|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **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. Sujitha Reddy | | 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 – Thursday** | **Time(s)** | | 23CSBTB01 To 23CSBTB52 | | | |
| **Duration** | | | 2 Hours | **Applicable to**  **Batches** | | All batches | | | |
| **Assignment Number:1.3**(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 (Prime Number Check Without Functions)   * **Scenario**   + You are developing a **basic validation script** for a numerical learning application. * **Task Description**   Use GitHub Copilot to generate a Python program that:   * + Checks whether a given number is **prime**   + Accepts user input   + Implements logic **directly in the main code**   + Does **not** use any user-defined functions * **Expected Output**   + Correct prime / non-prime result   + Screenshots showing Copilot-generated code suggestions   + Sample inputs and outputs   **PROMPT:-**    **CODE:-**    **EXPLANATION:-**  The task is to create a Python program that uses AI-generated logic to check if a number is prime without any user-defined functions. The program will take user input and apply the prime-checking logic directly in the main code with conditional statements and loops. The program will clearly state whether the entered number is prime or not. Screenshots of GitHub Copilot’s code suggestions, along with sample inputs and outputs, are included to show AI support and confirm the solution's correctness.  Task 2: Efficiency & Logic Optimization (Cleanup)   * **Scenario**   The script must handle larger input values efficiently.   * **Task Description**   Review the Copilot-generated code from Task 1 and improve it by:   * + Reducing unnecessary iterations   + Optimizing the loop range (e.g., early termination)   + Improving readability   + Use Copilot prompts like:     - *“Optimize prime number checking logic”*     - *“Improve efficiency of this code”*   Hint: Prompt Copilot with phrases like *“optimize this code”*, *“simplify logic”*, or *“make it more readable”*   * **Expected Output**   + Original and optimized code versions   + Explanation of how the improvements reduce time complexity   **PROMPT:-**    **CODE:-**    **EXPLANATION:-**  In the original version, the loop checks all numbers from 2 to num - 1. This results in a time complexity of O(n), which is not efficient for large inputs. In the optimized version, the loop runs only up to the square root of the number. If a number has a factor greater than its square root, the smaller factor would have already been found. This change significantly reduces the number of iterations and improves the time complexity to O(√n). Additionally, using early termination (break) stops the loop as soon as a divisor is found. This further boosts performance while keeping the code simple and easy to read.  Task 3: Modular Design Using AI Assistance (Prime Number Check Using Functions)   * **Scenario**   The prime-checking logic will be reused across multiple modules.   * **Task Description**   Use GitHub Copilot to generate a function-based Python program that:   * + Uses a user-defined function to check primality   + Returns a Boolean value   + Includes meaningful comments (AI-assisted) * **Expected Output**   + Correctly working prime-checking function   + Screenshots documenting Copilot’s function generation   + Sample test cases and outputs   **PROMPT:-**    **CODE:-**      **EXPLANATION:-**  In this task, we develop a modular Python program with AI help to check if a number is prime. The prime-checking logic is placed inside a user-defined function that returns a Boolean value. This setup makes the code reusable across different modules. We include helpful comments to make it easier to read and understand the logic. The expected output includes a functioning code, screenshots showing GitHub Copilot generating the function-based code, and sample test cases with their corresponding outputs to confirm correctness and reusability.  Task 4: Comparative Analysis –With vs Without Functions   * **Scenario**   You are participating in a technical review discussion.   * **Task Description**   Compare the Copilot-generated programs:   * + Without functions (Task 1)   + With functions (Task 3)   + Analyze them based on:   + Code clarity   + Reusability   + Debugging ease   + Suitability for large-scale applications * **Expected Output**   Comparison table or short analytical report  **PROMPT:-**    **CODE:-**        **OUTPUT:-**      **EXPLANATION:-**  The program without functions is simple and suitable for beginners or small scripts, as all logic is written directly in the main code. However, this approach reduces reusability and makes debugging difficult as the program grows. In contrast, the function-based program follows a modular design, improving code clarity, reusability, and ease of debugging. By isolating the prime-checking logic inside a user-defined function, the code becomes more maintainable and scalable, making it more suitable for large-scale applications and professional software development.  Task 5: AI-Generated Iterative vs Recursive Fibonacci Approaches (Different Algorithmic Approaches to Prime Checking)   * **Scenario**   Your mentor wants to evaluate how AI handles **alternative logical strategies**.   * **Task Description**   Prompt GitHub Copilot to generate:   * + A **basic divisibility check** approach   + An **optimized approach** (e.g., checking up to √n) * **Expected Output**   + Two correct implementations   + Comparison discussing:     - Execution flow     - Time complexity     - Performance for large inputs     - When each approach is appropriate   **PROMPT:-**    **CODE:-**        **OUTPUT:-**      **EXPLANATION:-**  **Execution Flow**  The basic approach checks divisibility from 2 to n−1, testing all possible factors. The optimized approach checks divisibility only up to the square root of the number, stopping early if a divisor is found.  **Time Complexity**  The basic divisibility method has a time complexity of O(n), as it may iterate through almost all numbers less than n. The optimized approach reduces this to O(√n) by limiting the loop range.  **Performance for Large Inputs**  For small numbers, both approaches work similarly. However, for large input values, the optimized approach is significantly faster because it performs far fewer iterations.  **When Each Approach Is Appropriate**  The basic approach is suitable for learning purposes and small inputs, as it is easy to understand. The optimized approach is better for large-scale applications and performance-critical programs, where efficiency and speed are important.  **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 |  |