AI\_AC Assignment – 1.2

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name: K. Tharani  HTNO -2303A52067  Batch - 32 | | | | |
|  | **Q. No.** | **Question** |  |  |
|  | 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.   **Step-by-Step Procedure**  Step 1: Install Visual Studio Code   1. Download and install Visual Studio Code from the official website. 2. Launch Visual Studio Code after successful installation.     **Step 2: Install and Configure GitHub Copilot**   1. Open the Extensions view (Ctrl + Shift + X) in VS Code. 2. Search for **GitHub Copilot** and install the extension. 3. Sign in using your GitHub account when prompted.     **Step 3: Verify Copilot Functionality**   1. Create a new Python file in VS Code. 2. Type a comment describing a simple program. 3. Observe Copilot generating inline code suggestions and accept them using Tab.       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 Given to GitHub Copilot**  # Write a python program to calculate factorial of a number  # Do not use any functions  # Use only loops and variables    **Sample input/output**   **Brief Reflection (5–6 Lines)**  * GitHub Copilot was very helpful for a beginner as it generated correct logic instantly. * The AI understood the constraint of not using functions. * It automatically followed standard looping logic for factorial calculation. * Variable names generated were simple and readable. * Basic error handling for negative input was included. * Overall, Copilot reduced development time and cognitive effort.   GitHub Copilot successfully generated a working factorial program without modularization, meeting all constraints. This demonstrates how AI-assisted coding can help beginners quickly implement correct logic using simple prompts.  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”*    **Side-by-Side Comparison**   | **Point** | **Original Code** | **Optimized Code** | | --- | --- | --- | | Loop start | Starts from 1 | Starts from 2 | | Multiplication | Extra step with 1 | Avoids unnecessary step | | Variable name | factorial | result (simpler) | | Readability | Good | Better | | Maintenance | Average | Easier |   **Explanation**  **What Was Improved?**   * The loop now starts from 2 instead of 1, which avoids an unnecessary multiplication. * The variable name was simplified to make the code cleaner. * The overall structure was kept simple and easy to follow.   **Why the New Version Is Better**   * The code is easier to read and understand. * It performs the same task with slightly fewer operations. * It is easier to maintain and review before committing to a shared repository.   **Conclusion**  By manually reviewing the code, small improvements were made without changing the logic. This shows that even simple AI-generated code should be reviewed and refined before final use.  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 Given to GitHub Copilot**  # Write a python program to calculate factorial using a function  # Use clear variable names and add comments    Sample Output:    **Short Note: How Modularity Improves Reusability**  By using a function, the factorial logic is written only once and can be reused in multiple programs. This makes the code easier to maintain, test, and modify. If changes are required, they can be done in one place without affecting the rest of the program.  **Conclusion**  The modular version of the factorial program is more structured and reusable compared to the non-function version. This approach is suitable for larger programs where the same logic may be needed multiple times.  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).   **Comparison Table**   | **Point** | **Without Functions** | **With Functions** | | --- | --- | --- | | Code clarity | Logic and input/output mixed | Logic clearly separated | | Reusability | Cannot be reused | Can be reused | | Debugging | Difficult | Easy | | Large programs | Not suitable | Suitable | | Code changes | Risky | Safer |   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.   **Execution Flow Explanation**   * In the **iterative approach**, the factorial is calculated using a loop. * In the **recursive approach**, the function calls itself until it reaches the base case. * Both methods give the same final result.   **Comparison**   | **Point** | **Iterative** | **Recursive** | | --- | --- | --- | | Readability | Easy to understand | Slightly complex | | Memory usage | Uses less memory | Uses more stack memory | | Performance | Faster | Slower | | Risk | No risk | Stack overflow for large input |   **When Recursion Is Not Recommended**   * When input size is large * When memory usage must be low * In performance-critical programs   **Conclusion**  Both iterative and recursive approaches solve the same problem. However, the iterative method is safer and more efficient for large inputs, while recursion is useful for understanding problem breakdown. |  |  |