# AI ASSISTED CODING

# 1st Assignment :

## TASK-1:

Factorial without Functions

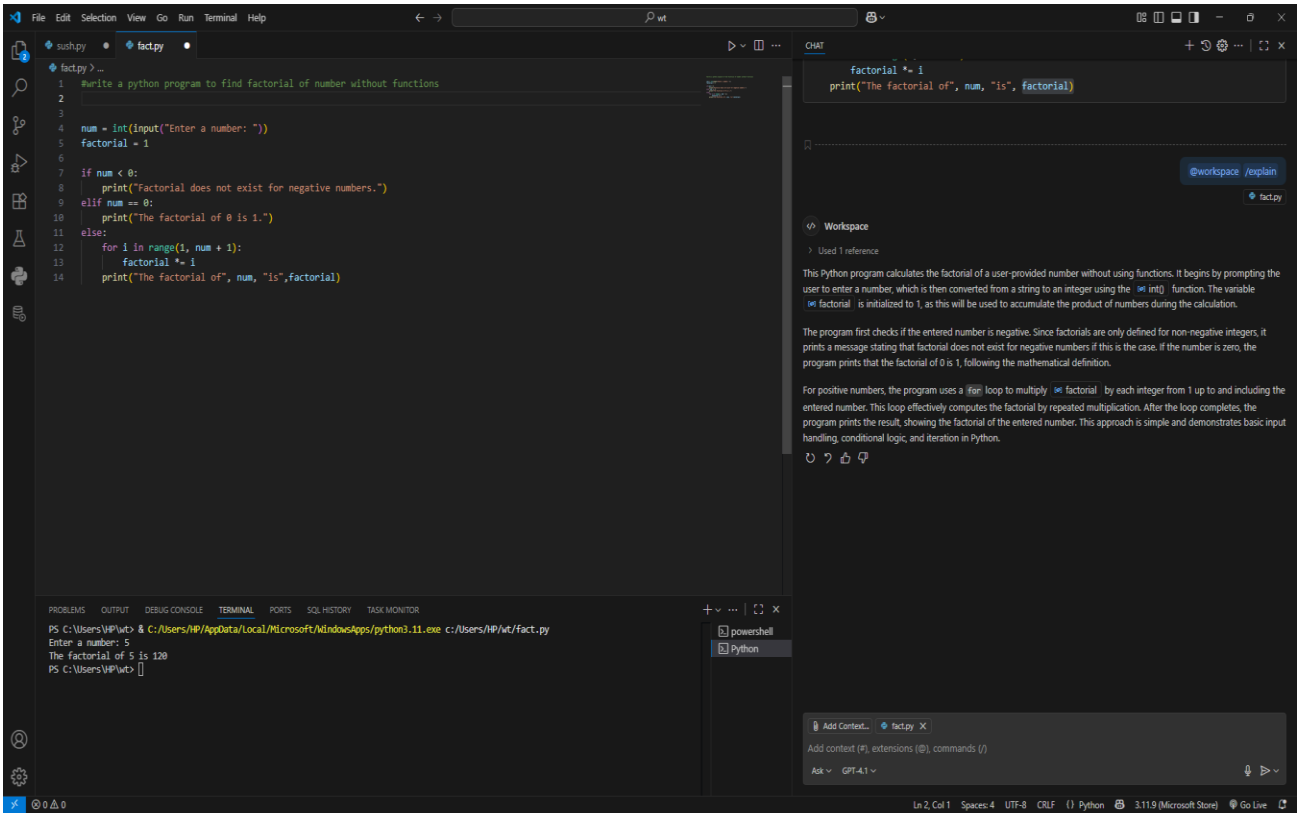
• Description:

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:

o A working program that correctly calculates the factorial for user-provided input.

o Screenshots of the code generation process




# TASK-2:

Improving Efficiency

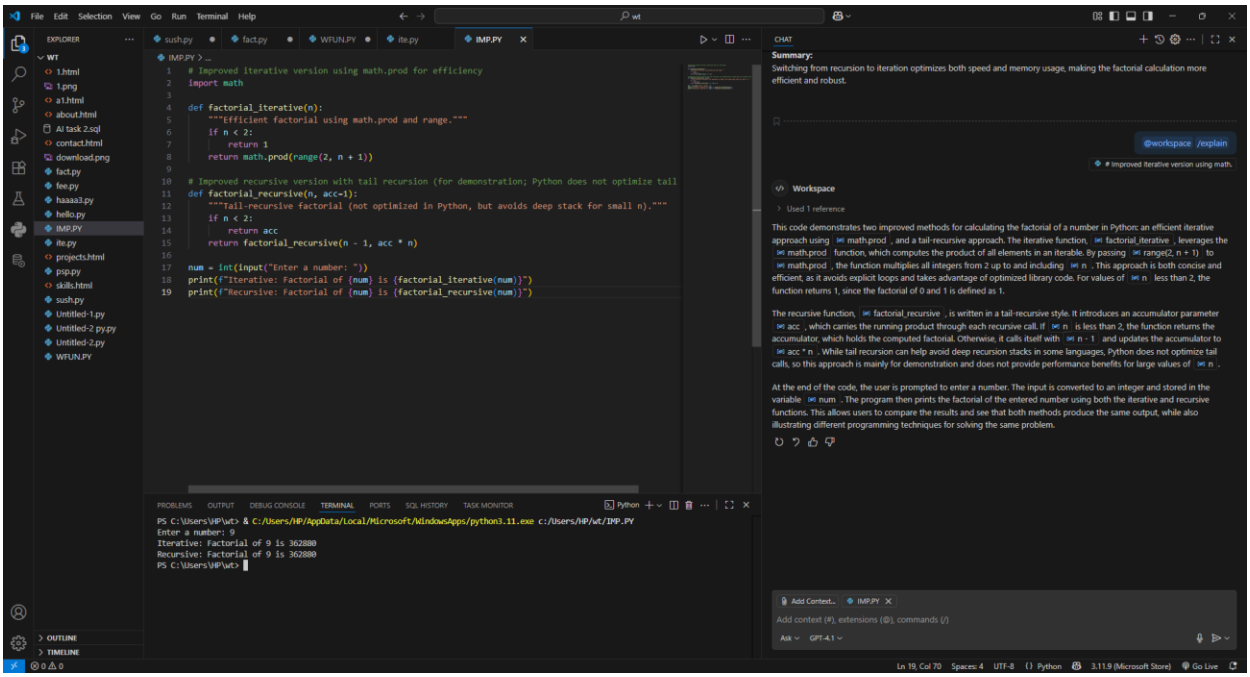
• Description:

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.




# TASK-3:

Factorial with Functions

• Description:

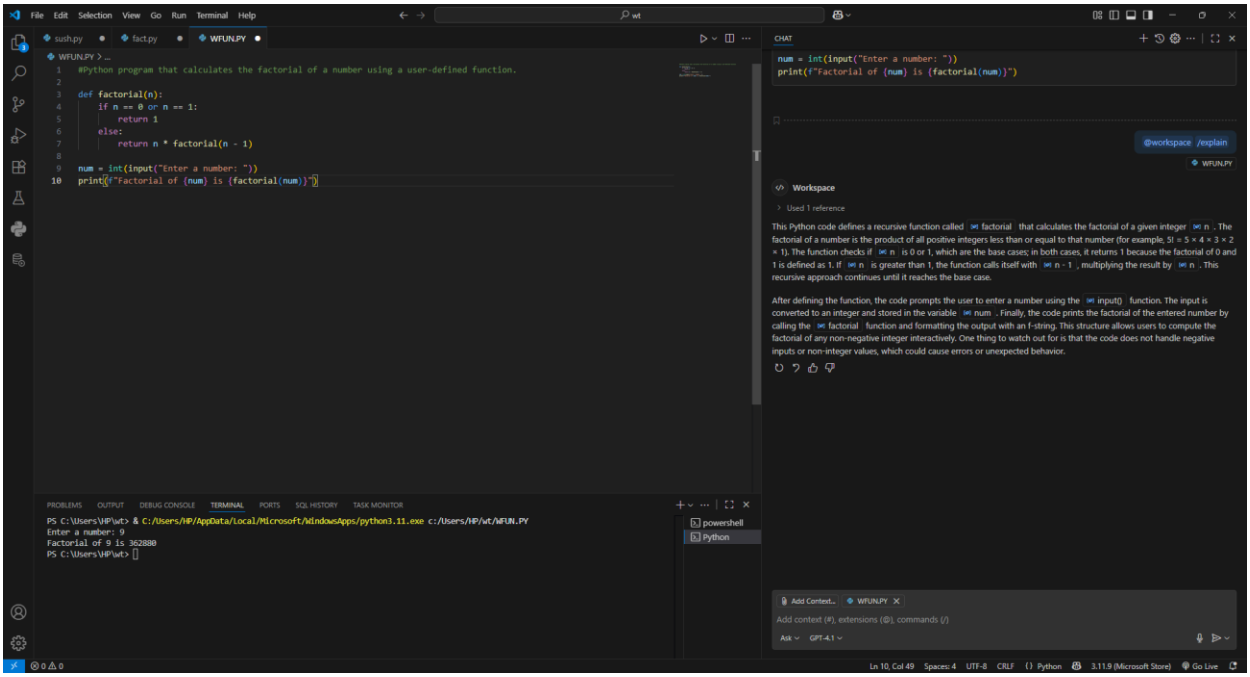
Use GitHub Copilot to generate a Python program that calculates the factorial of a number using a user-defined function.

• Expected Output:

o Correctly working factorial function with sample outputs.

o Documentation of the steps Copilot followed to generate the

function.




# TASK-4:

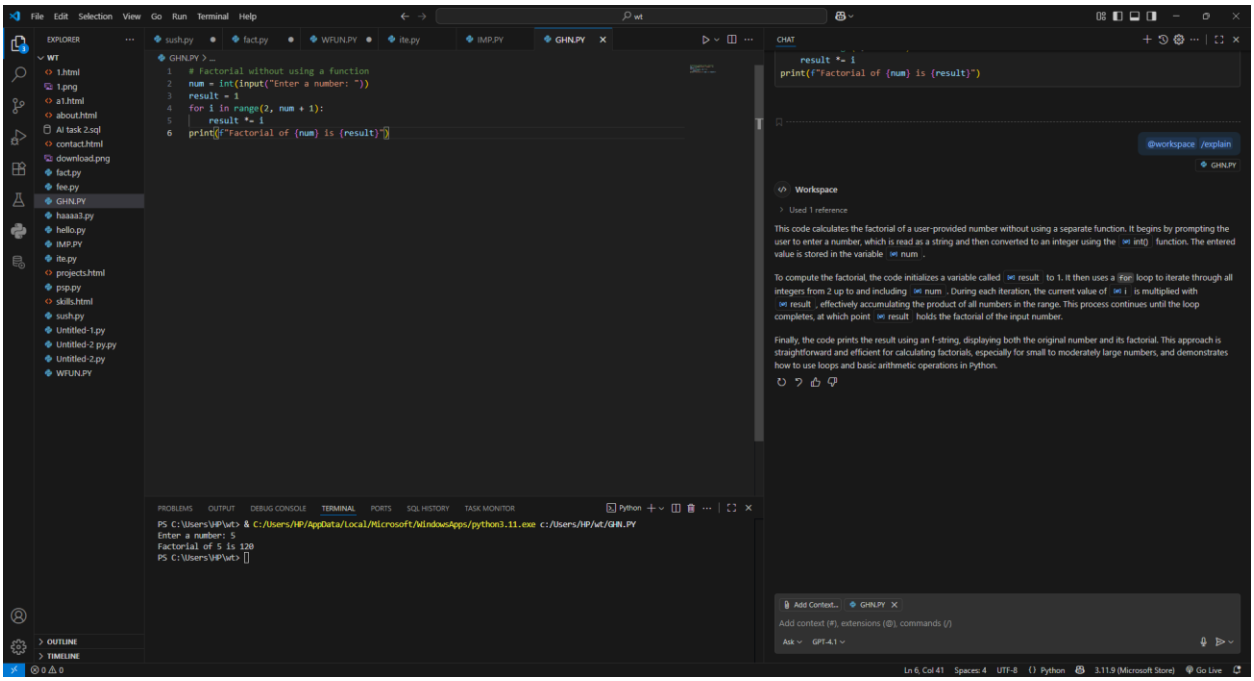
Comparative Analysis – With vs Without Functions

• Description:

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

• Expected Output:

- A comparison table or short report explaining the differences.




# TASK-5:

Iterative vs Recursive Factorial

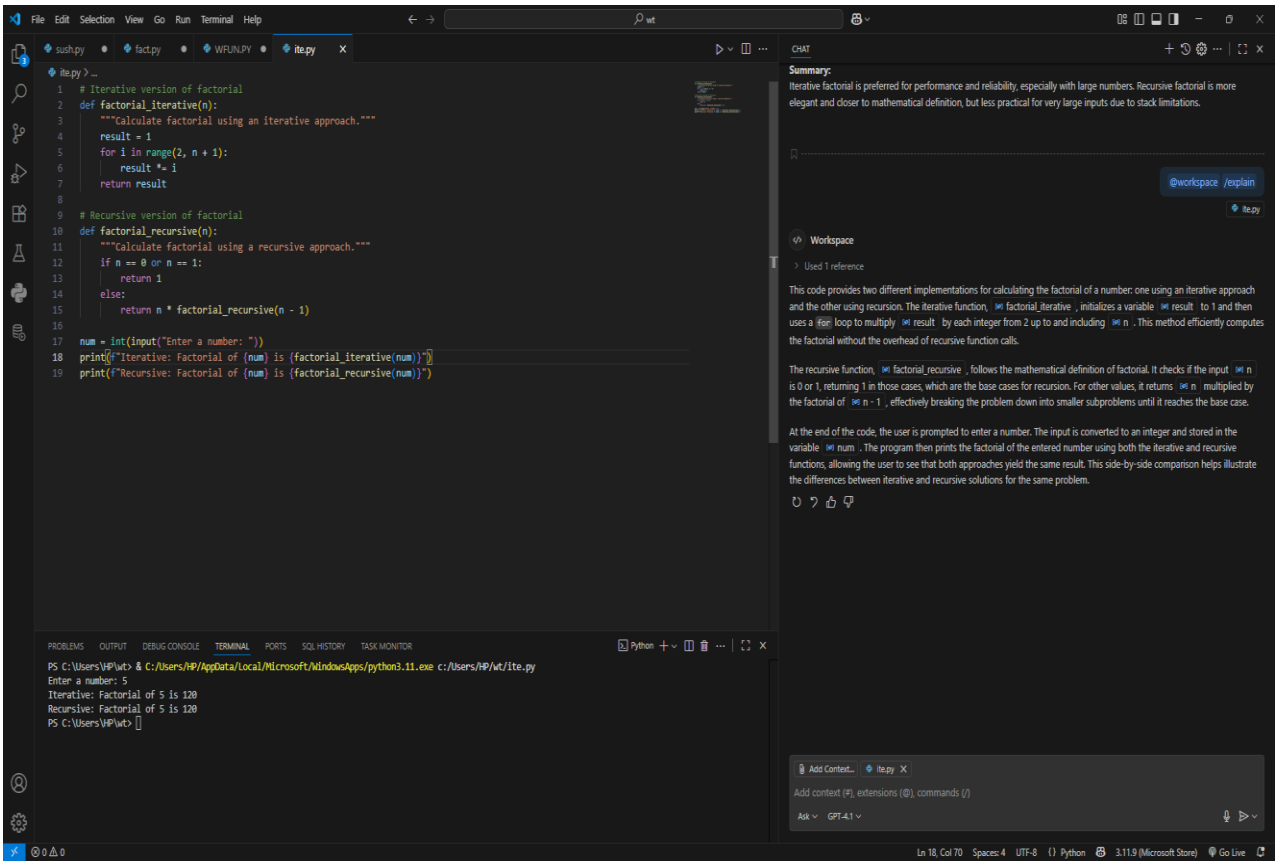
• Description:

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.


File Edit Selection View Go Run Terminal Help
wt
sush.py
fact.py
WFUN.PY
ite.py
ite.py > ...
# Iterative version of factorial
def factorial_iterative(n):
"""Calculate factorial using an iterative approach."""
result = 1
for i in range(2, n + 1):
result *= i
return result
# Recursive version of factorial
def factorial_recursive(n):
"""Calculate factorial using a recursive approach."""
if n == 0 or n == 1:
return 1
else:
return n * factorial_recursive(n - 1)
num = int(input("Enter a number: "))
print(f"Iterative: Factorial of {num} is {factorial_iterative(num)}")
print(f"Recursive: Factorial of {num} is {factorial_recursive(num)}")
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS SQL HISTORY TASK MONITOR
Python
PS C:\Users\HP\wt> & C:/Users/HP/AppData/Local/Microsoft/WindowsApps/python3.11.exe c:/Users/HP/wt/ite.py
Enter a number: 5
Iterative: Factorial of 5 is 120
Recursive: Factorial of 5 is 120
PS C:\Users\HP\wt>
CHAT
Summary:
Iterative factorial is preferred for performance and reliability, especially with large numbers. Recursive factorial is more elegant and closer to mathematical definition, but less practical for very large inputs due to stack limitations.
@workspace /explain
ite.py
Workspace
Used 1 reference
This code provides two different implementations for calculating the factorial of a number: one using an iterative approach and the other using recursion. The iterative function, factorial_iterative, initializes a variable result to 1 and then uses a for loop to multiply result by each integer from 2 up to and including n. This method efficiently computes the factorial without the overhead of recursive function calls.
The recursive function, factorial_recursive, follows the mathematical definition of factorial. It checks if the input n is 0 or 1, returning 1 in those cases, which are the base cases for recursion. For other values, it returns n multiplied by the factorial of n - 1, effectively breaking the problem down into smaller subproblems until it reaches the base case.
At the end of the code, the user is prompted to enter a number. The input is converted to an integer and stored in the variable num. The program then prints the factorial of the entered number using both the iterative and recursive functions, allowing the user to see that both approaches yield the same result. This side-by-side comparison helps illustrate the differences between iterative and recursive solutions for the same problem.
Add Context...
ite.py
Add context (#), extensions (@), commands (/)
Ask
GPT-4.1
Ln 18, Col 70 Spaces: 4 UTF-8 Python 3.11.9 (Microsoft Store) Go Live