AI Assisted Coding

ASSIGNMENT 1.2

Name: PALAKURTHI LALITH PRAKASH

HT No: 2303A52012

Batch: 31

Task 1: AI-Generated Logic Without Modularization (Factorial without Functions)

- Scenario

You are building a small command-line utility for a startup intern onboarding task. The program is simple and must be written quickly without modular design.

- Task Description

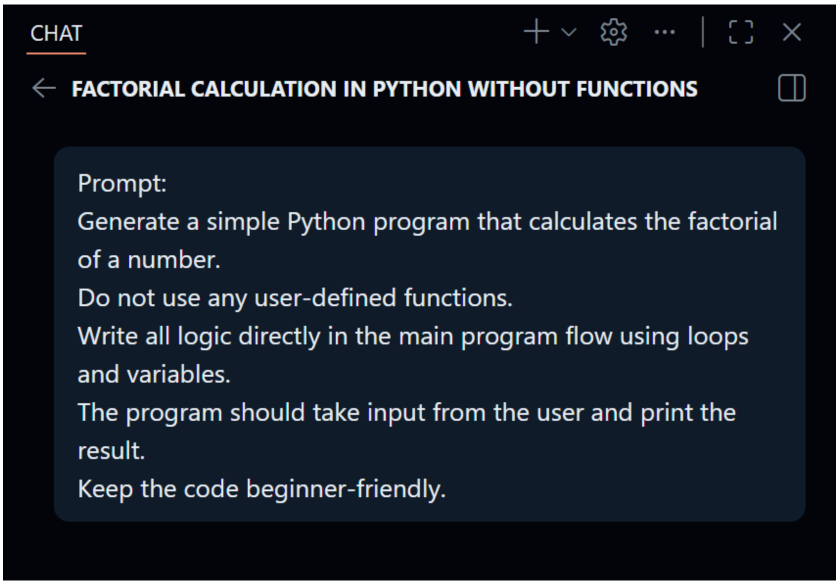
Use GitHub Copilot to generate a Python program that computes a mathematical product-based value (factorial-like logic) directly in the main execution flow, without using any user-defined functions.

- Constraint:

➢ Do not define any custom function

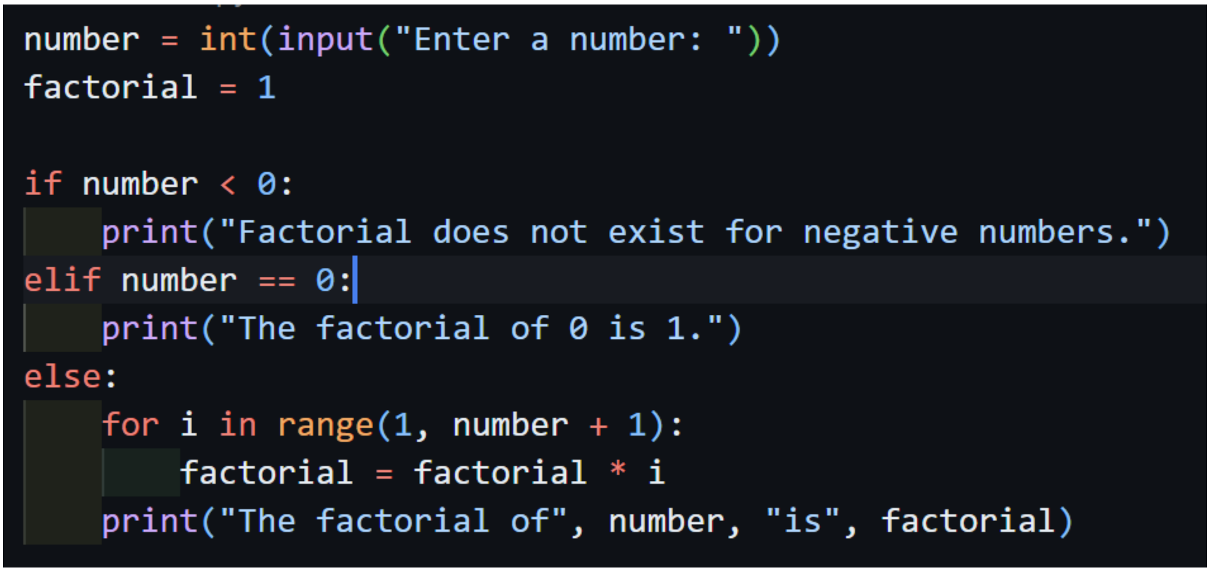
➢ Logic must be implemented using loops and variables only.

Prompt:



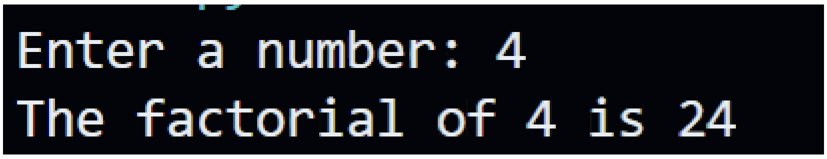
Prompt:
Generate a simple Python program that calculates the factorial of a number.
Do not use any user-defined functions.
Write all logic directly in the main program flow using loops and variables.
The program should take input from the user and print the result.
Keep the code beginner-friendly.

code:



```
number = int(input("Enter a number: "))
factorial = 1

if number < 0:
    print("Factorial does not exist for negative numbers.")
elif number == 0:
    print("The factorial of 0 is 1.")
else:
    for i in range(1, number + 1):
        factorial = factorial * i
    print("The factorial of", number, "is", factorial)
```

output:



```
Enter a number: 4
The factorial of 4 is 24
```

Task 2: AI Code Optimization & Cleanup (Improving Efficiency)

❖ Scenario

Your team lead asks you to review AI-generated code before committing it to

a shared repository.

❖ Task Description

Analyze the code generated in Task 1 and use Copilot again to:

➢ Reduce unnecessary variables

➢ Improve loop clarity

➢ Enhance readability and efficiency

Hint:

Prompt Copilot with phrases like

“optimize this code”, “simplify logic”, or “make it more readable”

❖ Scenario:

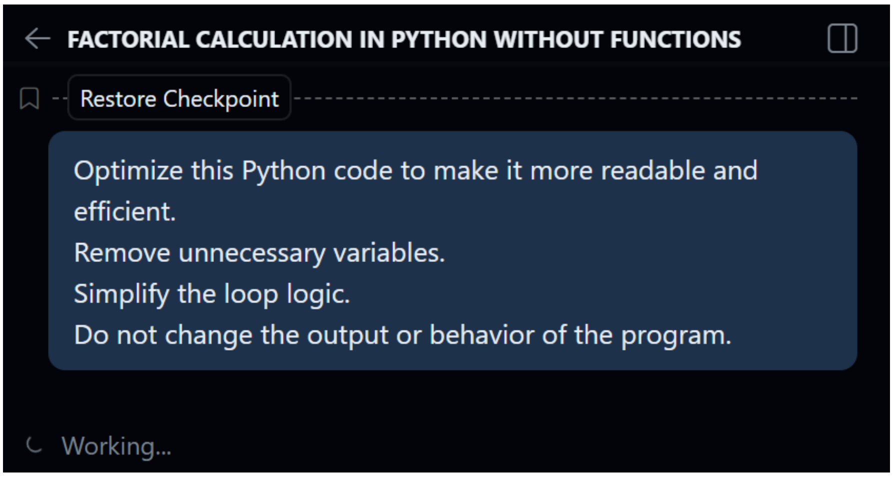
You are automating basic file operations.

❖ Task:

Use Cursor AI to generate a program that:

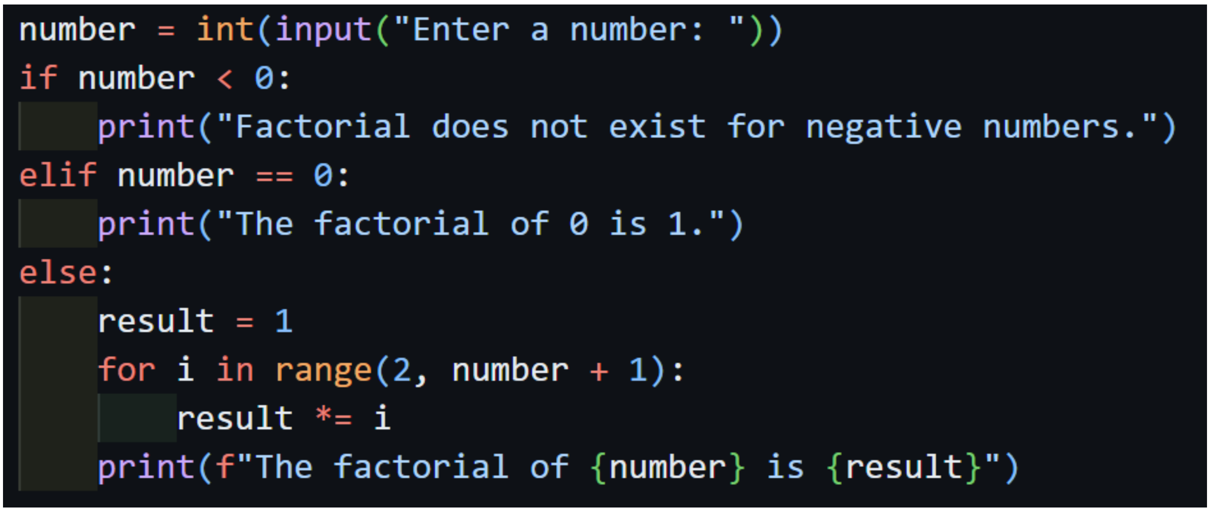
- ➢ Creates a text file
- ➢ Writes sample text
- ➢ Reads and displays the content

Prompt:



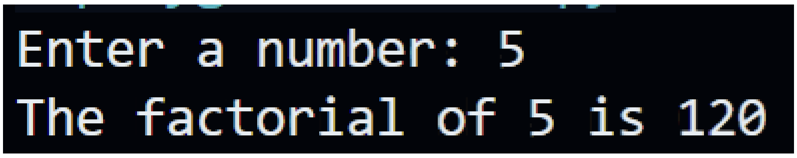
Optimize this Python code to make it more readable and efficient.
Remove unnecessary variables.
Simplify the loop logic.
Do not change the output or behavior of the program.

code:



```python
number = int(input("Enter a number: "))
if number < 0:
    print("Factorial does not exist for negative numbers.")
elif number == 0:
    print("The factorial of 0 is 1.")
else:
    result = 1
    for i in range(2, number + 1):
        result *= i
    print(f"The factorial of {number} is {result}")
```

output:



```
Enter a number: 5
The factorial of 5 is 120
```

Task 3: Modular Design Using AI Assistance (Factorial with Functions)

❖ Scenario

The same logic now needs to be reused in multiple scripts.

❖ Task Description

Use GitHub Copilot to generate a modular version of the program by:

➢ Creating a user-defined function

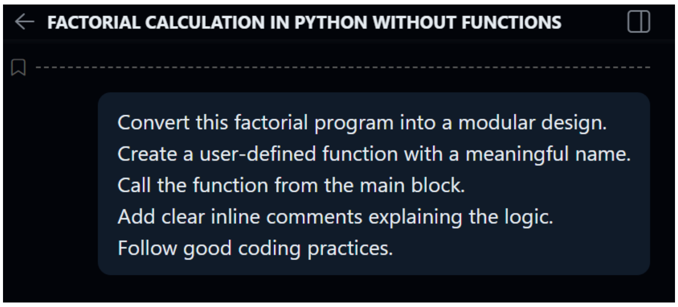
➢ Calling the function from the main block

❖ Constraints

➢ Use meaningful function and variable names

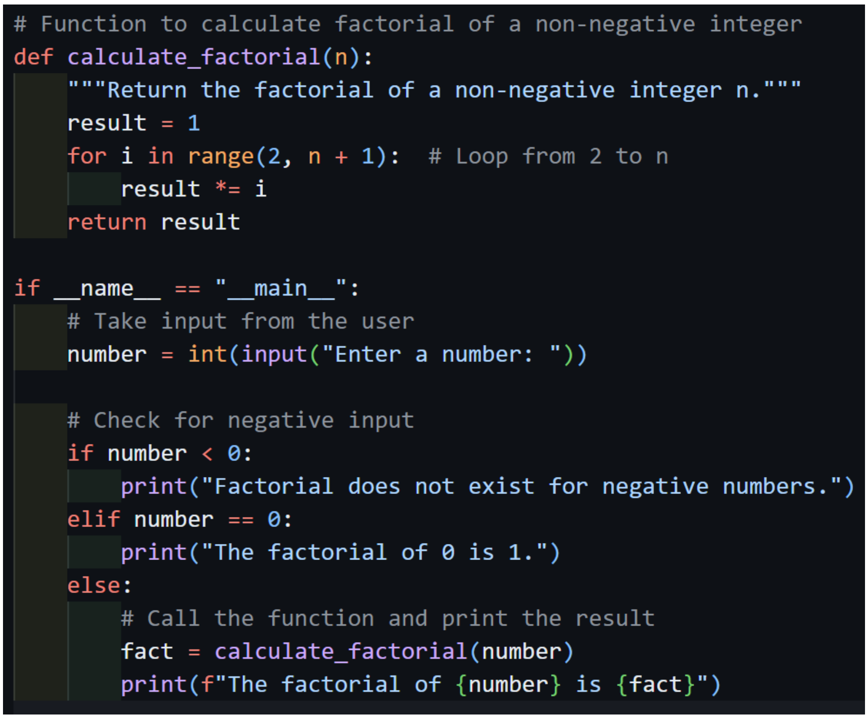
➢ Include inline comments (preferably suggested by Copilot)

Prompt:



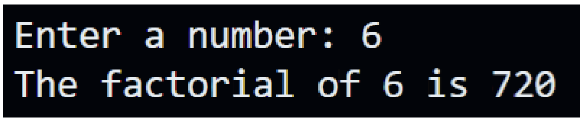
Convert this factorial program into a modular design.
Create a user-defined function with a meaningful name.
Call the function from the main block.
Add clear inline comments explaining the logic.
Follow good coding practices.

code:



```python
# Function to calculate factorial of a non-negative integer
def calculate_factorial(n):
    """Return the factorial of a non-negative integer n."""
    result = 1
    for i in range(2, n + 1):  # Loop from 2 to n
        result *= i
    return result

if __name__ == "__main__":
    # Take input from the user
    number = int(input("Enter a number: "))

    # Check for negative input
    if number < 0:
        print("Factorial does not exist for negative numbers.")
    elif number == 0:
        print("The factorial of 0 is 1.")
    else:
        # Call the function and print the result
        fact = calculate_factorial(number)
        print(f"The factorial of {number} is {fact}")
```

output:



```
Enter a number: 6
The factorial of 6 is 720
```

Task 4: Comparative Analysis – Procedural vs Modular AI Code (With vs Without Functions)

❖ Scenario

As part of a code review meeting, you are asked to justify design choices.

❖ Task Description

Compare the non-function and function-based Copilot-generated

programs on the following criteria:

➢ Logic clarity

➢ Reusability

➢ Debugging ease

➢ Suitability for large projects

➢ AI dependency risk

justification:

In the non-function approach, the code is okay for small programs and quick work. It is easy to write and understand in the beginning, especially for beginners. But when the program becomes bigger, everything is written in one place, so it becomes hard to manage and understand.

In the function-based approach, the logic is separated properly. This makes the code look clean and well-structured. If the same logic is needed again, we can reuse the function instead of writing the code again. From a review point of view, this approach is better because it makes the code easy to read, debug, and improve in the future.

Task 5: AI-Generated Iterative vs Recursive Thinking

❖ Scenario

Your mentor wants to test how well AI understands different

computational paradigms.

❖ Task Description

Prompt Copilot to generate:

An iterative version of the logic

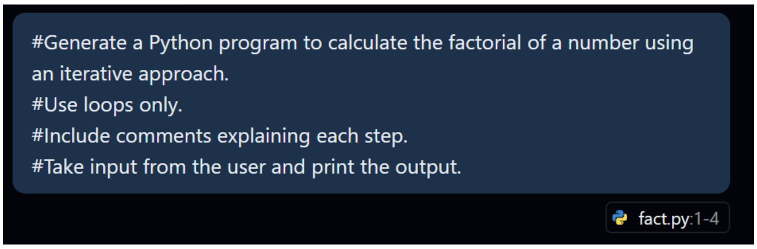
A recursive version of the same logic

❖ Constraints

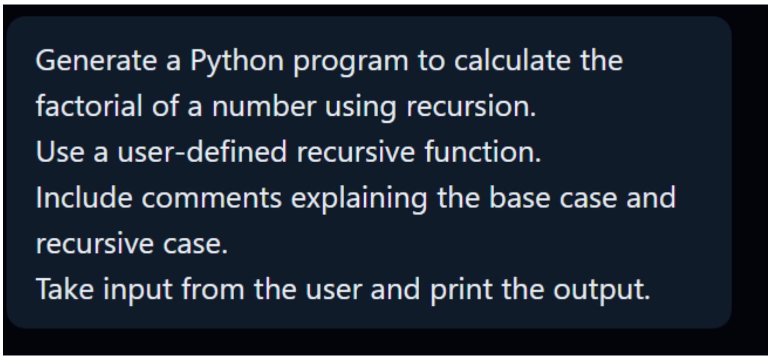
Both implementations must produce identical outputs

Students must not manually write the code first

prompts:

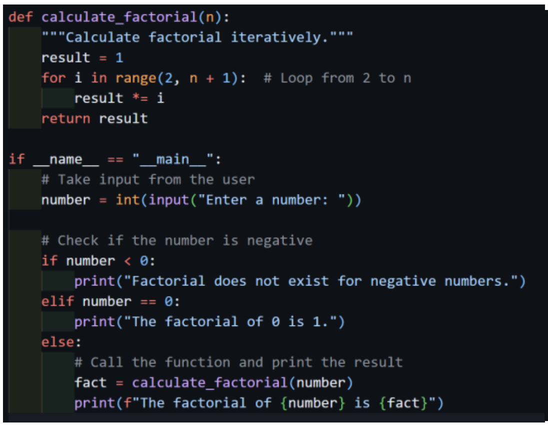


#Generate a Python program to calculate the factorial of a number using an iterative approach.
#Use loops only.
#Include comments explaining each step.
#Take input from the user and print the output.

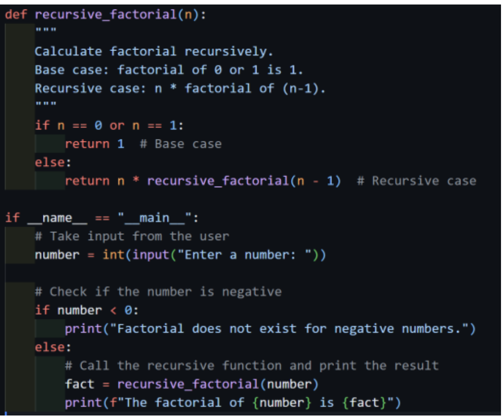


Generate a Python program to calculate the factorial of a number using recursion.
Use a user-defined recursive function.
Include comments explaining the base case and recursive case.
Take input from the user and print the output.

codes:

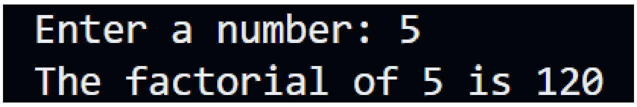


```python
def calculate_factorial(n):
    """Calculate factorial iteratively."""
    result = 1
    for i in range(2, n + 1):  # Loop from 2 to n
        result *= i
    return result

if __name__ == "__main__":
    # Take input from the user
    number = int(input("Enter a number: "))

    # Check if the number is negative
    if number < 0:
        print("Factorial does not exist for negative numbers.")
    elif number == 0:
        print("The factorial of 0 is 1.")
    else:
        # Call the function and print the result
        fact = calculate_factorial(number)
        print(f"The factorial of {number} is {fact}")
```

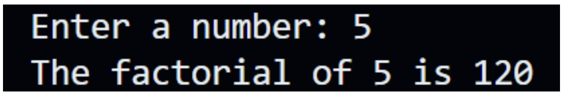


```python
def recursive_factorial(n):
    """
    Calculate factorial recursively.
    Base case: factorial of 0 or 1 is 1.
    Recursive case: n * factorial of (n-1).
    """
    if n == 0 or n == 1:
        return 1  # Base case
    else:
        return n * recursive_factorial(n - 1)  # Recursive case

if __name__ == "__main__":
    # Take input from the user
    number = int(input("Enter a number: "))

    # Check if the number is negative
    if number < 0:
        print("Factorial does not exist for negative numbers.")
    else:
        # Call the recursive function and print the result
        fact = recursive_factorial(number)
        print(f"The factorial of {number} is {fact}")
```

outputs:



```
Enter a number: 5
The factorial of 5 is 120
```



```
Enter a number: 5
The factorial of 5 is 120
```

justification:

Both iterative and recursive methods give the same factorial output, but the way they work is different. The iterative approach is easy to understand because it runs step by step using a loop. It is simple, clear, and works well even for large numbers without any risk.

The recursive approach is useful to understand how a problem can be broken into smaller parts. However, it is a little harder to follow because the function keeps calling itself. It also uses more memory and can cause issues if the input number is very large.

Because of this, the iterative method is better for real applications, while recursion is mainly helpful for learning and understanding concepts.