**AI ASSISTED CODING**

**Lab-1.1:**

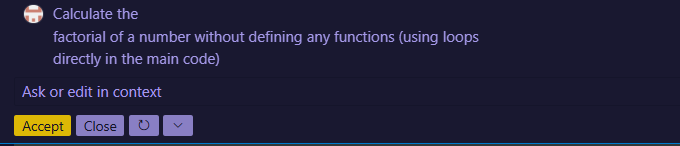
Environment Setup-GitHub Copilot & VS Code

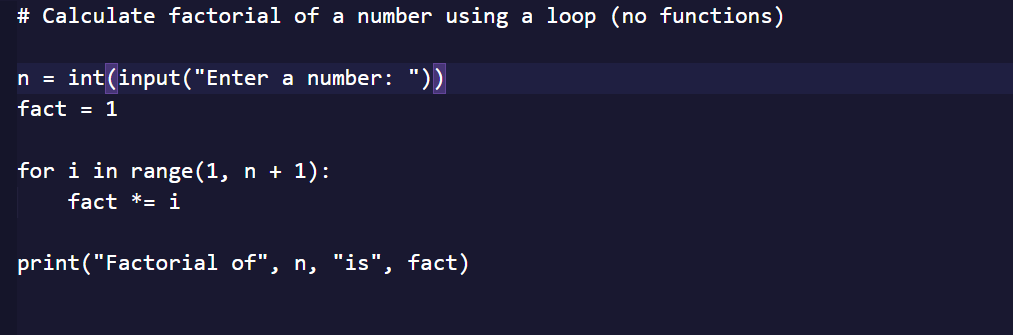
Name : Cholluri Srikruthi

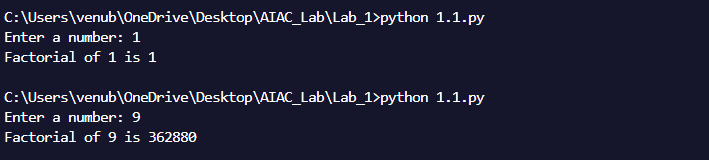
Enroll no : 2403A510H0

Batch : 06

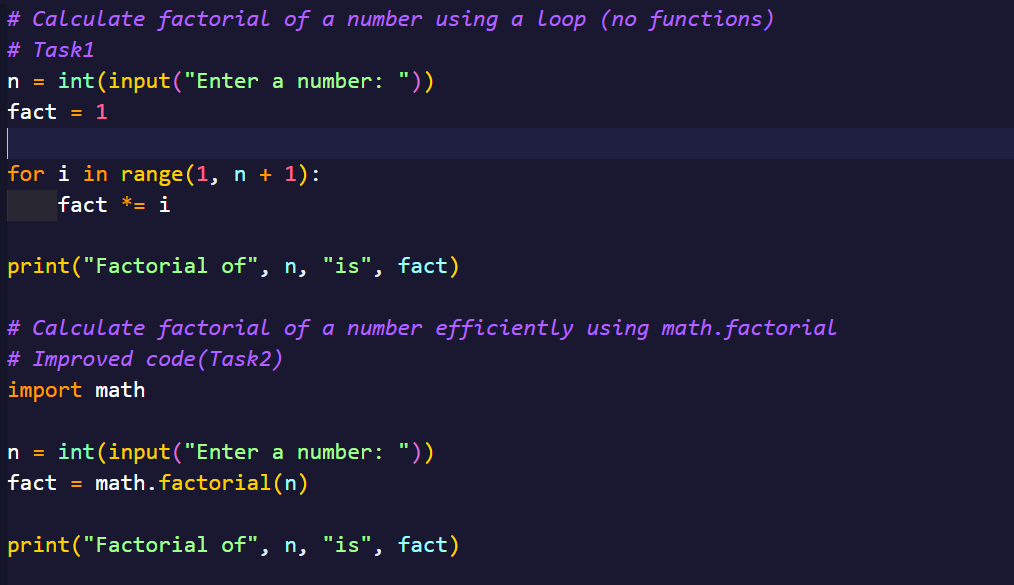
Task-1

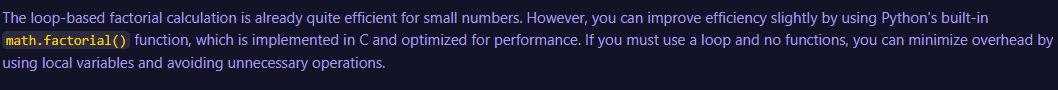




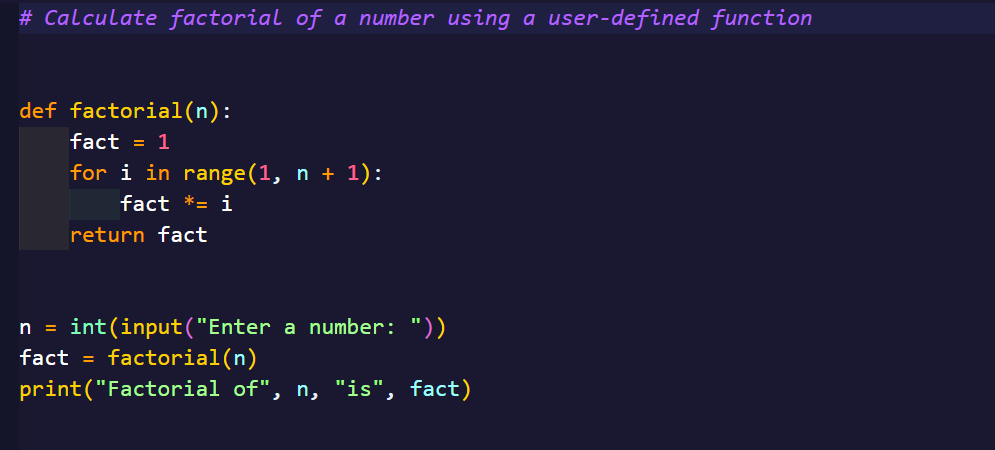


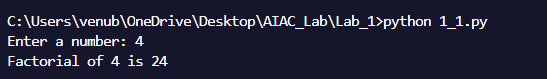
Task-2



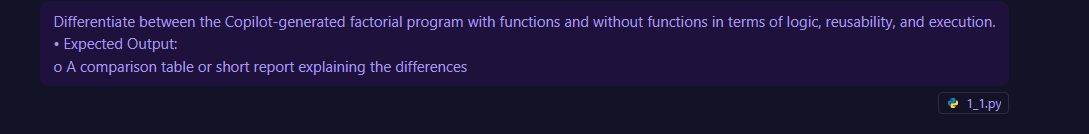


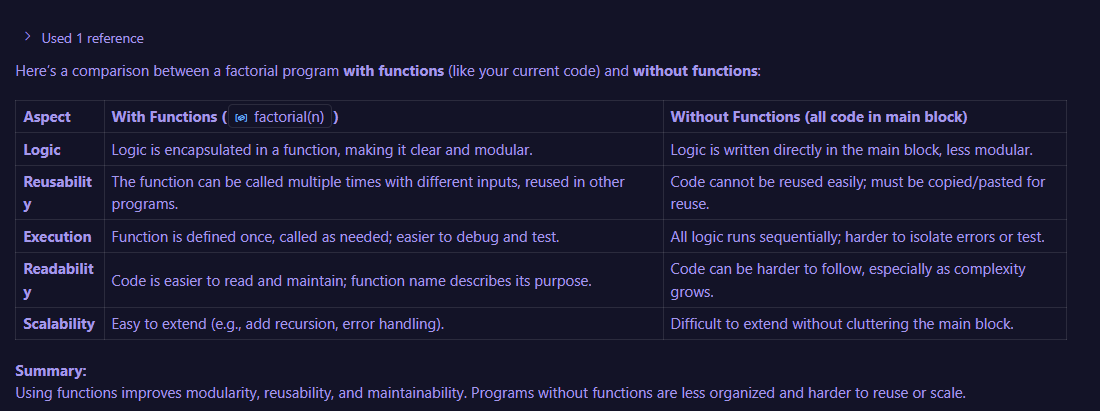
Task-3





Task-4

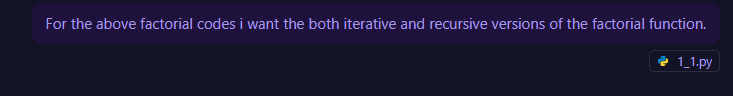


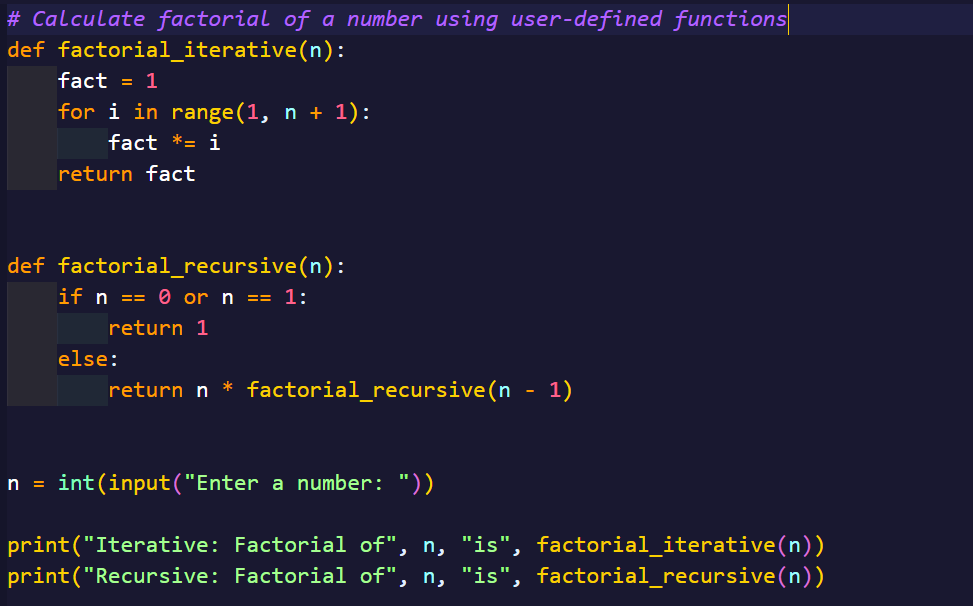


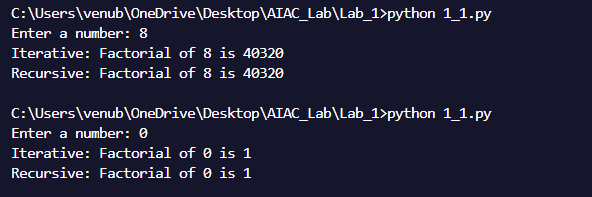
Descriptive form of Task-4:

Using functions encapsulates the factorial logic, making the code modular and easier to understand. Without functions, the logic is embedded in the main block, which can make the code harder to follow and maintain. Reusability: Functions allow the factorial logic to be reused in other parts of the program or in different programs. Without functions, reusing the logic requires copying and pasting code. Execution: Functions can be tested and debugged independently, improving reliability. Without functions, errors are harder to isolate.

Task-5







Descriptive form of Task-5:

The iterative approach uses a loop to multiply numbers from 1 up to the input value. This method is straightforward, efficient, and avoids the overhead of repeated function calls. It is well-suited for large input values, as it does not risk exceeding Python’s recursion depth limit.

The recursive approach defines the factorial in terms of itself, calling the function repeatedly with decremented values until reaching the base case. This method is elegant and closely matches the mathematical definition of factorial. However, it can be less efficient for large numbers due to the overhead of multiple function calls and the risk of stack overflow if the recursion depth is too great.