|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **SCHOOL OF COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE** | | | | | **DEPARTMENT OF COMPUTER SCIENCE ENGINEERING** | | | | |
| **Program Name:** B. Tech | | | | **Assignment Type: Lab** | | | **Academic Year:**2025-2026 | | |
| **Course Coordinator Name** | | | | Dr. Rishabh Mittal | | | | | |
| **Instructor(s) Name** | | | | |  | | --- | | Mr. S Naresh Kumar | | Ms. B. Swathi | | Dr. Sasanko Shekhar Gantayat | | Mr. Md Sallauddin | | Dr. Mathivanan | | Mr. Y Srikanth | | Ms. N Shilpa | | Dr. Rishabh Mittal (Coordinator) | | Dr. R. Prashant Kumar | | Mr. Ankushavali MD | | Mr. B Viswanath | | Ms. Rapelly Nandini | | Ms. A. Anitha | | Ms. M.Madhuri | | Ms. Katherashala Swetha | | Ms. Velpula sumalatha | | Mr. Bingi Raju | | | | | | |
| **CourseCode** | | | 23CS002PC304 | **Course Title** | | AI Assisted Coding | | | |
| **Year/Sem** | | | III/II | **Regulation** | | R23 | | | |
| **Date and Day**  **of Assignment** | | | **Week1 - Tuesday** | **Time(s)** | | 23CSBTB01 To 23CSBTB52 | | | |
| **Duration** | | | 2 Hours | **Applicable to**  **Batches** | | All batches | | | |
| **Assignment Number:1.2**(Present assignment number)/**24**(Total number of assignments) | | | | | | | | | |
|  | | | | | | | | | |
|  | **Q.No.** | **Question** | | | | | | ***Expected Time***  ***to complete*** |  |
|  | 1 | Lab 1: Environment Setup – *GitHub Copilot and VS Code Integration + Understanding AI-assisted Coding Workflow*  **Lab Objectives:**   * To install and configure GitHub Copilot in Visual Studio Code. * To explore AI-assisted code generation using GitHub Copilot. * To analyze the accuracy and effectiveness of Copilot's code suggestions. * To understand prompt-based programming using comments and code context   **Lab Outcomes (LOs):**  After completing this lab, students will be able to:   * Set up GitHub Copilot in VS Code successfully. * Use inline comments and context to generate code with Copilot. * Evaluate AI-generated code for correctness and readability. * Compare code suggestions based on different prompts and programming styles.   Task 0   * Install and configure GitHub Copilot in VS Code. Take screenshots of each step.   Expected Output   * Install and configure GitHub Copilot in VS Code. Take screenshots of each step.   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 * **Expected Deliverables**   + A working Python program generated with Copilot assistance   + Screenshot(s) showing:   + The prompt you typed   + Copilot’s suggestions   + Sample input/output screenshots   + Brief reflection (5–6 lines):   + How helpful was Copilot for a beginner?   + Did it follow best practices automatically?   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”*   * **Expected Deliverables**   + Original AI-generated code   + Optimized version of the same code   + Side-by-side comparison   + Written explanation:     - What was improved?     - Why the new version is better (readability, performance, maintainability.   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) * **Expected Deliverables**   + AI-assisted function-based program   + Screenshots showing:   + Prompt evolution   + Copilot-generated function logic   + Sample inputs/outputs   + Short note:   + How modularity improves reusability.   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 * **Expected Deliverables**   Choose **one**:   * + A comparison table **OR**   + A short technical report (300–400 words).   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**   * **Expected Deliverables**   Two AI-generated implementations  Execution flow explanation (in your own words)  Comparison covering:   * + Readability   + Stack usage   + Performance implications   + When recursion is *not* recommended.   **Submission Requirements**   1. Generate code for each task with comments. 2. Screenshots of Copilot suggestions. 3. Comparative analysis reports (Task 4 and Task 5). 4. Sample inputs/outputs demonstrating correctness.   **Note: Report should be submitted as a word document for all tasks in a single document with prompts, comments & code explanation, and output and if required, screenshots.** | | | | | | Week1 - Monday |  |

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

**Given prompt :**

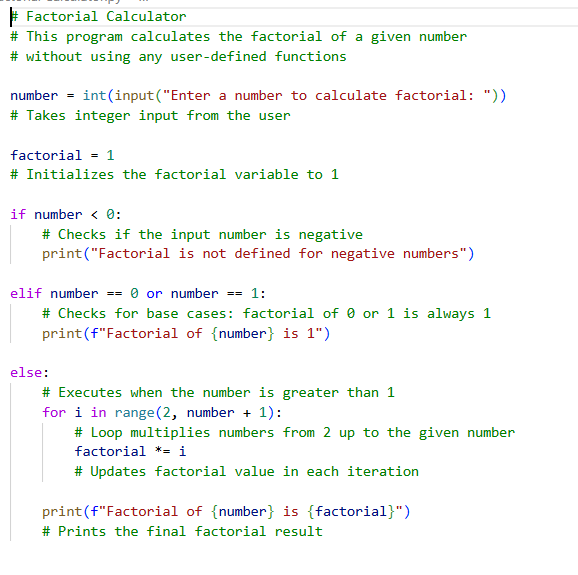
Write a Python program to calculate factorial of a number

Do not use any user-defined functions

Use loops and variables only

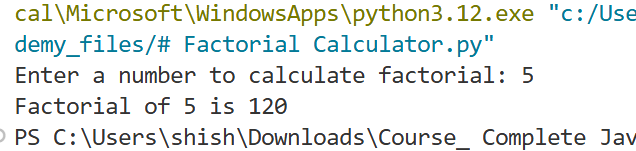
Implement logic directly in the main program flow

**Code suggested:**

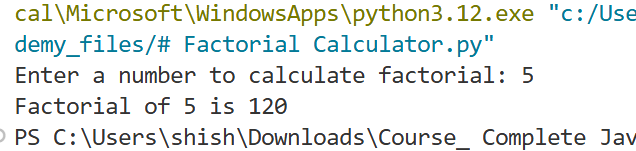
****

**Sample Input / Output**

**Input:**



**Output:**

****

**Reflection (5–6 Lines):**

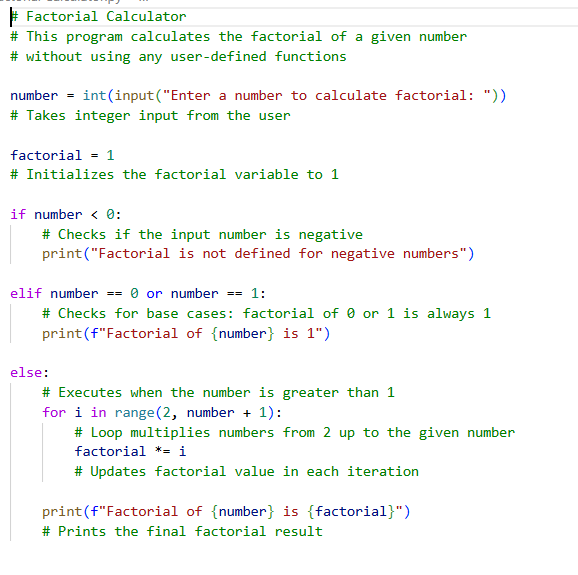
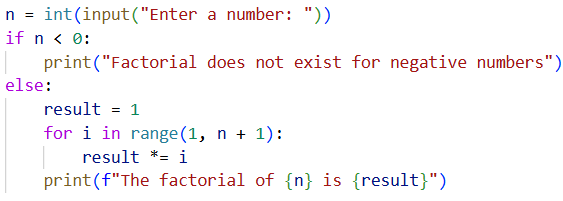
1. GitHub Copilot generated correct and efficient logic based on simple prompts.
2. It handled edge cases like negative numbers and zero automatically.
3. The AI reduced coding time significantly for beginners.
4. However, it does not optimize structure unless explicitly prompted.
5. Understanding the generated code is essential before using it in projects.
6. Copilot is best used as an assistant, not a replacement for logic building.
7. Task 2: AI Code Optimization & Cleanup (Improving Efficiency)

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

**Givenprompt:**

Simplify this factorial program and make it more readable and efficient without using functions.

**Code Suggested & Original Code:**

****

**Sample Input:-**

****

**Sample output:-**



**Written explanation :**

**What Was Improved?**

* Removed unnecessary conditional checks, especially for the n == 0 case
* Simplified the logic so the loop handles the factorial naturally
* Improved variable naming for clarity
* Used f-strings for cleaner output formatting

**Why the New Version Is Better?**

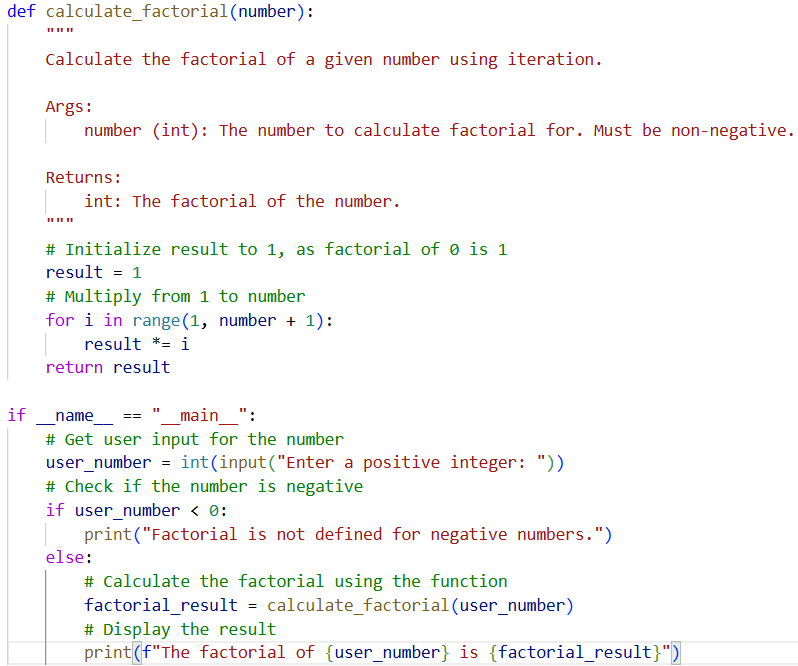
* **Readability:** The code is shorter and easier to understand at a glance
* **Performance:** Fewer conditional checks make the code slightly more efficient
* **Maintainability:** Cleaner structure makes it easier to modify or review later
* **Best Practices:** Uses modern Python features and avoids redundant logic

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

**Given prompt :**

Write a Python program to calculate factorial using a user-defined function. Use meaningful function and variable names, include inline comments, and call the function from the main block.

**Suggest Code:**

****

**Sample inputs**:



**Sample outputs:**



**Short note:**

**How modularity improves reusability.**

Modularity allows the factorial logic to be reused in multiple programs without rewriting code. The function can be imported into other scripts, improving consistency and reducing errors.

It also makes the code easier to test, debug, and maintain.

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

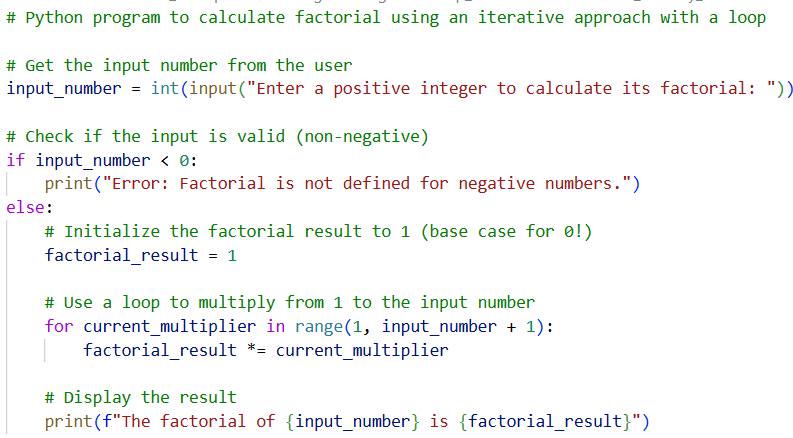
|  |  |  |
| --- | --- | --- |
| **Criteria** | **Procedural Code (Without Functions)** | **Modular Code (With Functions)** |
| **Logic Clarity** | Logic is written in a single block, which is easy for small programs but harder to follow as code grows | Logic is divided into functions, making the program flow clearer and more organized |
| **Reusability** | Code cannot be reused directly and must be rewritten in other scripts | Functions can be reused across multiple programs without duplication |
| **Debugging Ease** | Debugging is difficult because all logic is mixed together | Easier debugging since errors can be isolated within a function |
| **Suitability for Large Projects** | Not suitable for large projects due to poor structure and scalability | Well-suited for large projects with clean and maintainable design |
| **AI Dependency Risk** | Higher risk, as developers may rely on AI-generated blocks without understanding them | Lower risk, since modular code encourages understanding, testing, and reuse |

**Task 5: AI-Generated Iterative vs Recursive Thinking**

**Given Prompt**:

Generate a Python program to calculate factorial using an iterative approach with a loop and clear variable names.

**Suggested Code:**

****

**Sample Input:**

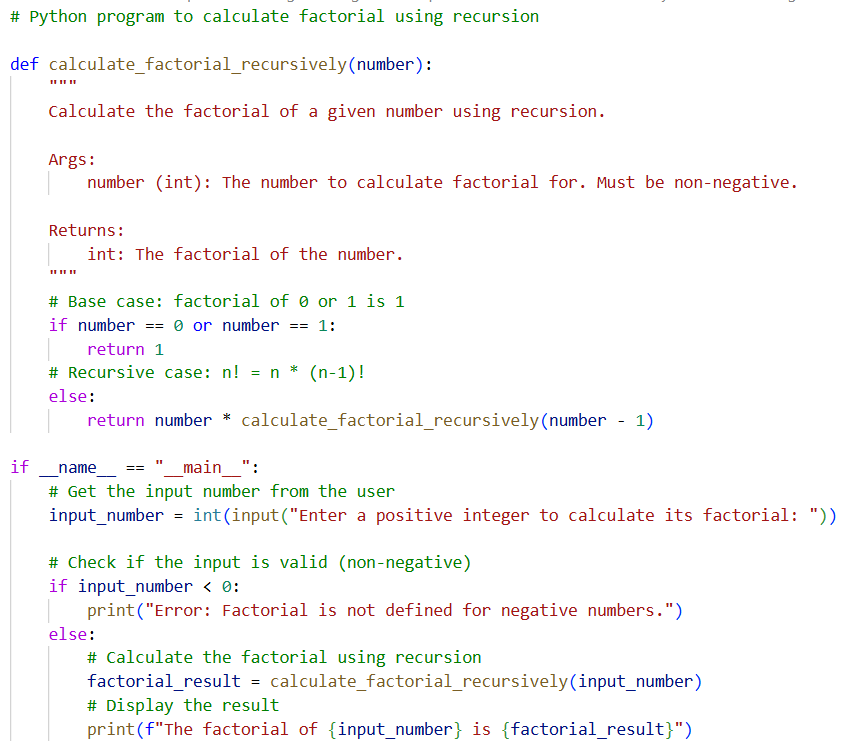
****

**Sample output:**



**Given Code:**

Generate a Python program to calculate factorial using recursion with proper base cases and comments.

**Suggested Code** **:**

**Sample Input:**

****

**Sample output:**

****

**Execution Flow Explanation :**

* In the iterative approach, the program uses a loop to multiply numbers from 1 to n, updating the result step by step.
* In the recursive approach, the function calls itself with a smaller value until it reaches the base case (n = 0 or 1), then returns the result back through the call stack.
* Both methods follow different thinking styles but compute the same final result.

**Comparison: Iterative vs Recursive**

|  |  |  |
| --- | --- | --- |
| Aspect | Iterative Approach | Recursive Approach |
| Readability | Easier for beginners to understand | Cleaner and mathematically expressive |
| Stack Usage | Uses constant memory | Uses call stack for each function call |
| Performance | More efficient, no overhead | Slower due to repeated function calls |
| When Recursion Is Not Recommended |  | For large inputs (may cause stack overflow) |