AI ASSISTED CODING

Assignment-1

2403A52107

Pavani Voddepalli

Batch : 24BTCAIAI05

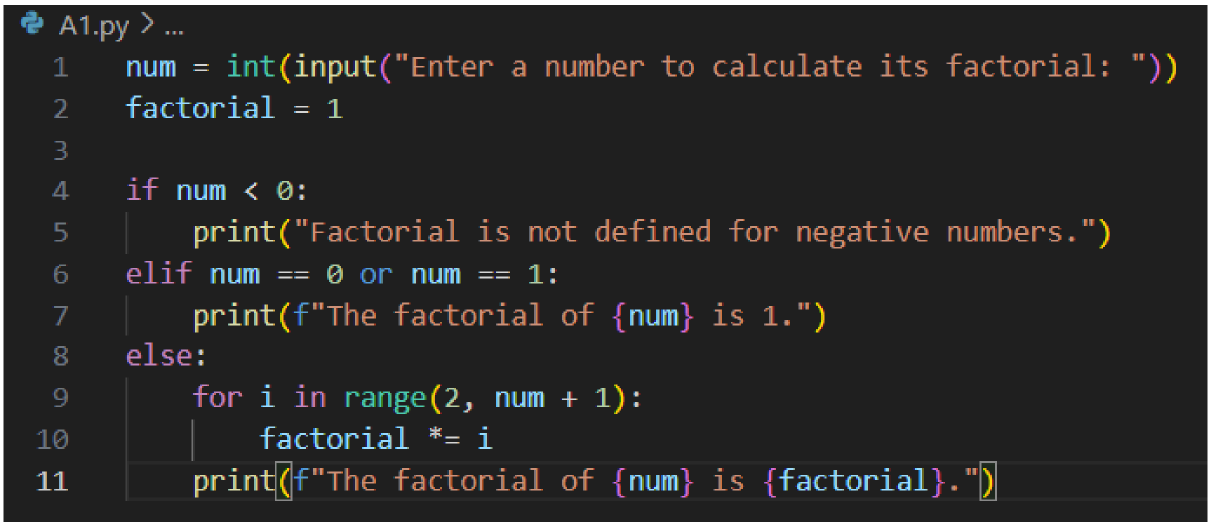
TASK 1: Factorial without Functions

- Description:
  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).
- Expected Output: ○ A working program that correctly calculates the factorial for user-provided input.

  ➢ *Screenshots of the code generation process.*

PROMPT : Generate a dynamic python program that calculates the factorial of a provided number using loops in the main code without defining any functions

CODE :

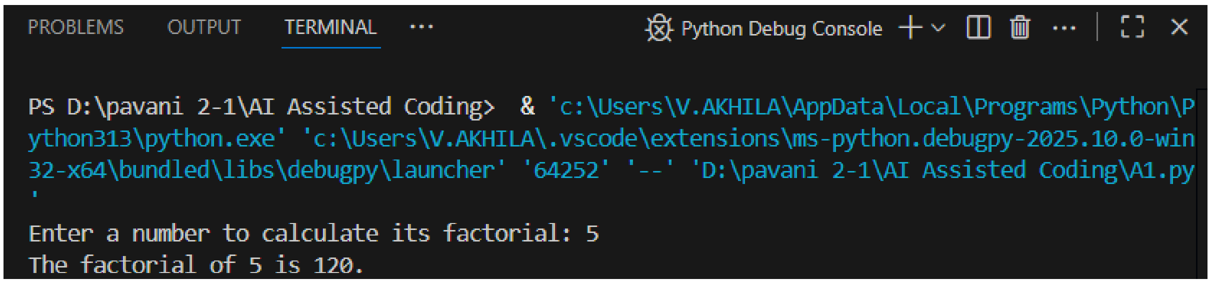


A1.py > ...

```python
num = int(input("Enter a number to calculate its factorial: "))
factorial = 1

if num < 0:
    print("Factorial is not defined for negative numbers.")
elif num == 0 or num == 1:
    print(f"The factorial of {num} is 1.")
else:
    for i in range(2, num + 1):
        factorial *= i
    print(f"The factorial of {num} is {factorial}.")
```

OUTPUT :



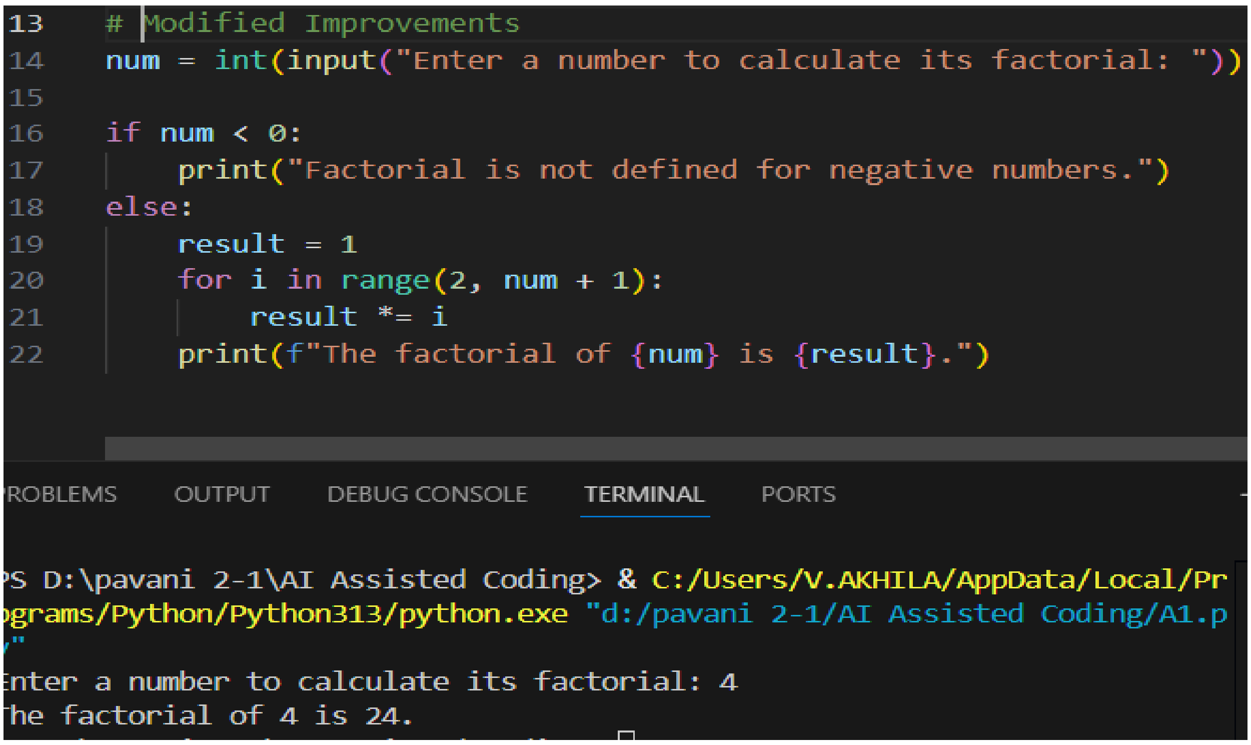
```
PS D:\pavani 2-1\AI Assisted Coding>  & 'c:\Users\V.AKHILA\AppData\Local\Programs\Python\Python313\python.exe' 'c:\Users\V.AKHILA\.vscode\extensions\ms-python.debugpy-2025.10.0-win32-x64\bundled\libs\debugpy\launcher' '64252' '--' 'D:\pavani 2-1\AI Assisted Coding\A1.py'

Enter a number to calculate its factorial: 5
The factorial of 5 is 120.
```

EXPLANATION:

This Python code calculates the factorial of a user-provided number. It begins by prompting the user to enter a number, which is then converted from a string to an integer using the int() function. The variable factorial is initialized to 1, which will be used to store the result of the factorial calculation.
The code then checks if the entered number is negative. Since factorials are not defined for negative numbers, it prints an appropriate message if this is the case. If the number is 0 or 1, the code prints that the factorial is 1, as both 0! and 1! are defined to be 1.
For numbers greater than 1, the code uses a for loop to multiply factorial by each integer from 2 up to and including the entered number. This loop effectively computes the product of all positive integers up to the given number, which is the definition of a factorial. Finally, it prints the result in a formatted string.

<u>TASK 2 :</u> Improving Efficiency
• Description:
Examine the Copilot-generated code from Task 1 and demonstrate how its efficiency can be improved (e.g., removing unnecessary variables, optimizing loops).
• Expected Output:
o Original and improved versions of the code.
o Explanation of how the improvements enhance performance.



```python
# Modified Improvements
num = int(input("Enter a number to calculate its factorial: "))

if num < 0:
    print("Factorial is not defined for negative numbers.")
else:
    result = 1
    for i in range(2, num + 1):
        result *= i
    print(f"The factorial of {num} is {result}.")
```

```
'S D:\pavani 2-1\AI Assisted Coding> & C:/Users/V.AKHILA/AppData/Local/Pr
ograms/Python/Python313/python.exe "d:/pavani 2-1/AI Assisted Coding/A1.p
y"
Enter a number to calculate its factorial: 4
The factorial of 4 is 24.
```

EXPLANATION of IMPROVEMENTS:

Removed Redundant Checks:
The check for num == 0 or num == 1 is unnecessary because the loop will not execute for these values, and result remains 1, which is correct for both 0! and 1!.
Variable Naming:
Renamed factorial to result for clarity, but this is optional.
Slightly Reduced Code Size:
The code is now more concise and easier to read, with fewer branches and variables.
Performance:
The computational efficiency is essentially the same (O(n)), but the code is cleaner and avoids unnecessary conditional checks, which can slightly improve runtime for small inputs and makes the code easier to maintain.

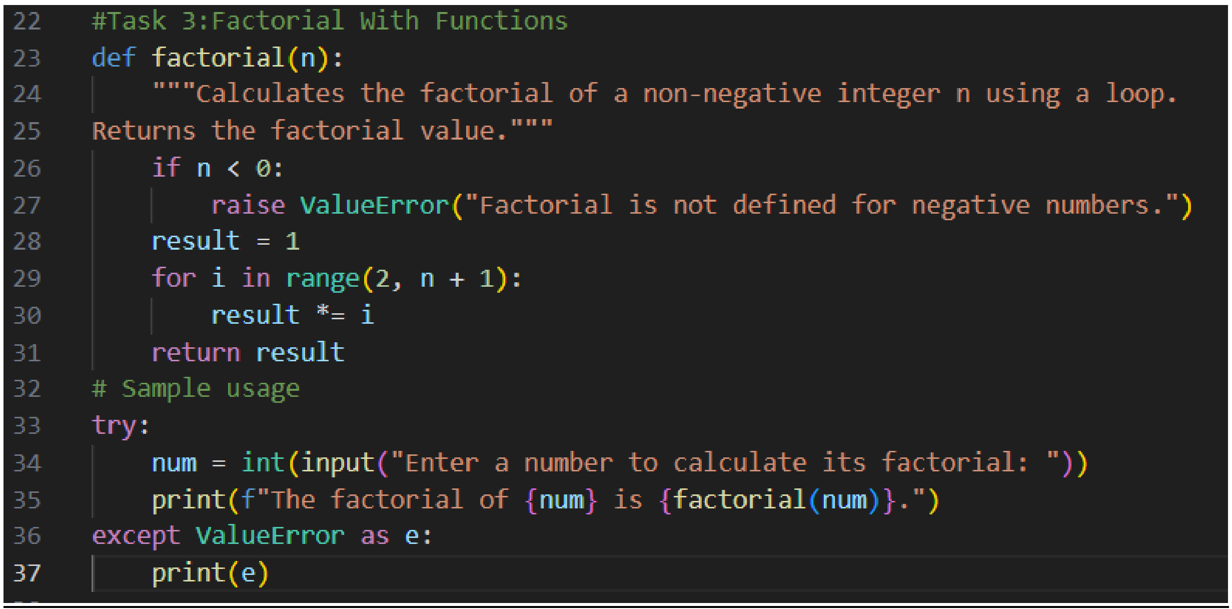
TASK 3 : Factorial with Functions

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

o Correctly working factorial function with sample outputs.
o Documentation of the steps Copilot followed to generate the function.

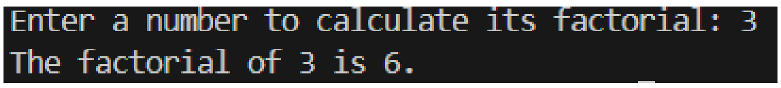
PROMPT : Generate a Python program that calculates the factorial of a number using a user-defined function.

CODE :



```python
#Task 3:Factorial With Functions
def factorial(n):
    """Calculates the factorial of a non-negative integer n using a loop.
Returns the factorial value."""
    if n < 0:
        raise ValueError("Factorial is not defined for negative numbers.")
    result = 1
    for i in range(2, n + 1):
        result *= i
    return result
# Sample usage
try:
    num = int(input("Enter a number to calculate its factorial: "))
    print(f"The factorial of {num} is {factorial(num)}.")
except ValueError as e:
    print(e)
```

OUTPUT :



```
Enter a number to calculate its factorial: 3
The factorial of 3 is 6.
```

DOCUMENTATION OF STEPS:

1. Defined a function `factorial(n)` that takes an integer n as input.
2. Checked if n is negative; if so, raised a ValueError.
3. Initialized result to 1.
4. Used a for loop to multiply result by each integer from 2 to n.
5. Returned the result.
6. In the main code, took user input, called the function, and displayed the result.
7. Added error handling for invalid input and negative numbers.

TASK 4 : Comparative Analysis – With vs Without Functions

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

• Expected Output: o A comparison table or short report explaining the differences.

Comparison of Factorial Programs: With vs. Without Functions

The factorial program without functions implements all logic directly in the main code block. This approach is straightforward but lacks reusability; if you need to calculate the factorial in another part of your program, you must duplicate the code. Error handling and testing are also less organized, making the code harder to maintain and extend.

On the other hand, the version with a user-defined function encapsulates the factorial logic inside a dedicated function. This makes the code modular and reusable, as the function can be called multiple times with different arguments. It also improves readability, allows for centralized error handling, and makes testing easier since the function can be tested independently.

In summary, using functions enhances code organization, reusability, and maintainability, while the non-function version is only suitable for simple, one-time calculations.

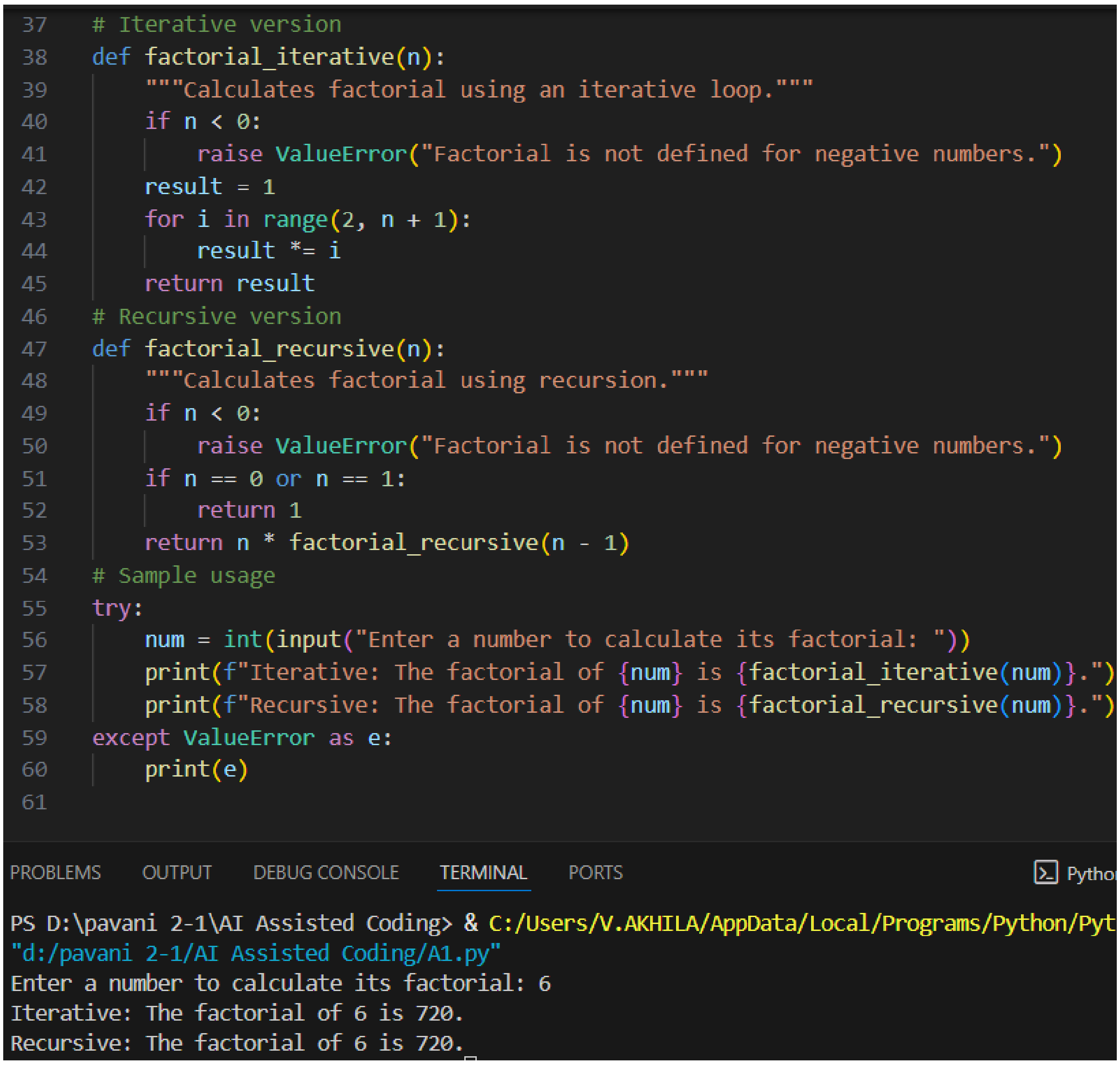
TASK 5 : Iterative vs Recursive Factorial

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

• Expected Output: o Two correct implementations.
o A documented comparison of logic, performance, and execution flow between iterative and recursive approaches.

PROMPT : **<u>Generate both iterative and recursive versions of the factorial function.</u>**

CODE WITH OUTPUT :



```python
# Iterative version
def factorial_iterative(n):
    """Calculates factorial using an iterative loop."""
    if n < 0:
        raise ValueError("Factorial is not defined for negative numbers.")
    result = 1
    for i in range(2, n + 1):
        result *= i
    return result
# Recursive version
def factorial_recursive(n):
    """Calculates factorial using recursion."""
    if n < 0:
        raise ValueError("Factorial is not defined for negative numbers.")
    if n == 0 or n == 1:
        return 1
    return n * factorial_recursive(n - 1)
# Sample usage
try:
    num = int(input("Enter a number to calculate its factorial: "))
    print(f"Iterative: The factorial of {num} is {factorial_iterative(num)}.")
    print(f"Recursive: The factorial of {num} is {factorial_recursive(num)}.")
except ValueError as e:
    print(e)
```

```
PS D:\pavani 2-1\AI Assisted Coding> & C:/Users/V.AKHILA/AppData/Local/Programs/Python/Pyt
"d:/pavani 2-1/AI Assisted Coding/A1.py"
Enter a number to calculate its factorial: 6
Iterative: The factorial of 6 is 720.
Recursive: The factorial of 6 is 720.
```

Comparison:

<u>Logic:</u>
- Iterative: Uses a loop to multiply numbers from 2 to n, accumulating the result.
- Recursive: Calls itself with (n-1) until reaching the base case (n==0 or n==1).

<u>Performance:</u>
- Iterative: Uses constant memory (O(1)), efficient for large n.
- Recursive: Uses O(n) stack space due to function calls, may hit recursion limits for large n.

<u>Execution Flow:</u>
- Iterative: Single loop, straightforward execution.
- Recursive: Multiple function calls, each waiting for the next to return, which can be less efficient and harder to debug for large n.