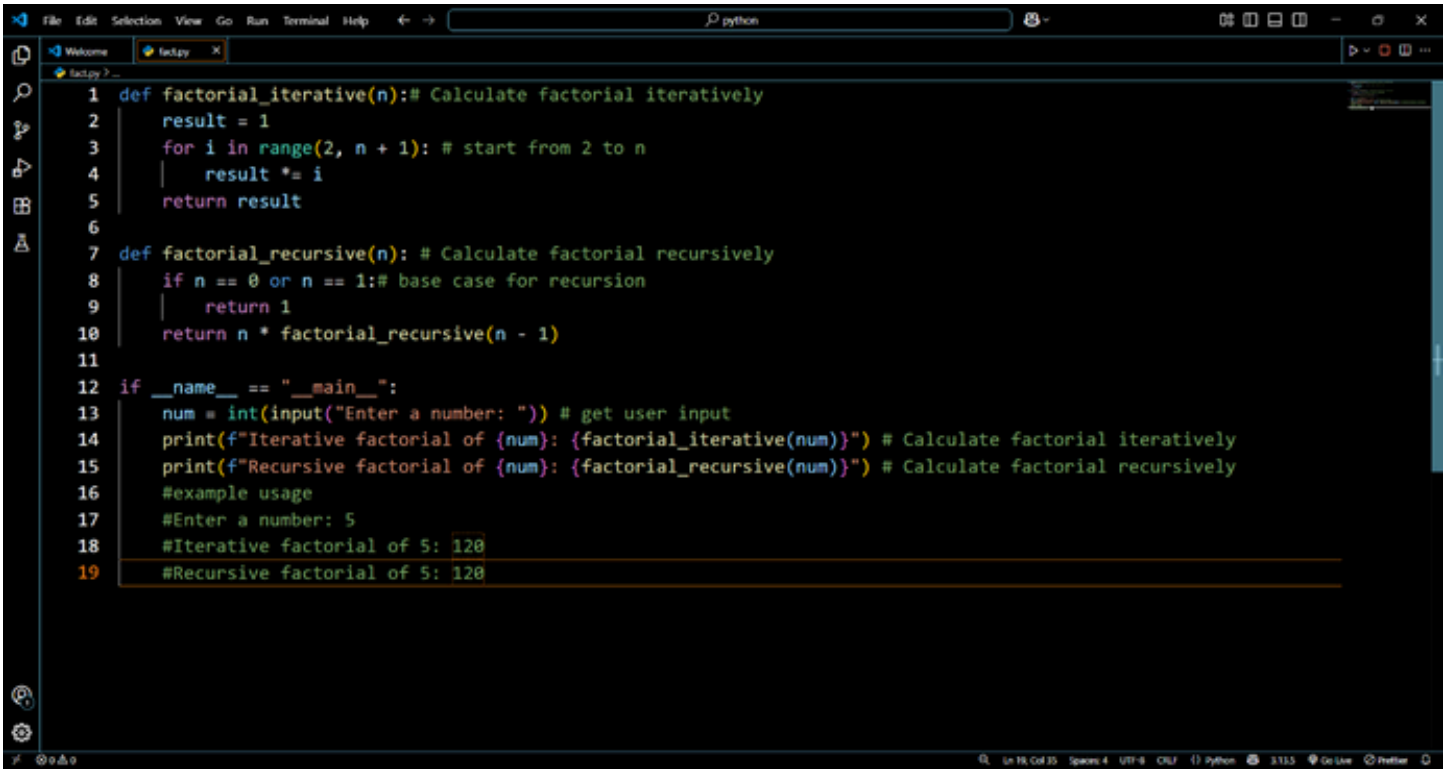
Output:

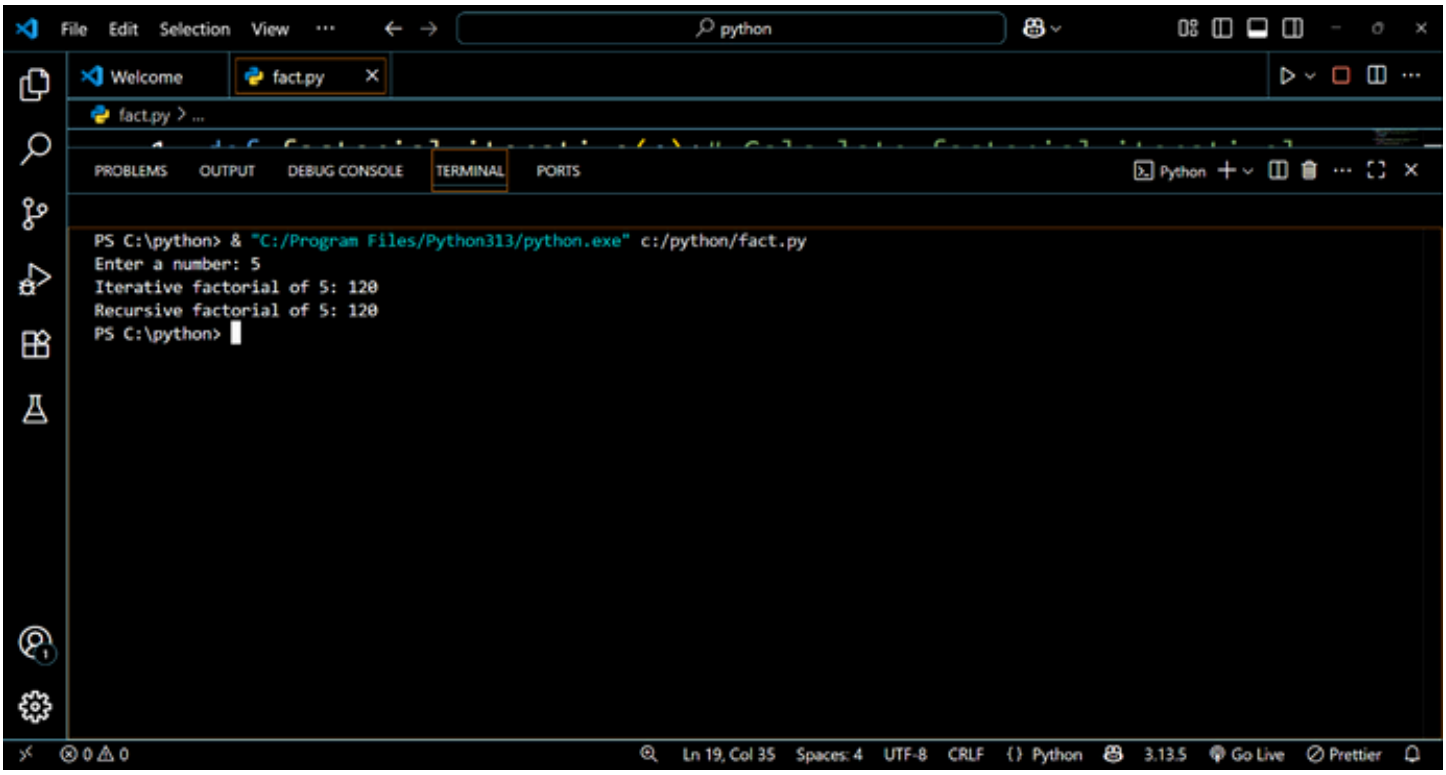


```
PS C:\python> & "C:/Program Files/Python313/python.exe" c:/python/fact.py
Enter a number:
```


File Edit Selection View Go Run
python
Welcome
fact.py
fact.py > factorial
1 def factorial(n):# Calculate the factorial of a number n
2 if n < 0:# Factorial is not defined for negative numbers
3 raise ValueError("Factorial is not defined for negative numbers.")
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Python
PS C:\python> & "C:/Program Files/Python313/python.exe" c:/python/fact.py
Enter a number: 8
Factorial of 8 is 40320
PS C:\python>
Ln 2, Col 61 Spaces: 4 UTF-8 CRLF Python 3.13.5 Go Live Prettier

FINAL CODE:


File Edit Selection View Go Run Terminal Help
python
Welcome
fact.py
1 def factorial_iterative(n):# Calculate factorial iteratively
2 result = 1
3 for i in range(2, n + 1): # start from 2 to n
4 result *= i
5 return result
6
7 def factorial_recursive(n): # Calculate factorial recursively
8 if n == 0 or n == 1:# base case for recursion
9 return 1
10 return n * factorial_recursive(n - 1)
11
12 if __name__ == "__main__":
13 num = int(input("Enter a number: ")) # get user input
14 print(f"Iterative factorial of {num}: {factorial_iterative(num)}") # Calculate factorial iteratively
15 print(f"Recursive factorial of {num}: {factorial_recursive(num)}") # Calculate factorial recursively
16 #example usage
17 #Enter a number: 5
18 #Iterative factorial of 5: 120
19 #Recursive factorial of 5: 120



Comparison of logic, performance, and execution flow between iterative and recursive approaches:

In the Task 5 I implemented both iterative and recursive approaches of finding the factorial of a number with outputs using GitHub Copilot.

Logic comparison:

Iterative approach:

Iterative implementation uses a simple loop (for i in range (2, n+1) ignoring 1 which saves extra iteration and avoids useless calculation.

Recursive approach:

In Recursive approach, uses a function calls itself with n-1 until it reaches the base case. This approach is more detailed into real mathematical definition of factorial.

Performance Comparison:

Iterative: Time Complexity is O(n) and O (1)

Speed- speed is faster, especially for large and safe to use.

Recursive: Time Complexity is O(n) and space complexity O(n) – due to recursive call stack.

Speed- Slower for large n due to function call overhead. Risk of stack overflow for large inputs.

Execution Flow Comparison

- **Iterative Flow**:

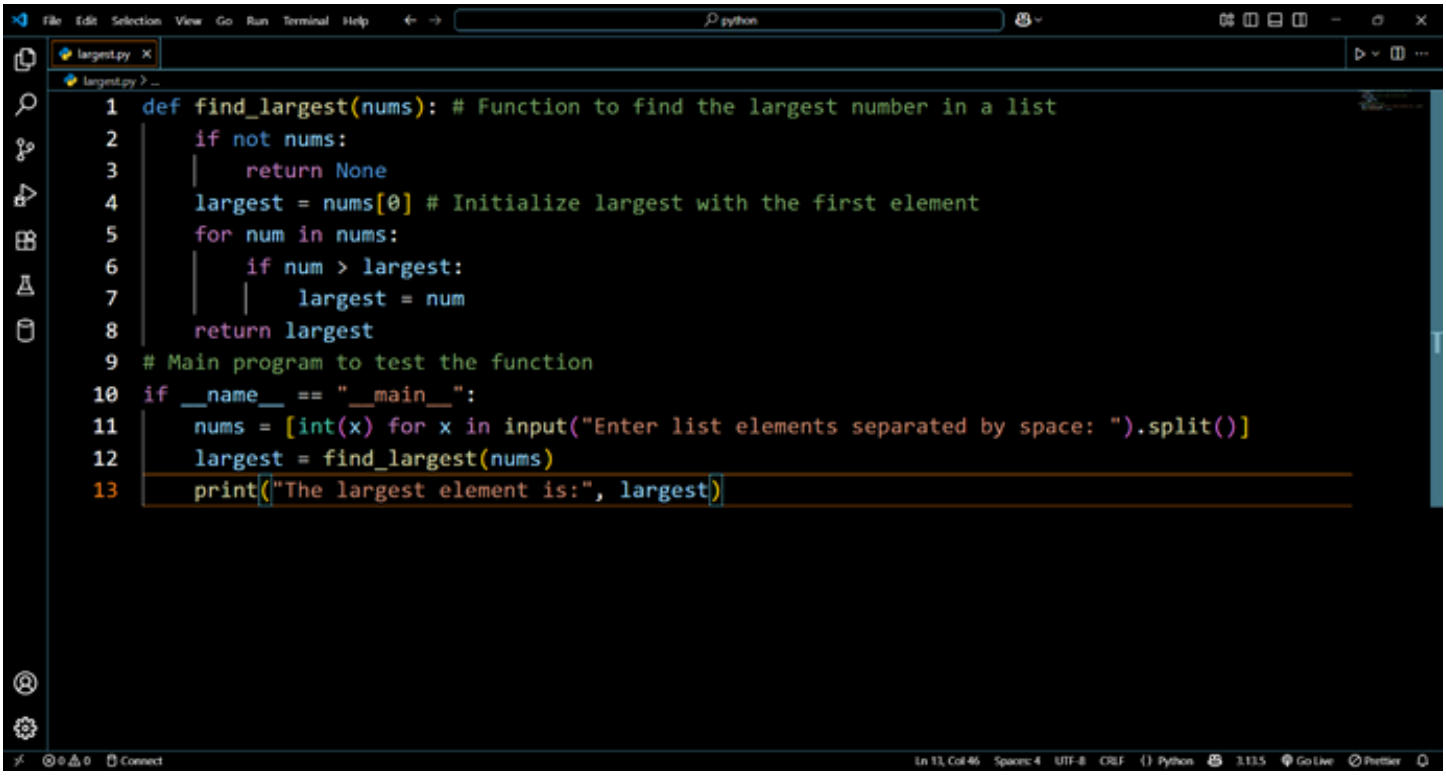
- The program enters a loop.
- Each step multiplies the result by the next number.
- Once the loop ends, it returns the final result.

- **Recursive Flow**:
  - Function keeps calling itself with smaller values of n.
  - Once it reaches base case, it starts returning values back up the call stack.
  - Final result is computed during the "unwinding" phase.

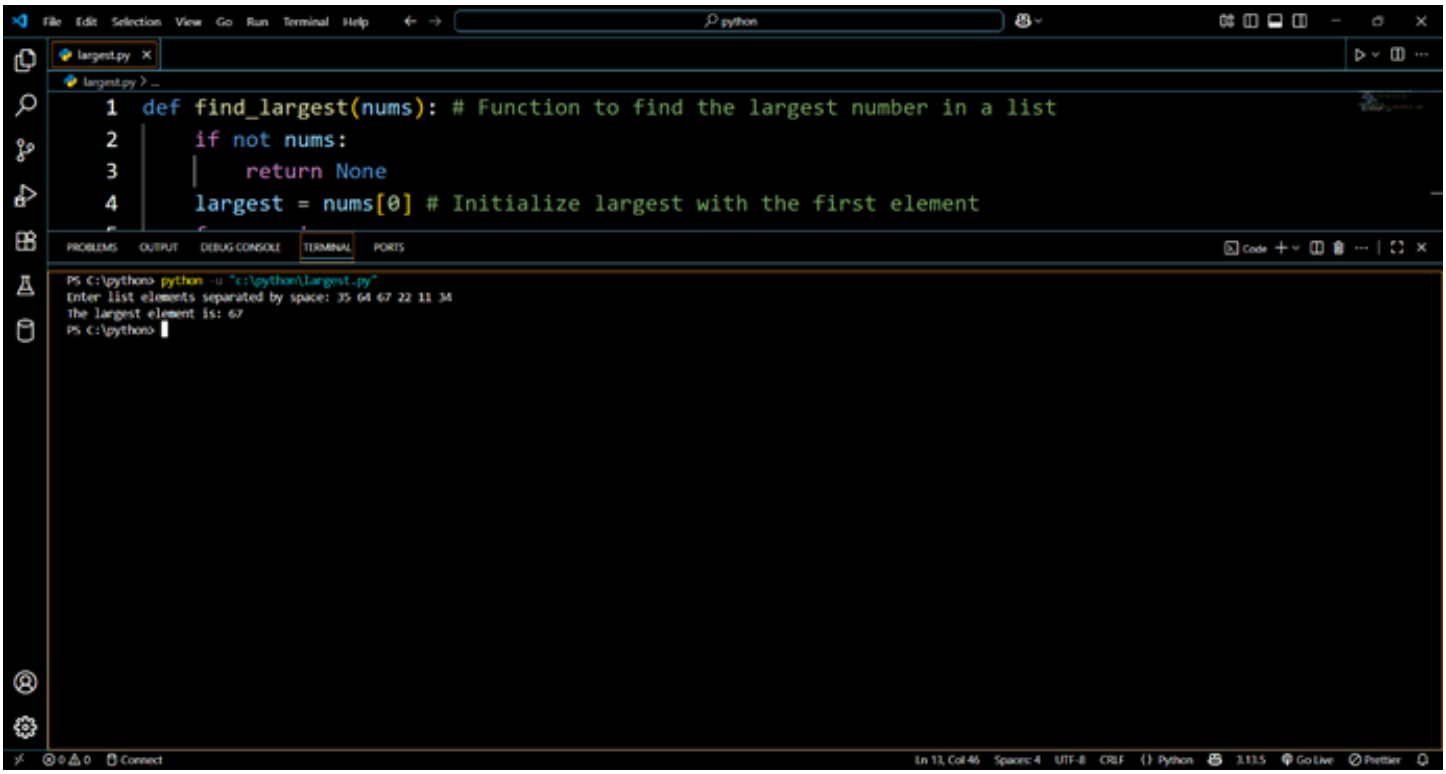
Task 5: find the largest number in a list.

Prompt: write a python code to find the largest number in a list.

Code:



OUTPUT:



CODE OBSERVATIONS:

- **Clarity**:
  - The function find_largest has a clear and descriptive name.
  - Inline comment for initialization is helpful, but more function-level documentation (docstring) would improve maintainability.
- **Readability**:
  - Code is neatly indented and easy to read.
  - Variable names (nums, largest, num) are simple and meaningful.
- **Structure**:
  - The program is well-structured with a separate function for logic and a main section for input/output.
  - Follows good practice by using if __name__ == "__main__": block.
- **Error Handling**:
  - Handles empty list case with if not nums: return None.
  - No handling for non-integer input (could cause ValueError).
- **Best Practices**:
  - Could use Python's built-in max() for simplicity, but writing your own loop demonstrates algorithmic thinking.
- SUMMARY OF CODE:

The Python program finds the largest number in a list entered by the user. It defines a function find_largest(nums) that first checks if the list is empty; if so, it returns None. Otherwise, it initializes the largest value with the first element and iterates through the list, updating the largest value when a bigger number is found. The main block takes space-separated integers as input, calls the function, and prints the largest element. The algorithm runs in **O(n)** time and **O(1)** space, making it optimal for unsorted lists.