# AIASSISTEDCODING

## LABASSIGNMENT-1

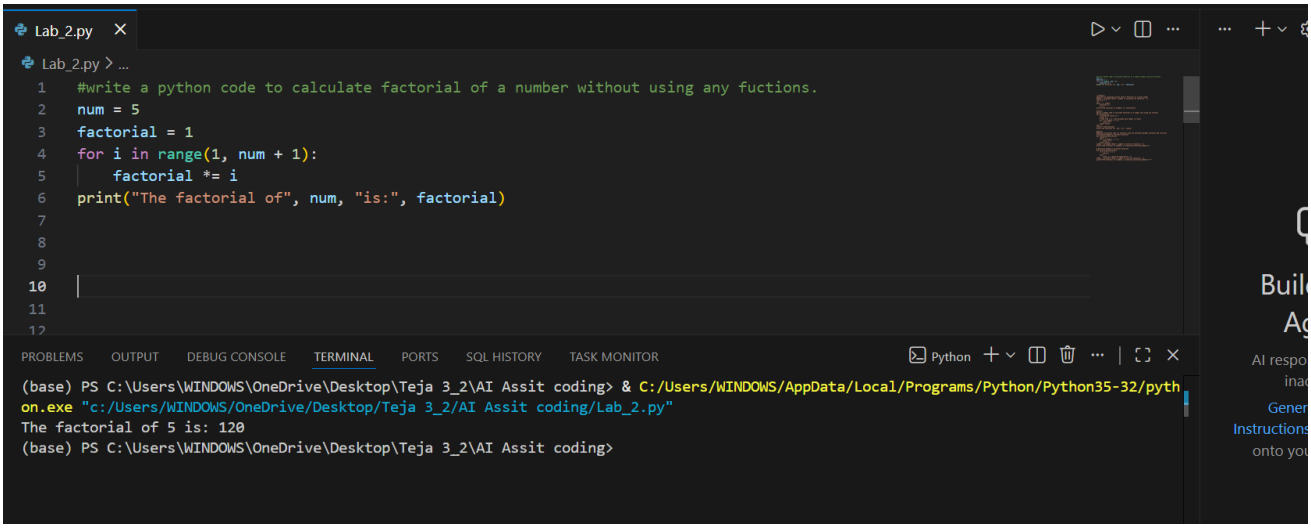
Name:Kashireddy Tejaswini

HT.NO: 2303A51425

Batch:21

Question-1:AI-GeneratedLogicWithoutModularization(Factorialwithout Functions)

Prompt:#generateacodetogetfactorialofanumberwithoutusingfunctions Code and Output Screenshot:

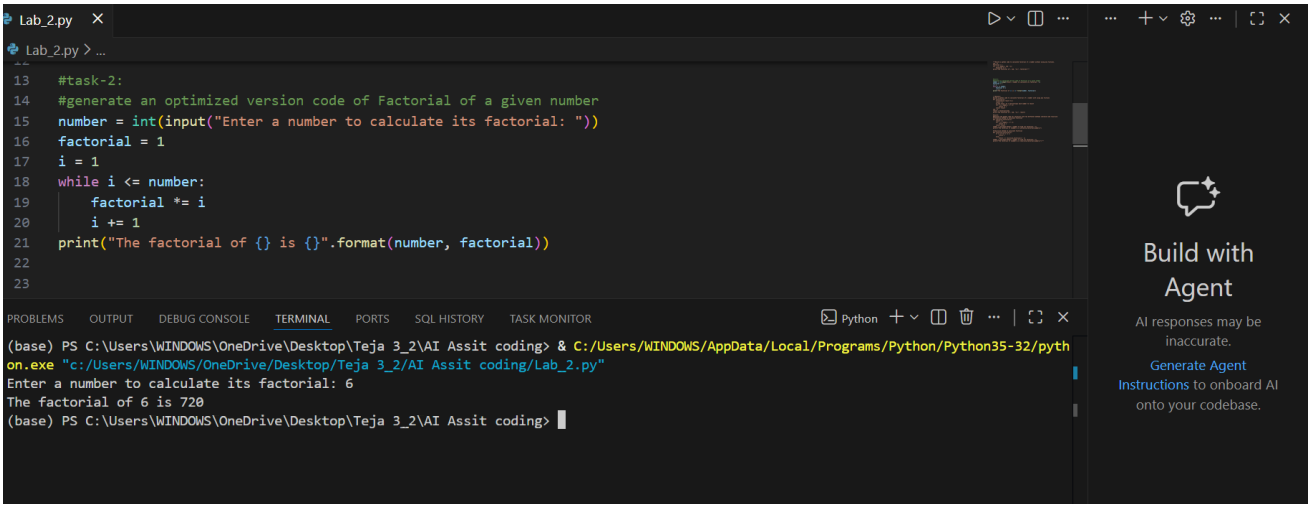



## Observation:

GitHub Copilot was helpful formebeinga beginner,ithelpedmewiththeright typeoflogicinloops.Itshortenedthetimetoconsidersyntaxandbasiccontrol flow logic. Copilot made the things easy like initializing a variable properly and choosing good loop condition expressions. For new user it works more like an intelligent code assistant than an educator. Finally it improves confidence and quickness and must be done while also learning base skills.

Question-2:AICodeOptimization&Cleanup(ImprovingEfficiency)

Prompt:#generateanoptimizedversioncodeofFactorialofagivenNumber. Code and Output Screenshot:

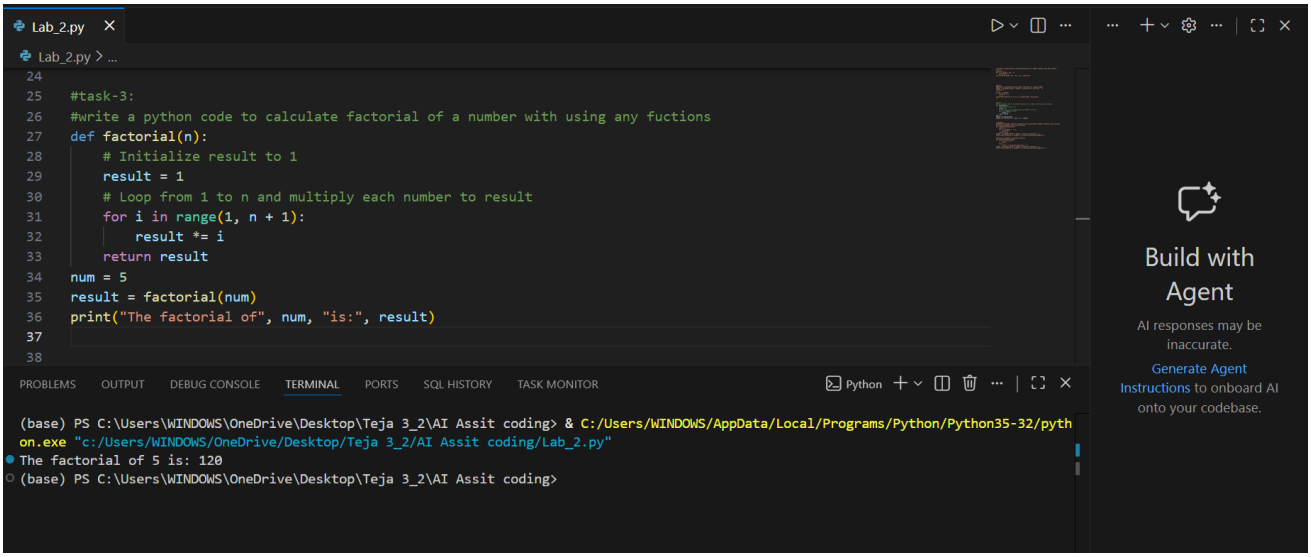



Observation:

UsingGitHubCopilotfortheoptimizedfactorialcodeproducedamoreefficient and well-structured solution. The optimized logic reduced unnecessary computations and improved performance. Copilot suggested clear function design and concise implementation, making the code easy to read and reuse. Inlinecommentshelpedexplaintheoptimizedapproach,encouraginggood programming practices.

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

Prompt:#generateacodetogetfactorialofanumberwithusingfunctions Code and Output Screenshot:




Observation:

UsingGitHubCopilotforamodulardesignmadethecodemorestructuredand easier to understand. Copilot suggested meaningful function names and clear parameters,whichimprovesreadability.Theseparationoflogicintoafunction allows the same factorial computation to be reused across multiple programs. Inline comments generated by Copilot helped clarify each step of the logic for beginners. Copilot naturally encourages good programming practices through function-based design.

Question-4:ComparativeAnalysis–ProceduralvsModularAICode(Withvs Without Functions)

Prompt:Noprompt

CodeandOutputScreenshot:Nocode Comparison

Table:

| Features | Without Functions (Procedural Code) | With Functions (Modular Code) |
|---|---|---|
| Logic Clarity | Logic is written in a single flow, easy to follow for small programs. | Logic is divided into functions, making it clearer and more structured. |
| Code Structure | Simple and linear structure. | Organized and modular structure. |
| Reusability | Code cannot be reused easily and may require repetition. | Functions can be reused multiple times in the program. |
| Debugging Ease | Debugging becomes difficult as the program grows larger. | Easier to debug because errors can be traced to specific functions. |
| Suitability for Large Projects | Not suitable for large or complex programs. | Well suited for large-scale projects. |
| AI Dependency Risk | Higher risk due to long blocks of AI-generated code that are harder to understand. | Lower risk because functions encourage understanding and verification of AI-generated logic. |

TechnicalReport:

In terms of logic clarity, procedural code written without functions is simple and straightforward for very small programs. Since all instructions are written in a single continuous sequence, beginners can easily follow the execution flow from input to output. This approach helps new programmers understand basic programming concepts without the additional complexity of functions. However, as the program becomes longer and more complex, procedural code starts to lose clarity. Large blocks

of logic reduce readability and make it difficult to understand how different parts of the program are connected. This often leads to confusion and errors during development.

Modular code using functions significantly improves logic clarity by breaking the program into smaller, meaningful units. Each function is designed to perform a specific task, making the code easier to read, understand, and manage. Well-named functions clearly describe their purpose, allowing developers to understand the overall program structure at a glance. This structured approach enhances code organization and makes collaboration easier, especially when multiple developers are working on the same project.

Debugging is another important aspect where modular code performs better than procedural code. Procedural programs are easy to debug only when they are small. As the program grows, locating errors becomes difficult because all logic is tightly coupled in one flow. A single mistake can affect multiple parts of the program, increasing debugging time. In contrast, modular programming simplifies debugging by allowing developers to test and debug individual functions independently. Errors can be quickly traced to a specific function, making troubleshooting faster and more efficient.

Regarding AI dependency risk, both procedural and modular code can be affected if developers blindly trust AI-generated suggestions. However, procedural code poses a higher risk because large blocks of AI-generated logic are harder to verify and understand. Modular code reduces this risk by encouraging developers to review, test, and understand each function separately. This promotes better human oversight and validation of AI-generated code.

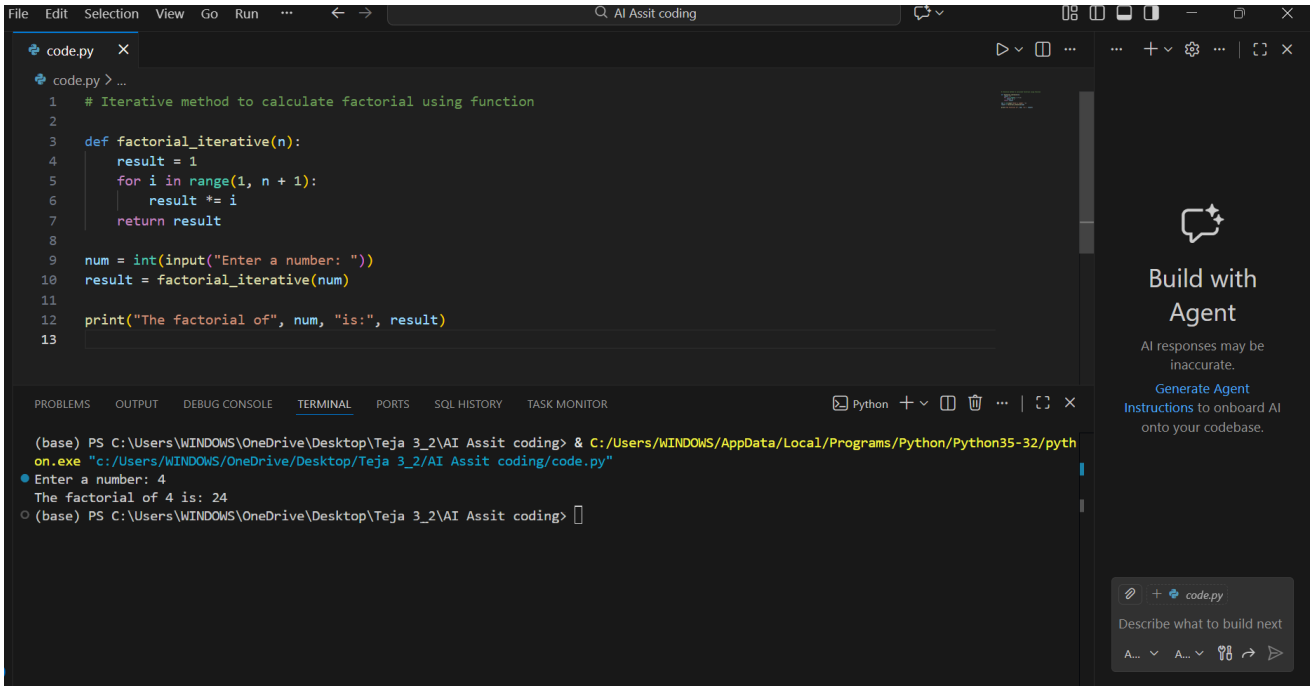
Overall, function-based modular programming is more reliable, maintainable, and suitable for large-scale and professional software development.

## Question-5:AI-GeneratedIterativevsRecursiveThinking

### Iterative:

### Prompt:#generateacodetogetfactorialiteratively Code

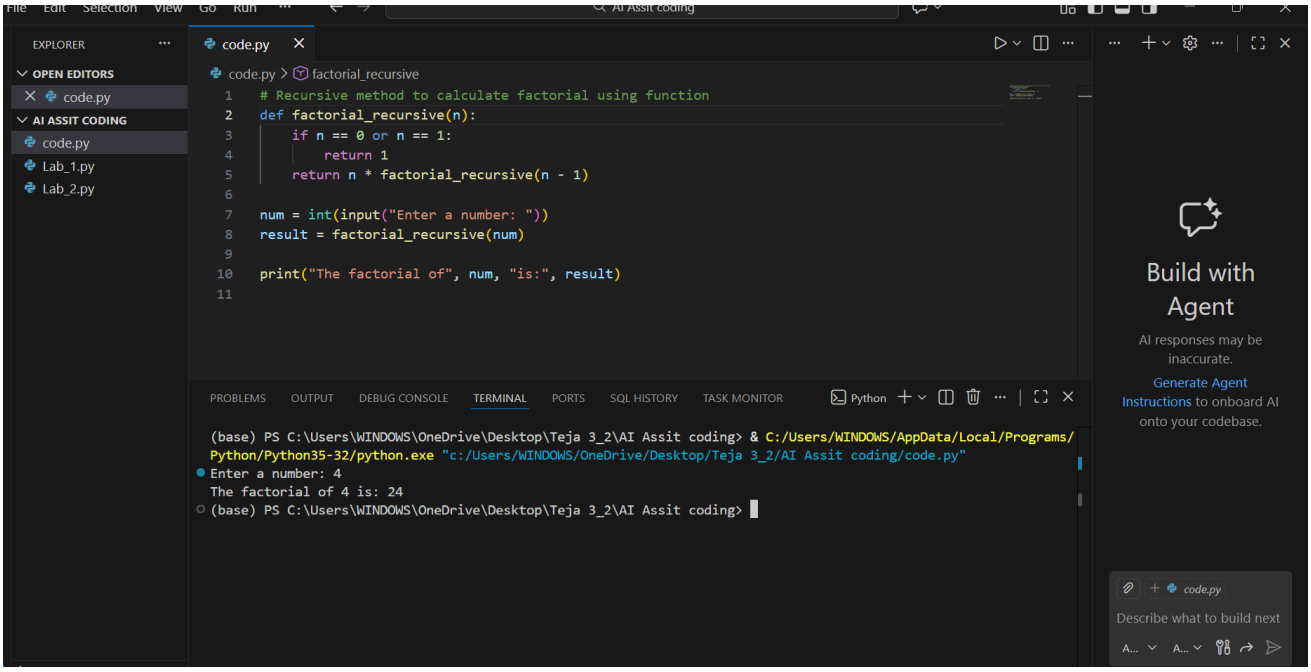
### and Output Screenshots:




## Recursive:

## Prompt:#generateacodetogetfactorialrecursively Code

## and Output Screenshots:




## ExecutionFlowExplanation:

In the **iterative approach**,the program starts with a value of 1and uses a loop to multiply it with every number from 1 up to the given input. The result is updated step by step inside the same loop until the final factorial value is obtained.

In the **recursive approach**,the function solves the problem by breaking it into smaller parts. Each function call depends on the result of the next call, continuing until it reaches a base case (0 or 1). After reaching the base case, the function calls return one by one, multiplying the values together to produce the final factorial.

## ComparativeAnalysis:

**Readability:**
The iterative approach is usually easier for beginners to read and understand because the flow of execution is straightforward. Recursive code, although mathematicallyelegant,canbehardertofollowsincethefunctionkeepscalling itself, which makes tracing the execution more complex.

**StackUsage:**
Iterativeimplementationsuseconstantmemorybecausetheyrelyonasingle loop.Incontrast,recursiveimplementationsconsumeextrastackmemoryfor every function call, which increases memory usage.

**PerformanceImplications:**
Iterativesolutionsaregenerallyfasterandmorememory-efficient.Recursive solutions introduce overhead due to repeated function calls and stack operations,whichcanslowdownexecution.

**When Recursion Is Not Recommended:**
Recursion should beavoided when dealing with very large in puts because it can cause stack overflow. It is also not ideal for performance-critical or memory-limitedapplications,andwhentheproblemlogicdoesnotnaturally suit a recursive approach.