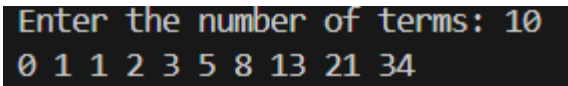
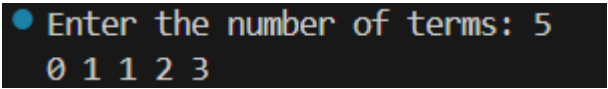


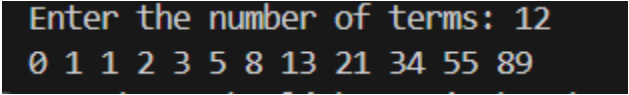
Goli Akash
Roll_no: 2303A53010

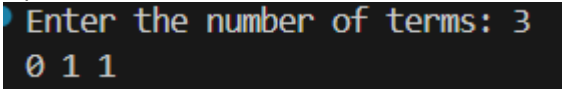
SCHOOL OF COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE		DEPARTMENT OF COMPUTER SCIENCE ENGINEERING	
Program Name: B. Tech		Assignment Type: Lab	Academic Year: 2025-2026
Course Coordinator Name		Dr. Rishabh Mittal	
Instructor(s) Name		Mr. S Naresh Kumar	
		Ms. B. Swathi	
		Dr. Sasanko Shekhar Gantayat	
		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 - Wednesday	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 – <i>GitHub Copilot and VS Code Integration + Understanding AI-assisted Coding Workflow</i>		Week1 - Monday

	<p>Lab Objectives:</p> <ul style="list-style-type: none"> ● 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 <p>Lab Outcomes (LOs): After completing this lab, students will be able to:</p> <ul style="list-style-type: none"> ● 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. 	
	<p>Task 0</p> <ul style="list-style-type: none"> ● Install and configure GitHub Copilot in VS Code. Take screenshots of each step. <p>Expected Output</p> <ul style="list-style-type: none"> ● Install and configure GitHub Copilot in VS Code. Take screenshots of each step. 	
	<p>Task 1: AI-Generated Logic Without Modularization (Fibonacci Sequence Without Functions)</p> <p>❖ Scenario You are asked to write a quick numerical sequence generator for a learning platform prototype.</p> <p>❖ Task Description Use GitHub Copilot to generate a Python program that:</p> <ul style="list-style-type: none"> ➤ Prints the Fibonacci sequence up to n terms ➤ Accepts user input for n ➤ Implements the logic directly in the main code ➤ Does not use any user-defined functions <p>❖ Expected Output</p> <ul style="list-style-type: none"> ➤ Correct Fibonacci sequence for given n 	

	<div>➤ Screenshot(s) showing Copilot-generated suggestions</div> <div>➤ Sample inputs and outputs</div> <div>Solution:</div> <div>Write a python program that takes an integer n as input prints the Fibonacci sequence upto to n terms.</div> <div><pre>#code def fibonacci(n): a, b = 0, 1 for _ in range(n): print(a, end=' ') a, b = b, a + b n = int(input("Enter the number of terms: ")) fibonacci(n)</pre></div> <div>#Output</div> <div></div>	
	<div>Task 2: AI Code Optimization & Cleanup (Improving Efficiency)</div> <div>❖ Scenario</div> <div>The prototype will be shared with other developers and needs optimization.</div> <div>❖ Task Description</div> <div>➤ Examine the Copilot-generated code from Task 1 and improve it by:</div> <div>➤ Removing redundant variables</div> <div>➤ Simplifying loop logic</div> <div>➤ Avoiding unnecessary computations</div> <div>➤ Use Copilot prompts such as:</div> <div><ul style="list-style-type: none">▪ <i>“Optimize this Fibonacci code”</i>▪ <i>“Simplify variable usage”</i></div> <div>Hint:</div> <div>Prompt Copilot with phrases like <i>“optimize this code”, “simplify logic”, or “make it more readable”</i></div> <div>❖ Expected Output</div> <div>➤ Original vs improved code</div>	

	<p>➤ Written explanation of:</p> <ul style="list-style-type: none"> ▪ What was inefficient ▪ How the optimized version improves performance and readability <p>Solution:</p> <p>Write an optimized Python program that takes an integer n as input and prints the Fibonacci sequence up to n terms by simplifying the logic and removing unnecessary variables, without using any user-defined functions.</p> <p>#Code</p> <pre>n = int(input("Enter the number of terms: ")) a, b = 0, 1 for _ in range(n): print(a, end=' ') a, b = b, a + b</pre> <p>#Output</p> 	
	<p>Task 3: Modular Design Using AI Assistance (Fibonacci Using Functions)</p> <p>❖ Scenario</p> <p>The Fibonacci logic is now required in multiple modules of an application.</p> <p>❖ Task Description</p> <p>Use GitHub Copilot to generate a function-based Python program that:</p> <ul style="list-style-type: none"> ➤ Uses a user-defined function to generate Fibonacci numbers ➤ Returns or prints the sequence up to n ➤ Includes meaningful comments (AI-assisted) <p>❖ Expected Output</p> <ul style="list-style-type: none"> ➤ Correct function-based Fibonacci implementation ➤ Screenshots documenting Copilot's function generation ➤ Sample test cases with outputs <p>Solution:</p> <p>Write a Python program that uses a user-defined function to generate and print the Fibonacci sequence up to n terms, accepts user input for n, and includes clear, meaningful comments.</p>	

	<pre> #Code # Function to print Fibonacci series def fibonacci(n): a, b = 0, 1 # First two numbers for _ in range(n): # Loop n times print(a, end=' ') a, b = b, a + b # Update values n = int(input("Enter the number of terms: ")) # User input fibonacci(n) # Function call #Output </pre> 	
	<p>Task 4: Comparative Analysis – Procedural vs Modular Fibonacci Code</p> <ul style="list-style-type: none"> ❖ Scenario You are participating in a code review session. ❖ Task Description Compare the Copilot-generated Fibonacci programs: <ul style="list-style-type: none"> ➤ Without functions (Task 1) ➤ With functions (Task 3) ➤ Analyze them in terms of: <ul style="list-style-type: none"> ▪ Code clarity ▪ Reusability ▪ Debugging ease ▪ Suitability for larger systems ❖ Expected Output Comparison table or short analytical report <p>Solution:</p> <p>Write a Python program to generate the Fibonacci series for n terms, once without using functions and once using functions.</p>	

	<p>#Code</p> <p>Without using functions</p> <pre> n = int(input("Enter the number of terms: ")) a, b = 0, 1 for _ in range(n): print(a, end=' ') a, b = b, a + b print() </pre> <p>Using functions</p> <pre> def fibonacci(n): a, b = 0, 1 for _ in range(n): print(a, end=' ') a, b = b, a + b </pre> <p>#Output</p> 	
	<p>Task 5: AI-Generated Iterative vs Recursive Fibonacci Approaches (Different Algorithmic Approaches for Fibonacci Series)</p> <ul style="list-style-type: none"> ❖ Scenario Your mentor wants to assess AI's understanding of different algorithmic paradigms. ❖ Task Description Prompt GitHub Copilot to generate: <ul style="list-style-type: none"> An iterative Fibonacci implementation A recursive Fibonacci implementation ❖ Expected Output <ul style="list-style-type: none"> ➤ Two correct implementations ➤ Explanation of execution flow for both ➤ Comparison covering: <ul style="list-style-type: none"> ▪ Time and space complexity ▪ Performance for large n ▪ When recursion should be avoided 	

Solution:

Write Python programs for the Fibonacci series using iterative and recursive methods, explain how each works, and compare them in terms of time, space, performance for large n, and when recursion should be avoided

#Code

Iterative method

```
def fibonacci_iterative(n):  
    a, b = 0, 1  
    for _ in range(n):  
        print(a, end=' ')  
        a, b = b, a + b  
    print()
```

#Recursive method

```
def fibonacci_recursive(n, a=0, b=1):  
    if n > 0:  
        print(a, end=' ')  
        fibonacci_recursive(n - 1, b, a + b)  
    else:  
        print()
```

#Output

```
Enter the number of terms: 12  
0 1 1 2 3 5 8 13 21 34 55 89
```