

hall ticket: 2403A54112

name: Neela.Sai shivathmika

batch: 24BTCAICSB18

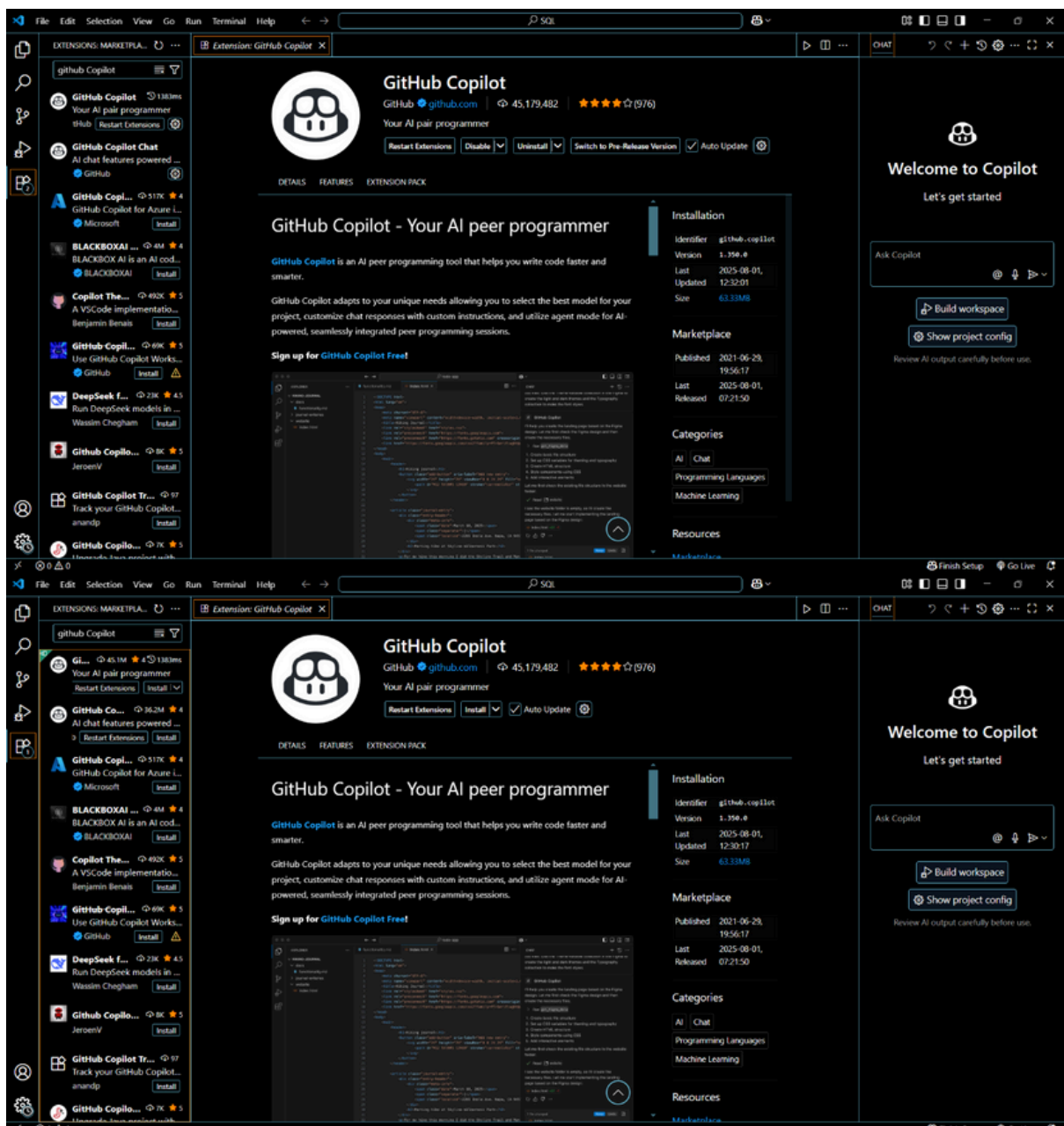
Lab 1: Environment Setup – GitHub Copilot and VS Code Integration

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

Installation of GitHub Copilot in Vs Code.

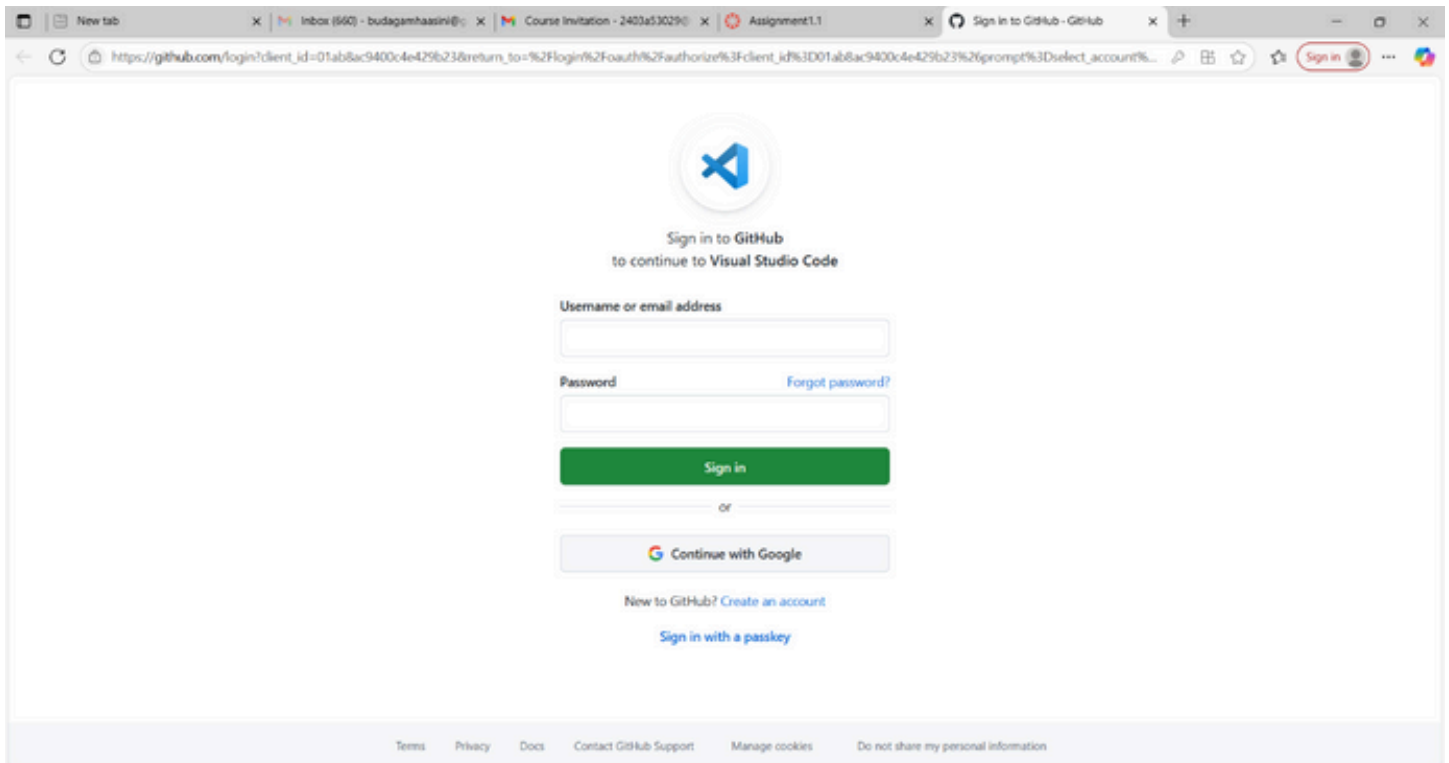
Here, we have to search for the GitHub Copilot extension in Vs Code and install it.

And it will direct to sign in page.



Signing into GitHub

Here, I am entering the Mail and password.



The screenshot shows the GitHub login page for Visual Studio Code. The browser's address bar displays a URL with a client ID and a return URL. The page features the GitHub logo at the top, followed by the text "Sign in to GitHub to continue to Visual Studio Code". Below this, there are two input fields: "Username or email address" and "Password". A "Forgot password?" link is located next to the password field. A green "Sign in" button is positioned below the input fields. Below the button, there is an "or" separator and a "Continue with Google" button. At the bottom of the login section, there are links for "New to GitHub? Create an account" and "Sign in with a passkey". The footer contains links for "Terms", "Privacy", "Docs", "Contact GitHub Support", "Manage cookies", and "Do not share my personal information".


Sign in to GitHub
to continue to Visual Studio Code

Username or email address

Password [Forgot password?](#)

Sign in

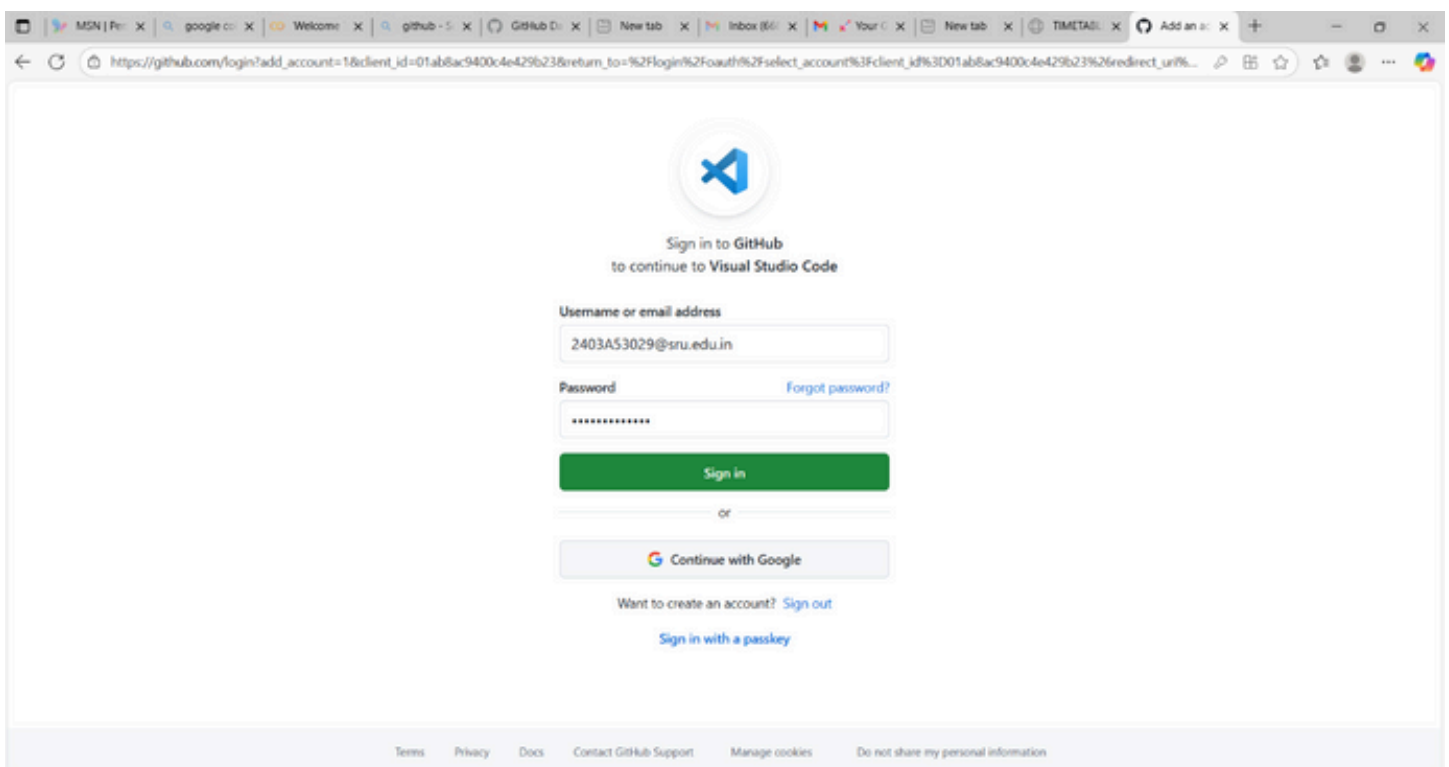
or

 Continue with Google

New to GitHub? [Create an account](#)

[Sign in with a passkey](#)

[Terms](#) [Privacy](#) [Docs](#) [Contact GitHub Support](#) [Manage cookies](#) [Do not share my personal information](#)



This screenshot shows the same GitHub login page, but with the "Username or email address" field filled with "2403A53029@sru.edu.in" and the "Password" field filled with "*****". The "Sign in" button is still visible. The "Continue with Google" button and the links for "New to GitHub? Create an account" and "Sign in with a passkey" are also present. The footer links remain the same.

Sign in to GitHub
to continue to Visual Studio Code


Username or email address

2403A53029@sru.edu.in

Password [Forgot password?](#)

Sign in

or

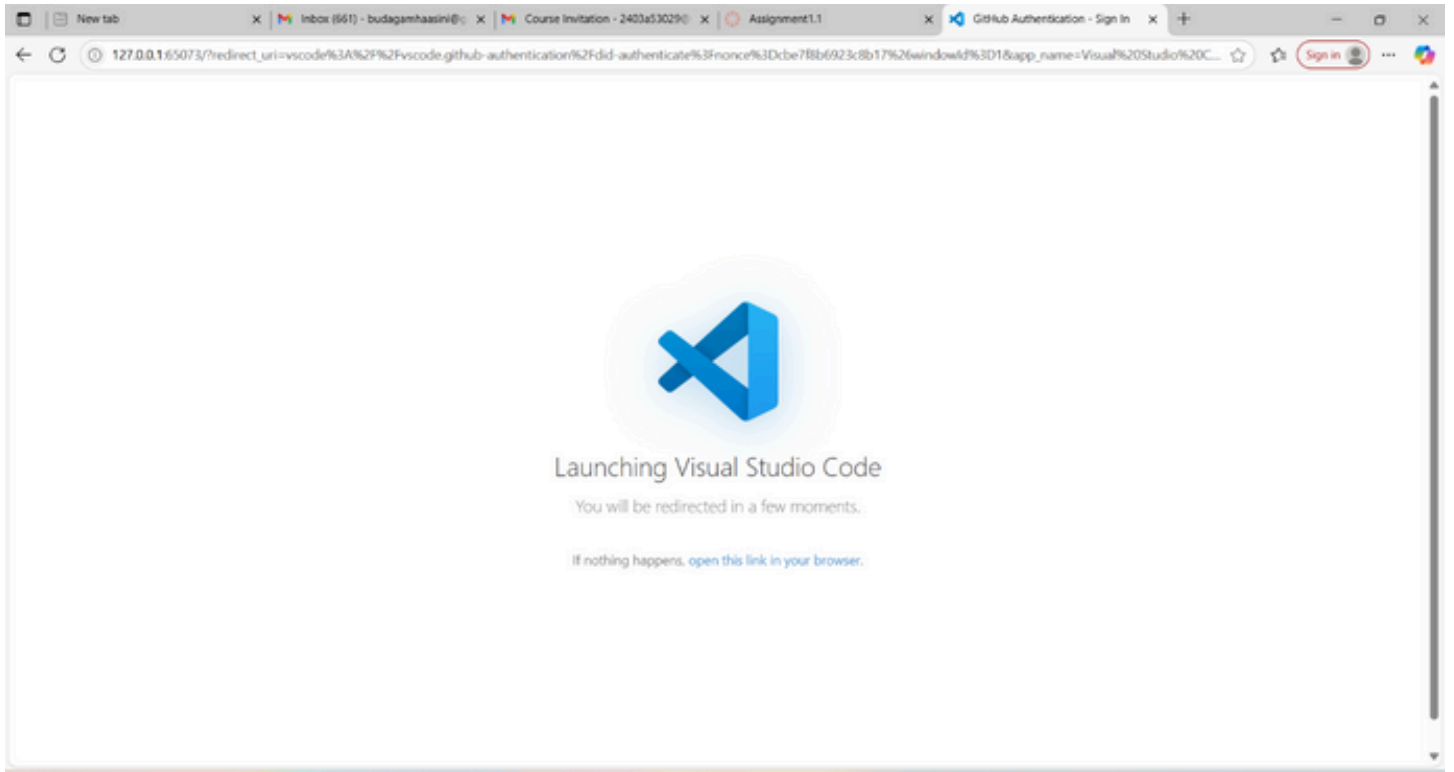
 Continue with Google

Want to create an account? [Sign out](#)

[Sign in with a passkey](#)

[Terms](#) [Privacy](#) [Docs](#) [Contact GitHub Support](#) [Manage cookies](#) [Do not share my personal information](#)

Launching of GitHub in vscode.

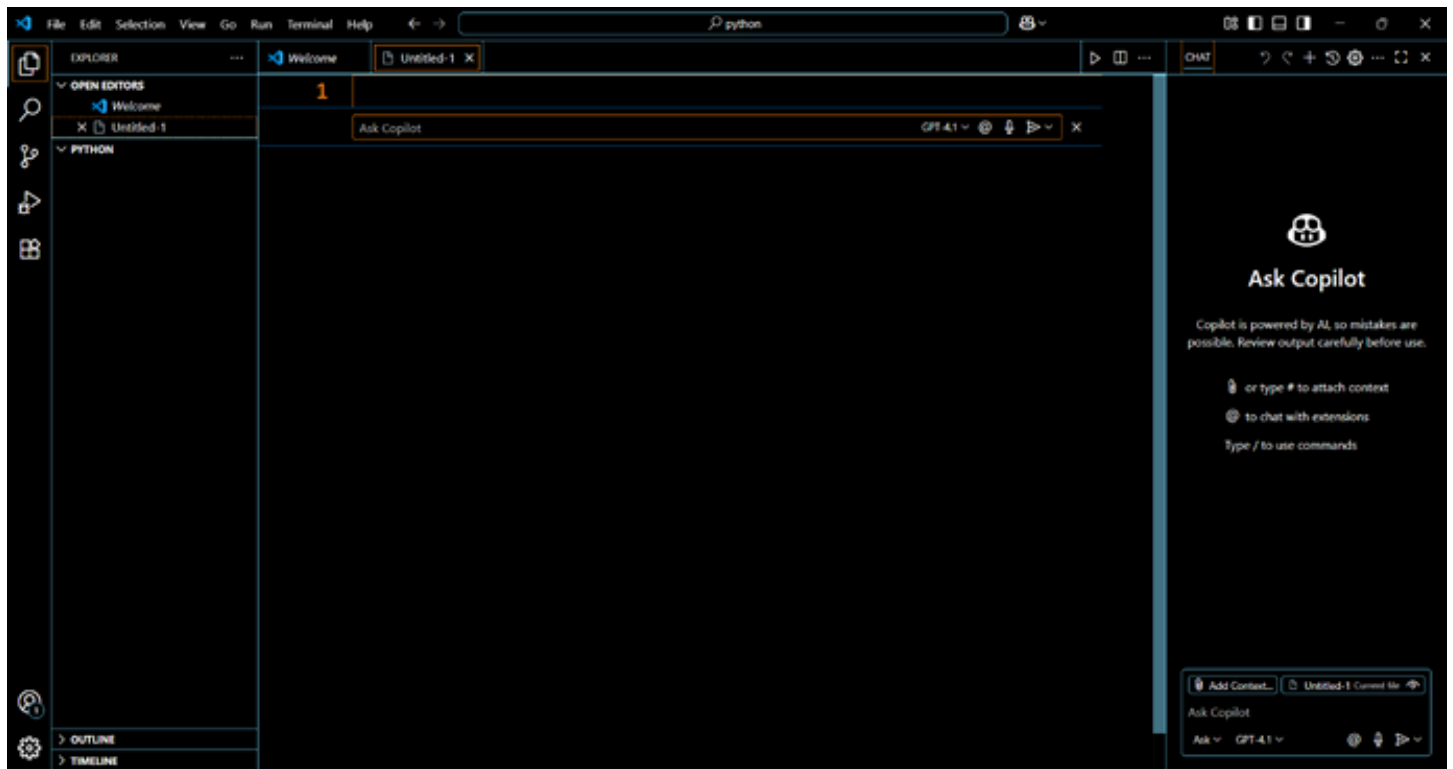


This connects GitHub Copilot to the Vs Code through installation of GitHub Copilot extension. By selecting the language, we can start using GitHub Copilot.

Ctrl + I Is the command for starting GitHub Copilot.

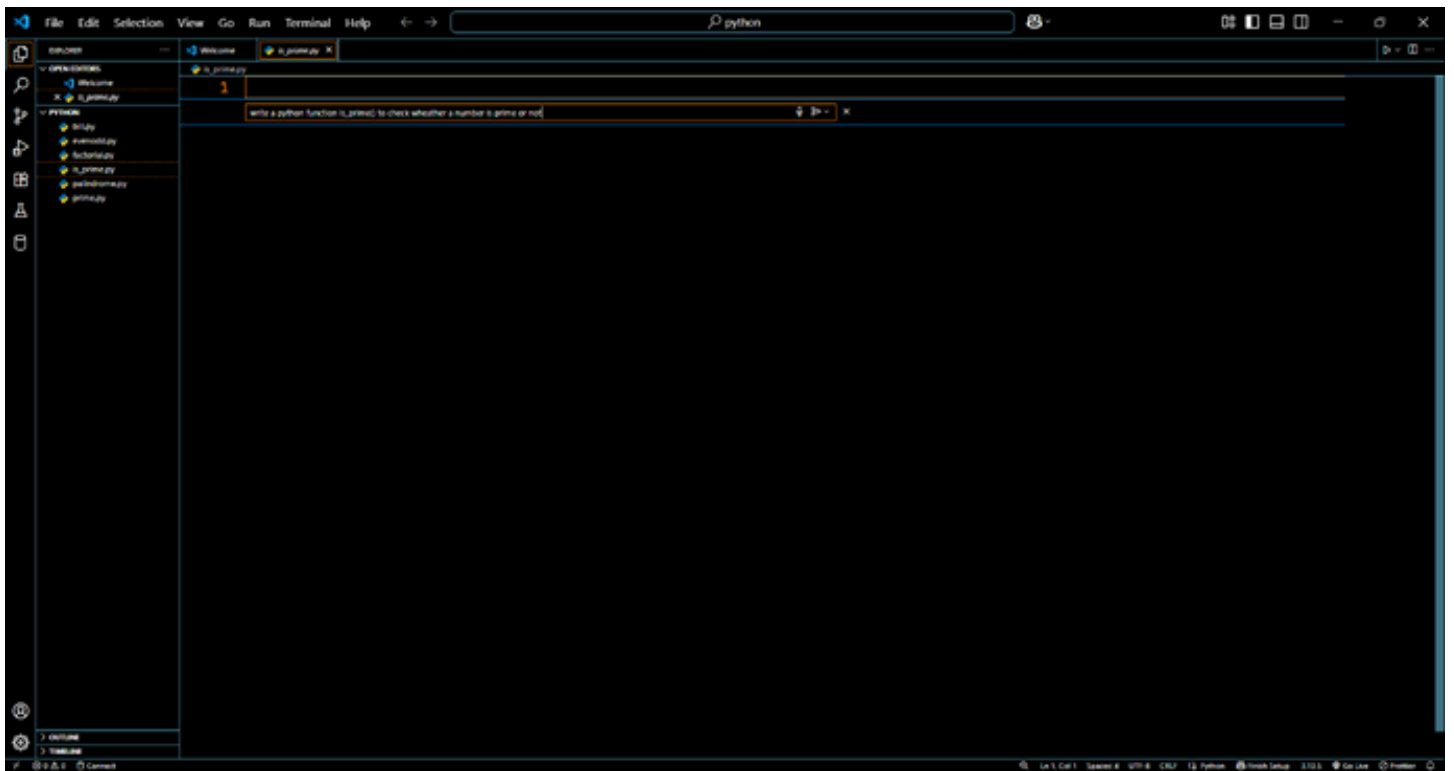
Task 2: is prime or not a prime number with function

- Ctrl + I for GitHub Copilot

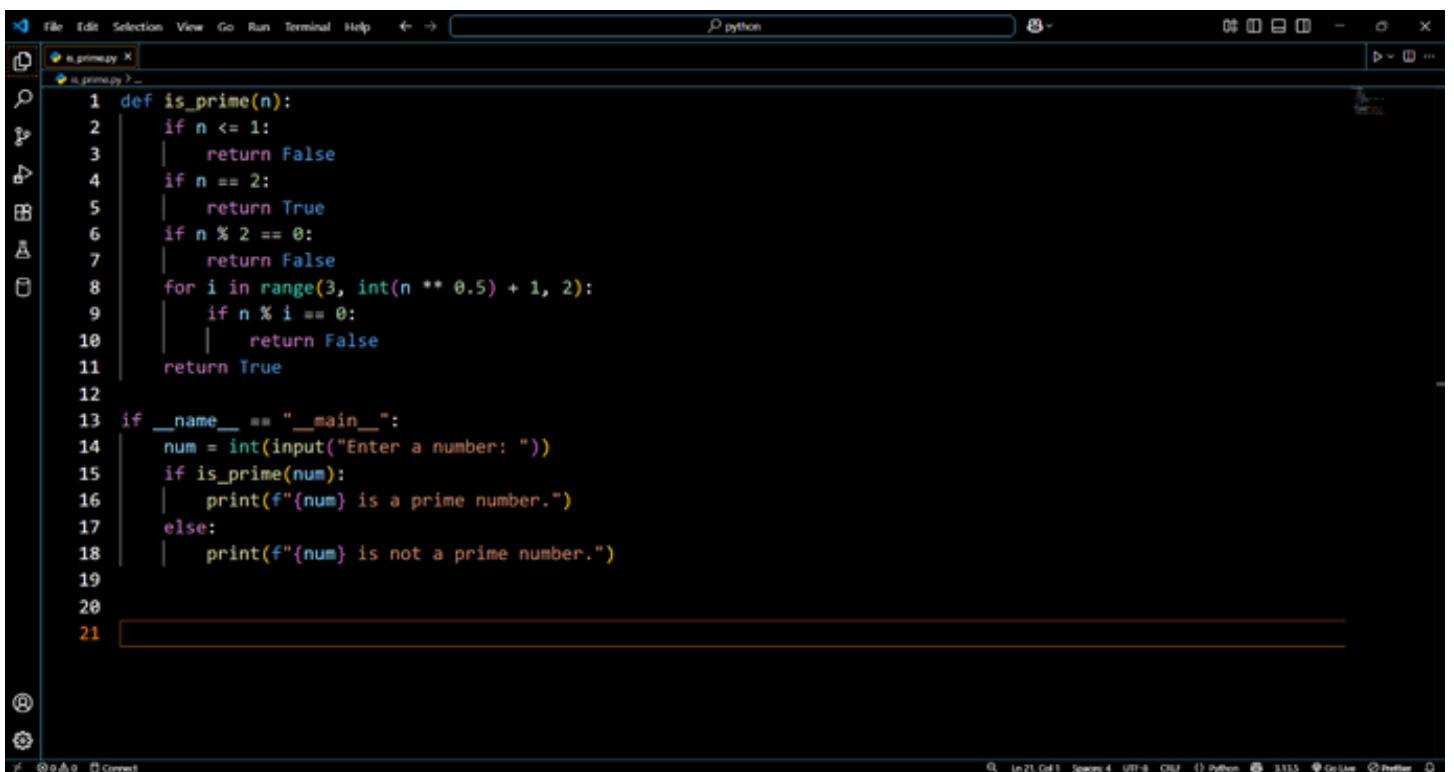


- Asking the GitHub Copilot to give the prime or not prime of a given number with function.

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.



- Output:



The image shows a Visual Studio Code (VS Code) interface with a terminal window open. The terminal is running a Python script named `is_prime.py`. The prompt is `PS C:\python> python -u "c:\python\is_prime.py"`. The script prompts the user to "Enter a number: 7" and outputs "7 is a prime number." The terminal window is titled `is_prime.py` and has tabs for `PROBLEMS`, `OUTPUT`, `DEBUG CONSOLE`, `TERMINAL`, and `PORTS`. The `TERMINAL` tab is active. The status bar at the bottom shows the file is at `Ln 21, Col 1`, with `Spaces: 4`, `UTF-8`, `CRLF`, and `Python` 3.13.5. The status bar also includes icons for `Go Live` and `Prettier`.

```
PS C:\python> python -u "c:\python\is_prime.py"
Enter a number: 7
7 is a prime number.
PS C:\python>
```

●

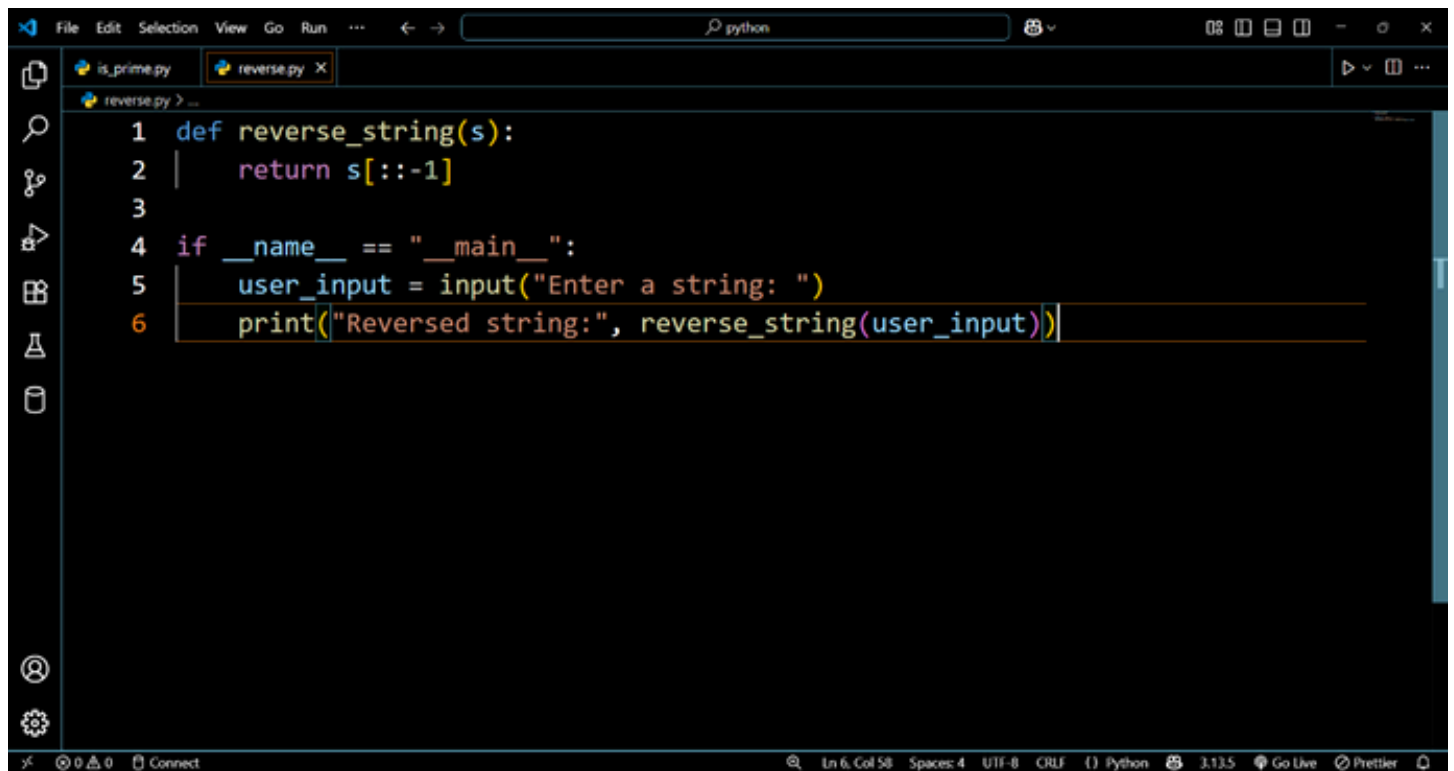
The above code is the simple code for checking a number is prime or not.

[Task 3: Description: function to find reverse of a string.](#)

Prompt: write a python function to reverse a string.



code:



output:

The image shows a Visual Studio Code (VS Code) interface with a terminal window open. The terminal is running a Python script named `reverse.py`. The script prompts the user to "Enter a string:" and the user has entered "madam". The output of the script is "Reversed string: madam". The terminal window is titled "reverse.py" and is located in the "TERMINAL" tab. The VS Code interface includes a menu bar at the top with options like File, Edit, Selection, View, Go, Run, and a search bar. The left sidebar shows the Explorer, Search, and Run and Debug views. The bottom status bar displays information about the current file, including line and column numbers, encoding, and the Python version (3.13.5).

```
PS C:\python> & "C:/Program Files/Python313/python.exe" c:/python/reverse.py
Enter a string: madam
Reversed string: madam
PS C:\python>
```


Task 4:

Iterative vs Recursive Factorial

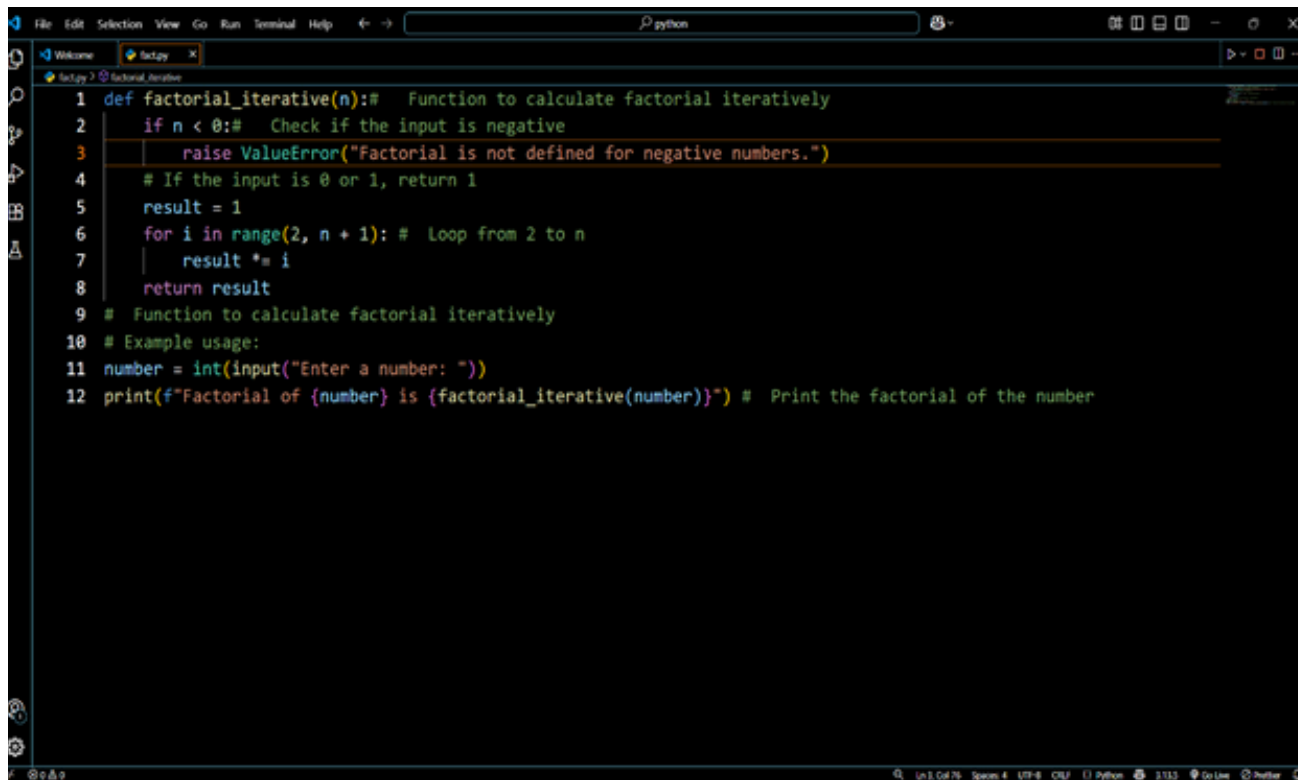
- Description: Prompt GitHub Copilot to generate both iterative and recursive versions of the factorial function.

Iterative function:

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

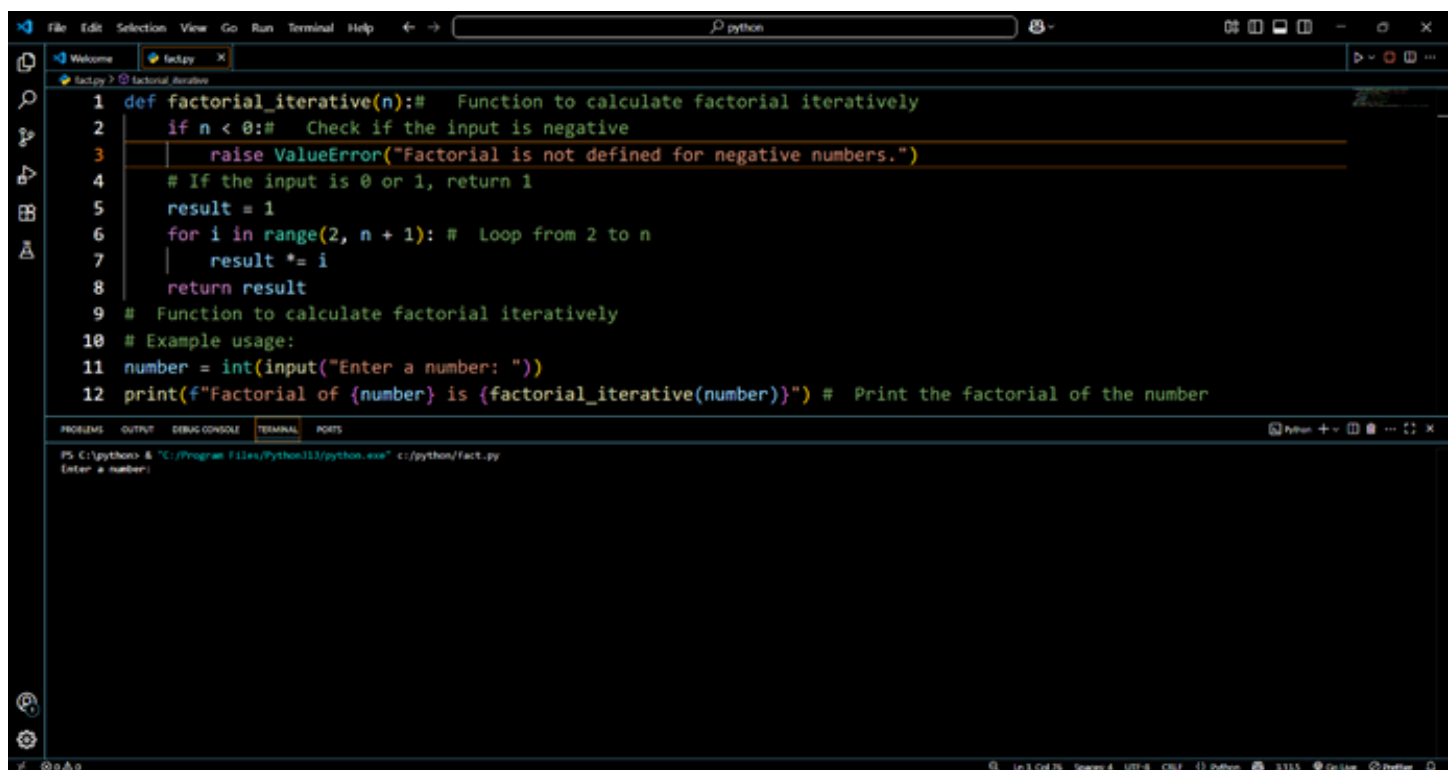


Code:



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

Asking for input:

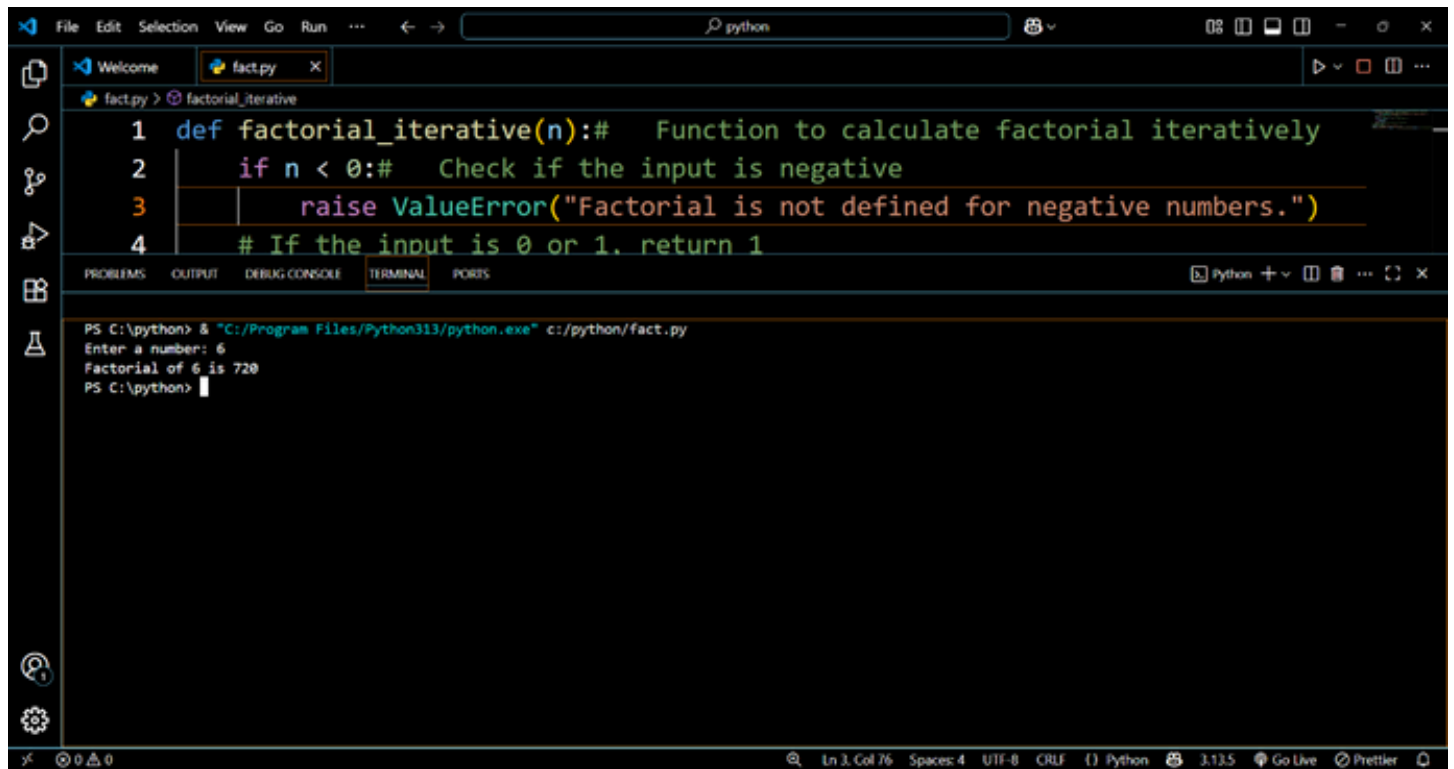


```
1 def factorial_iterative(n):# Function to calculate factorial iteratively
2     if n < 0:# Check if the input is negative
3         raise ValueError("Factorial is not defined for negative numbers.")
4     # If the input is 0 or 1, return 1
5     result = 1
6     for i in range(2, n + 1): # Loop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

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS C:\Python> & "C:/Program Files/Python13/python.exe" c:/python/Fact.py
(Enter a number:)

Output:



```
1 def factorial_iterative(n):# Function to calculate factorial iteratively
2     if n < 0:# Check if the input is negative
3         raise ValueError("Factorial is not defined for negative numbers.")
4     # If the input is 0 or 1. return 1
```

PS C:\python> & "C:/Program Files/Python313/python.exe" c:/python/fact.py
Enter a number: 6
Factorial of 6 is 720
PS C:\python> |

Recursive function for finding the Factorial of a number:

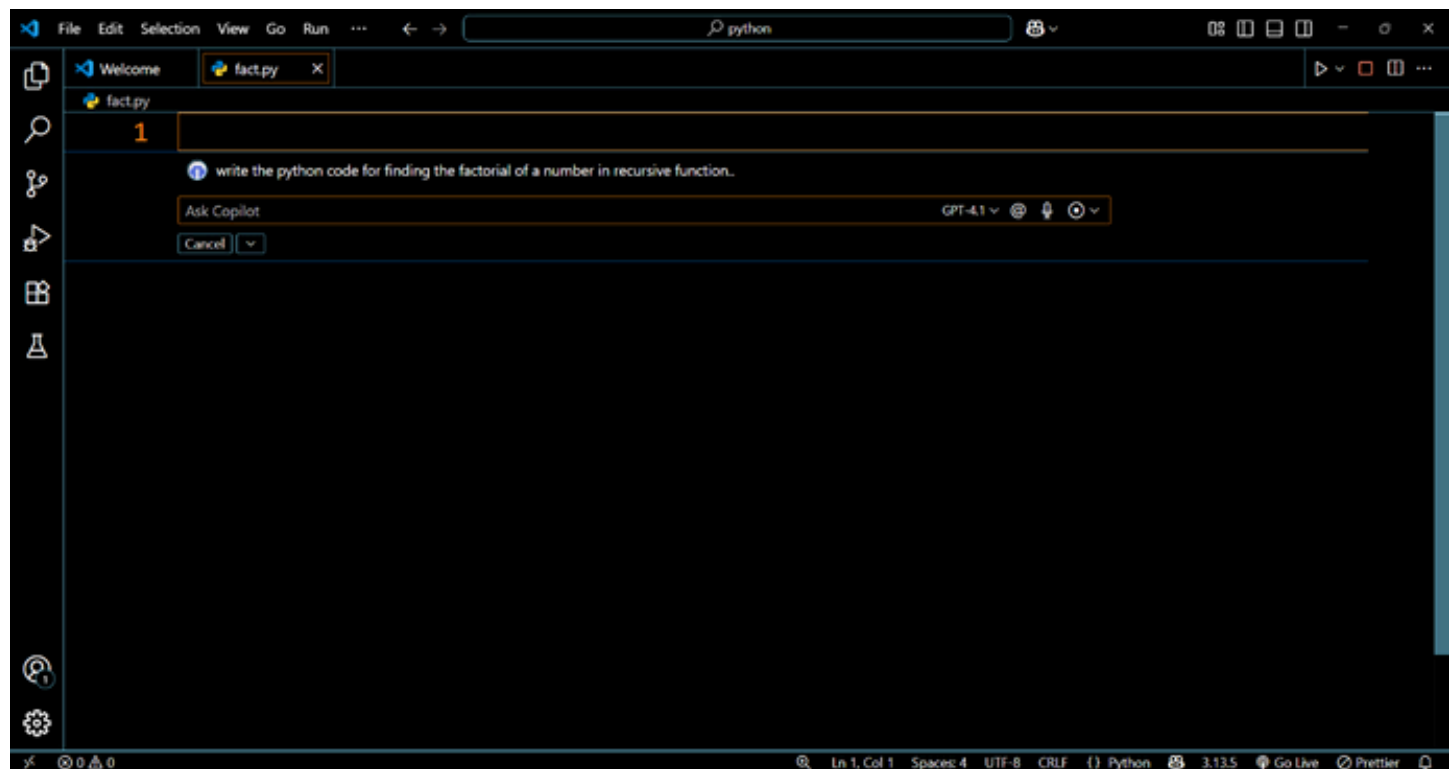
Prompt: write the python program for finding the factorial of a number in recursive function.



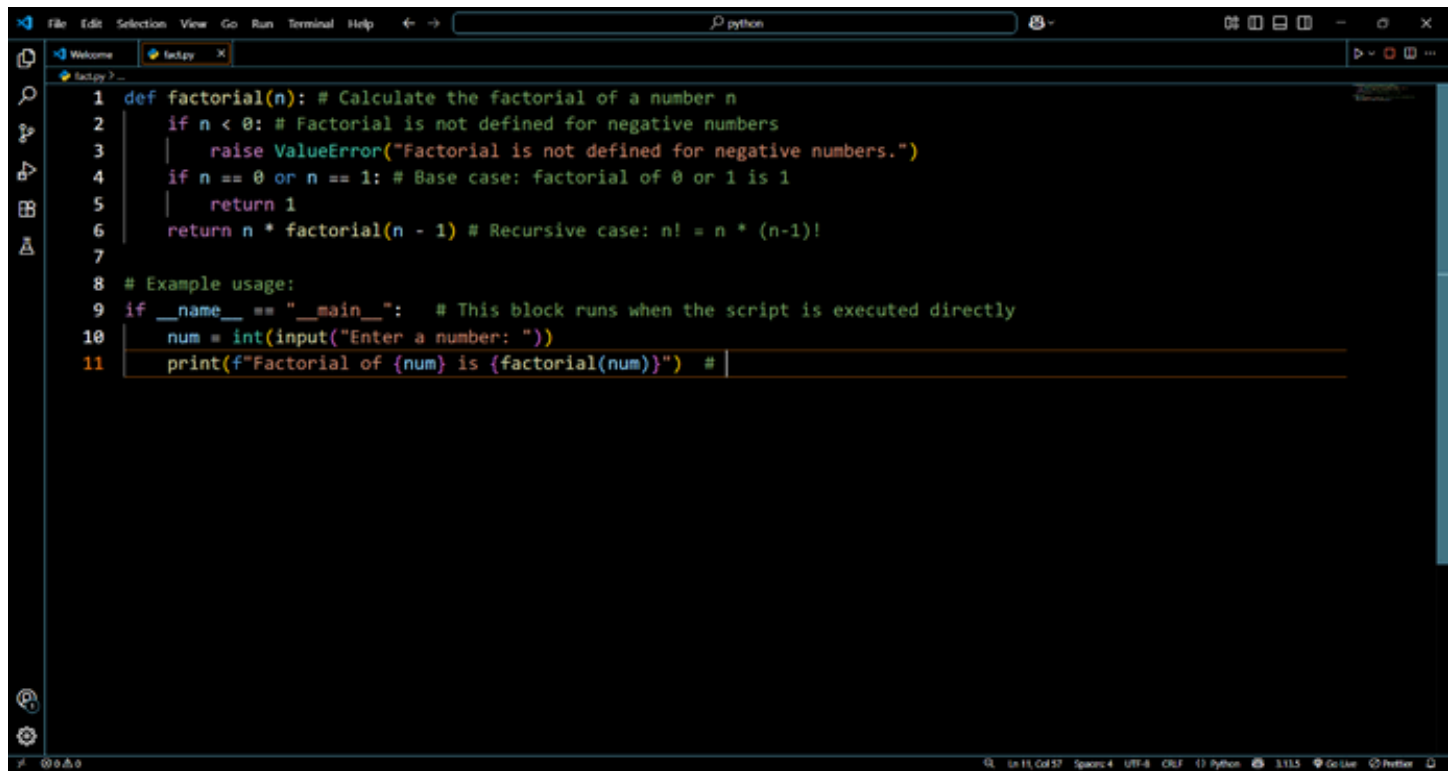
```
1
```

write the python code for finding the factorial of a number in recursive function

Copilot generating the Code:



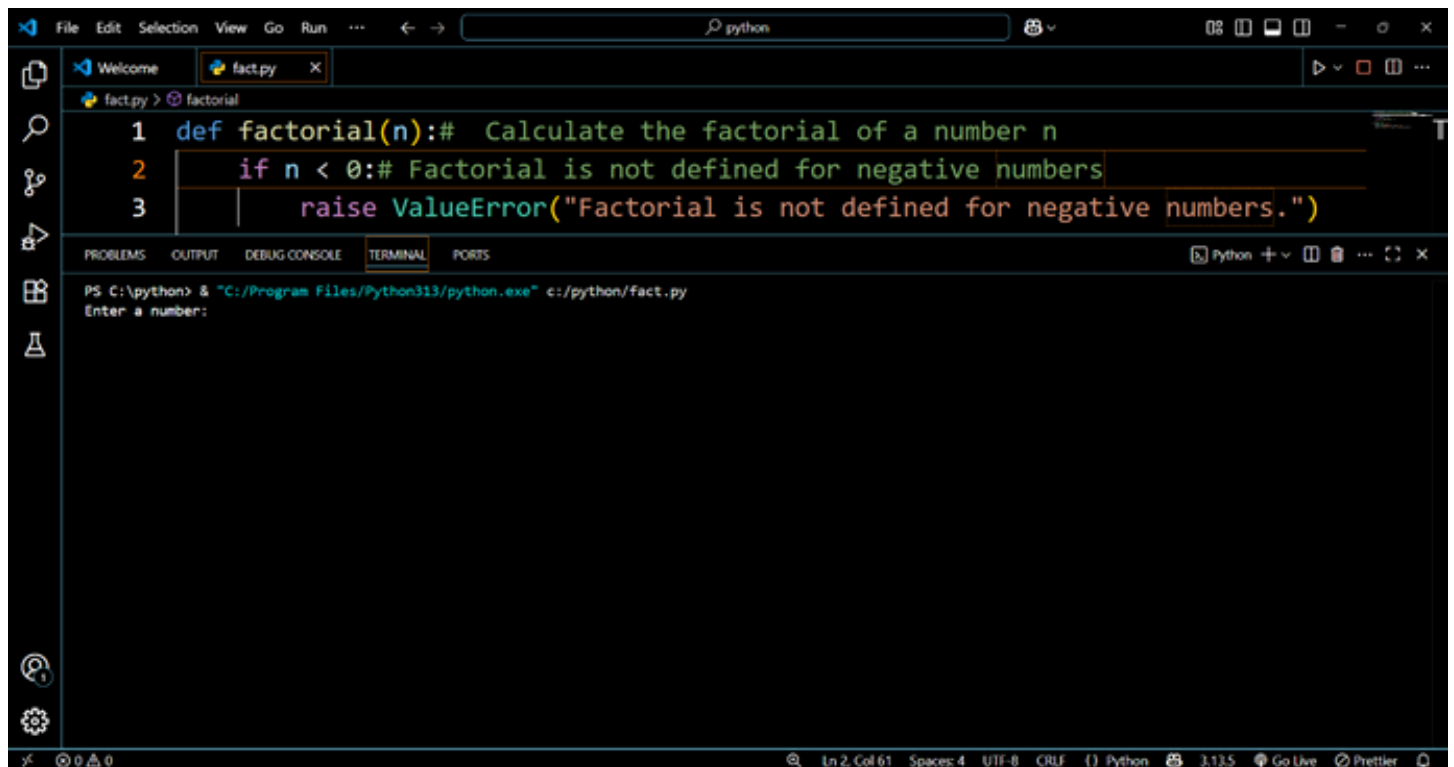
Code:



A screenshot of a code editor window titled 'python'. The editor shows a Python script for calculating the factorial of a number. The code is as follows:

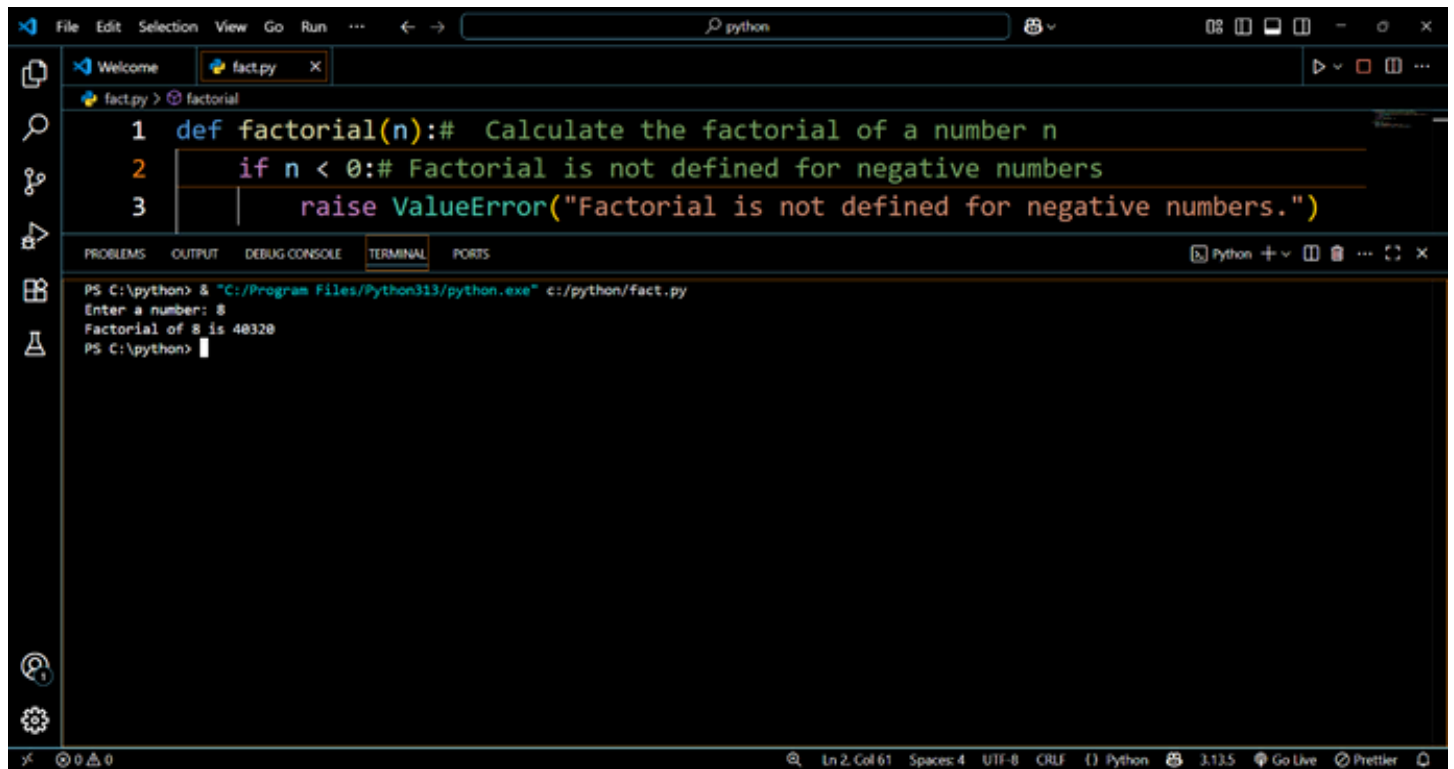
```
1 def factorial(n): # Calculate the factorial of a number n
2     if n < 0: # Factorial is not defined for negative numbers
3         raise ValueError("Factorial is not defined for negative numbers.")
4     if n == 0 or n == 1: # Base case: factorial of 0 or 1 is 1
5         return 1
6     return n * factorial(n - 1) # Recursive case: n! = n * (n-1)!
7
8 # Example usage:
9 if __name__ == "__main__": # This block runs when the script is executed directly
10     num = int(input("Enter a number: "))
11     print(f"Factorial of {num} is {factorial(num)}") #
```

Output:



A screenshot of the same code editor window, but now showing the execution of the script. The code is the same as in the previous image. The terminal window at the bottom shows the command prompt and the input/output of the program:

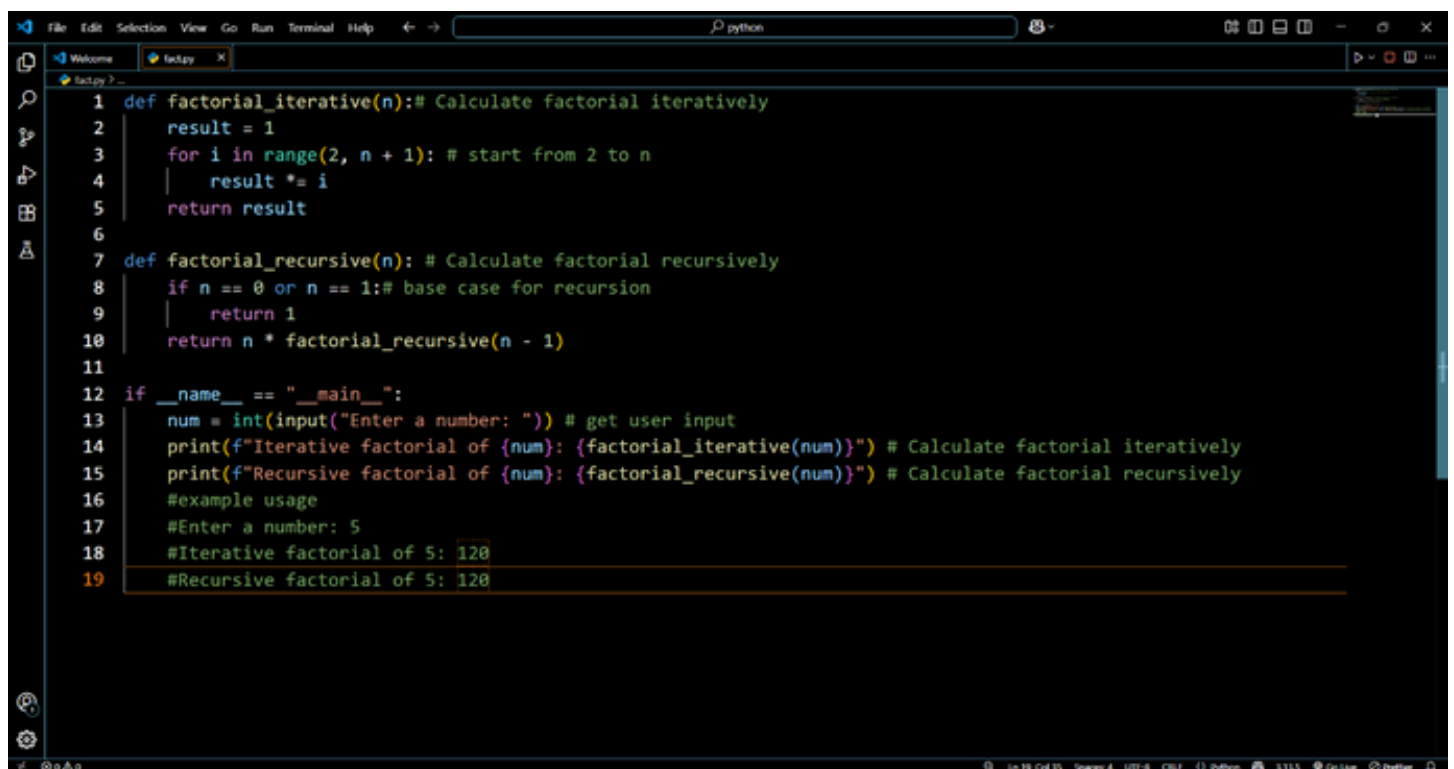
```
PS C:\python> & "C:/Program Files/Python313/python.exe" c:/python/fact.py
Enter a number:
```



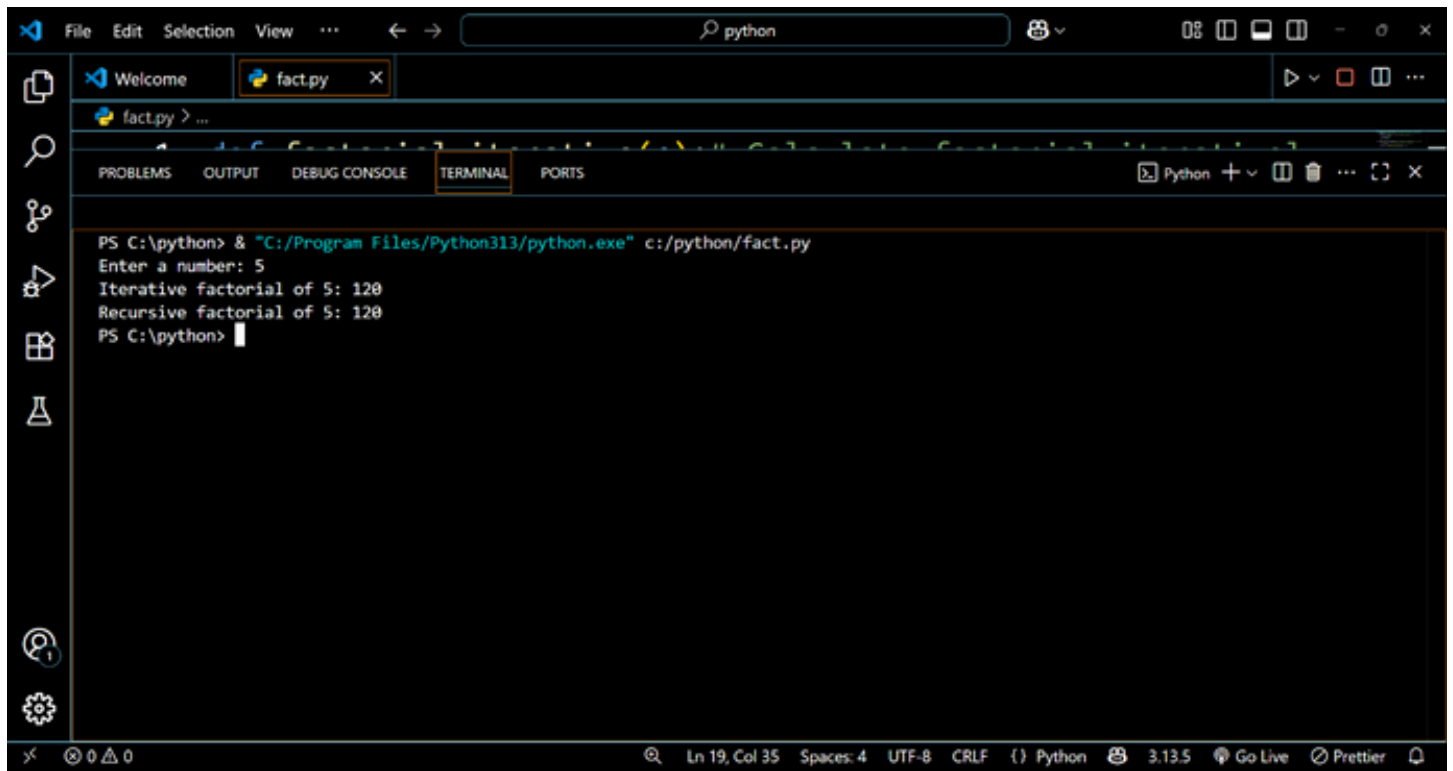
```
1 def factorial(n):# Calculate the factorial of a number n
2     if n < 0:# Factorial is not defined for negative numbers
3         raise ValueError("Factorial is not defined for negative numbers.")
```

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> |

FINAL CODE:



```
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
```



```
PS C:\python> & "C:/Program Files/Python313/python.exe" c:/python/fact.py
Enter a number: 5
Iterative factorial of 5: 120
Recursive factorial of 5: 120
PS C:\python>
```

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

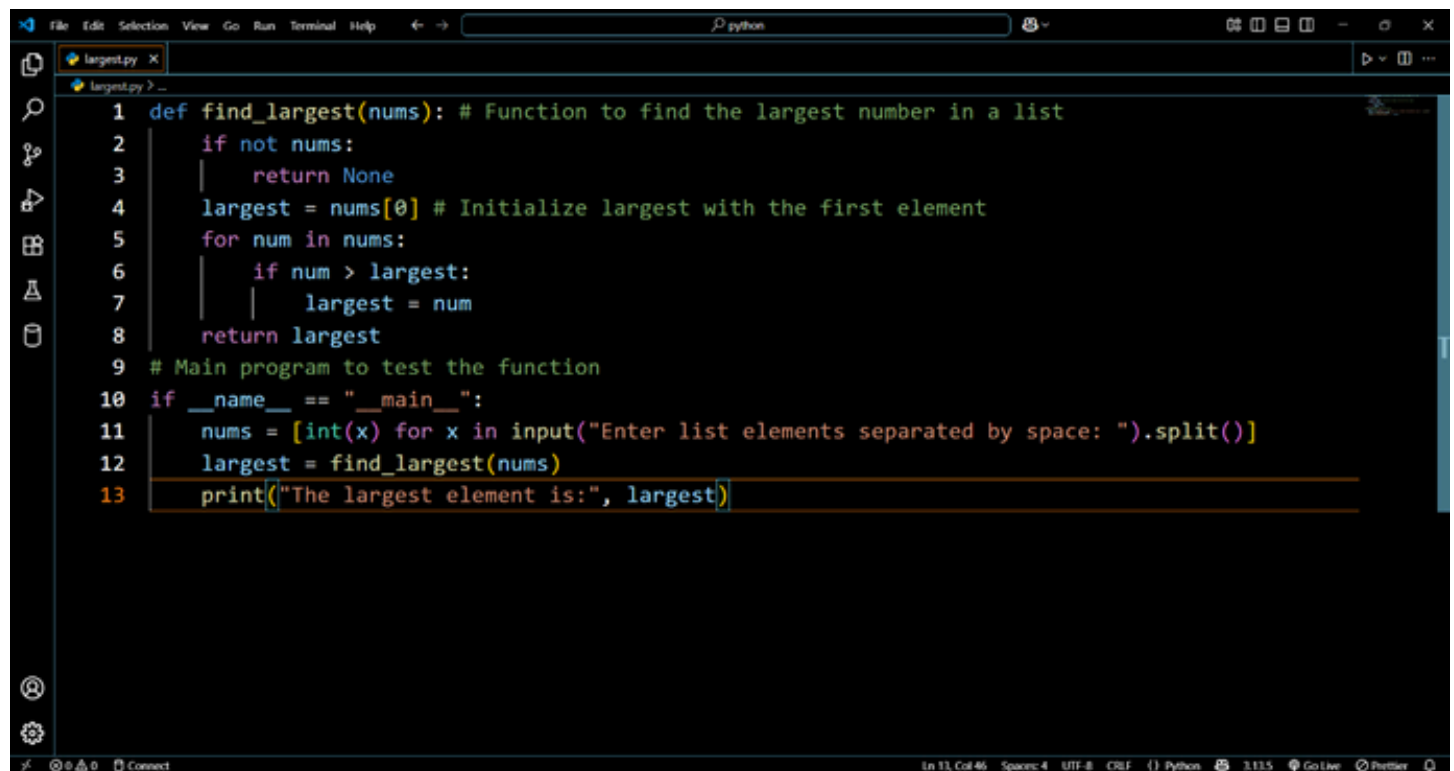
- **Iterative Flow:**

- 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.

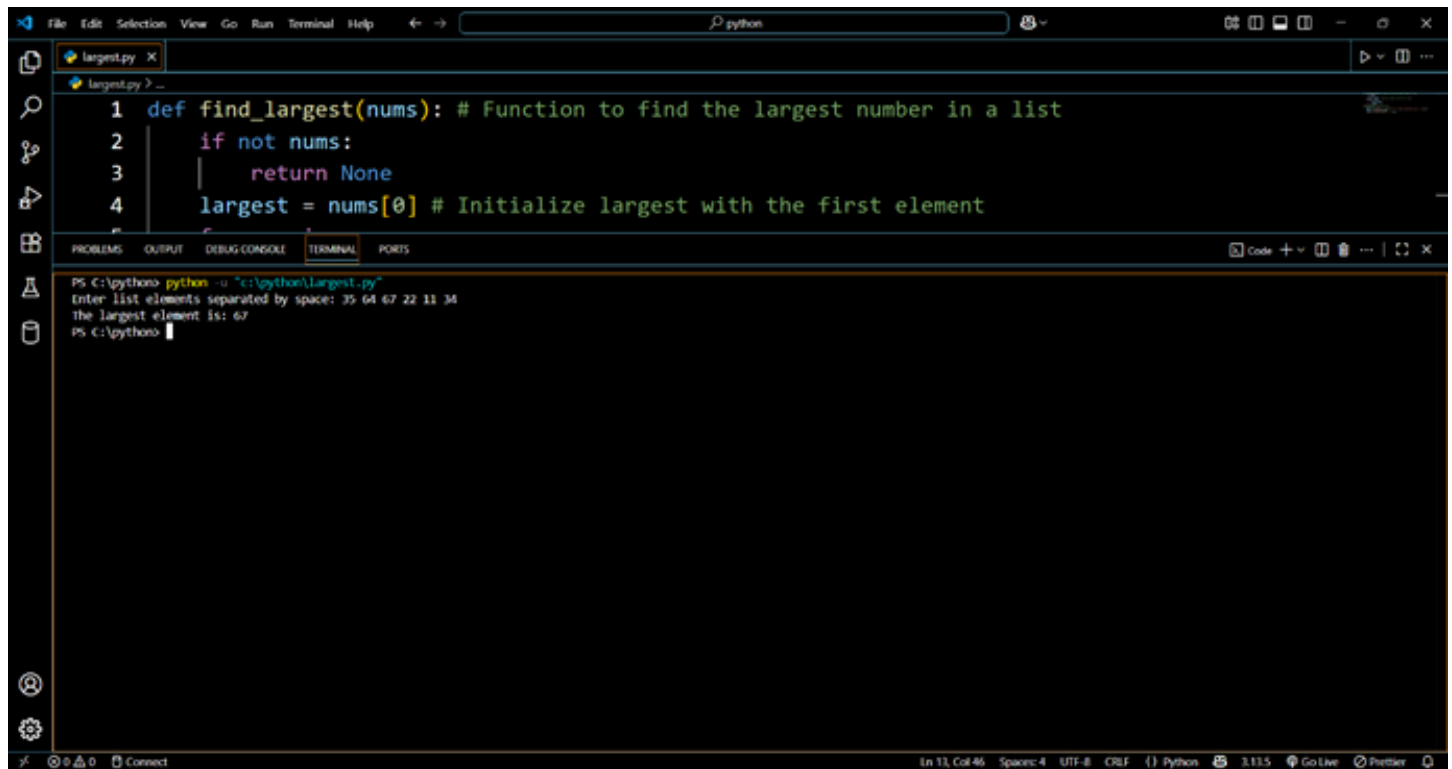
Prompt: write a python code to find the largest number in a list.

Code:

A screenshot of a Python IDE window titled 'largest.py'. The code defines a function 'find_largest' that takes a list 'nums' and returns the largest element. It uses a loop to iterate through the list and update the 'largest' variable. The main program prompts the user to enter list elements separated by space, splits the input into a list of integers, calls the 'find_largest' function, and prints the result. The code is as follows:

```
1 def find_largest(nums): # Function to find the largest number in a list
2     if not nums:
3         return None
4     largest = nums[0] # Initialize largest with the first element
5     for num in nums:
6         if num > largest:
7             largest = num
8     return largest
9 # Main program to test the function
10 if __name__ == "__main__":
11     nums = [int(x) for x in input("Enter list elements separated by space: ").split()]
12     largest = find_largest(nums)
13     print("The largest element is:", largest)
```

OUTPUT:



```
1 def find_largest(nums): # Function to find the largest number in a list
2     if not nums:
3         return None
4     largest = nums[0] # Initialize largest with the first element

PS C:\python> python -u "c:\python\largest.py"
Enter list elements separated by space: 35 64 67 22 11 34
The largest element is: 67
PS C:\python>
```

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.

