Name:G.Bhagath H.No:2303A51807 Batch: 26

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*	Week1 - Monday

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:

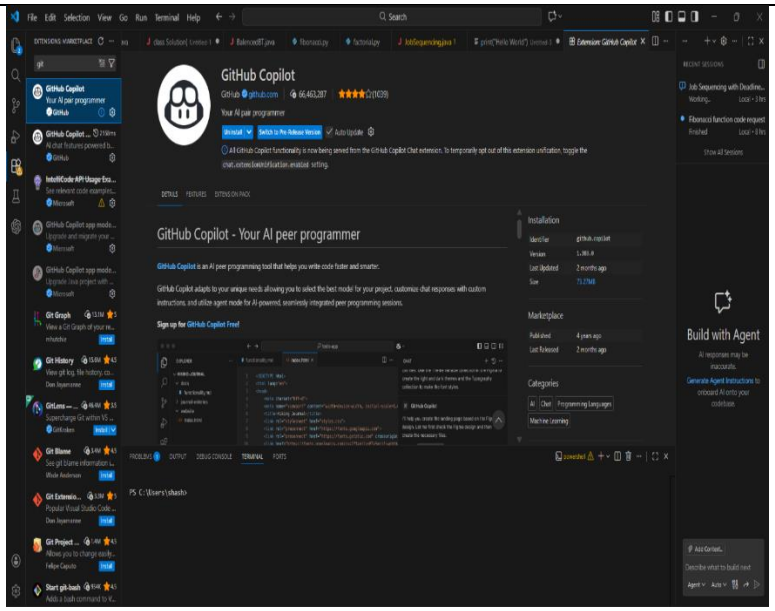
- Set up GitHub Copilot in VS Code successfully.
- Use inline comments and context to generate code with Copilot.
- Evaluate AI-generated code for correctness and readability.
- Compare code suggestions based on different prompts and programming styles.

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**

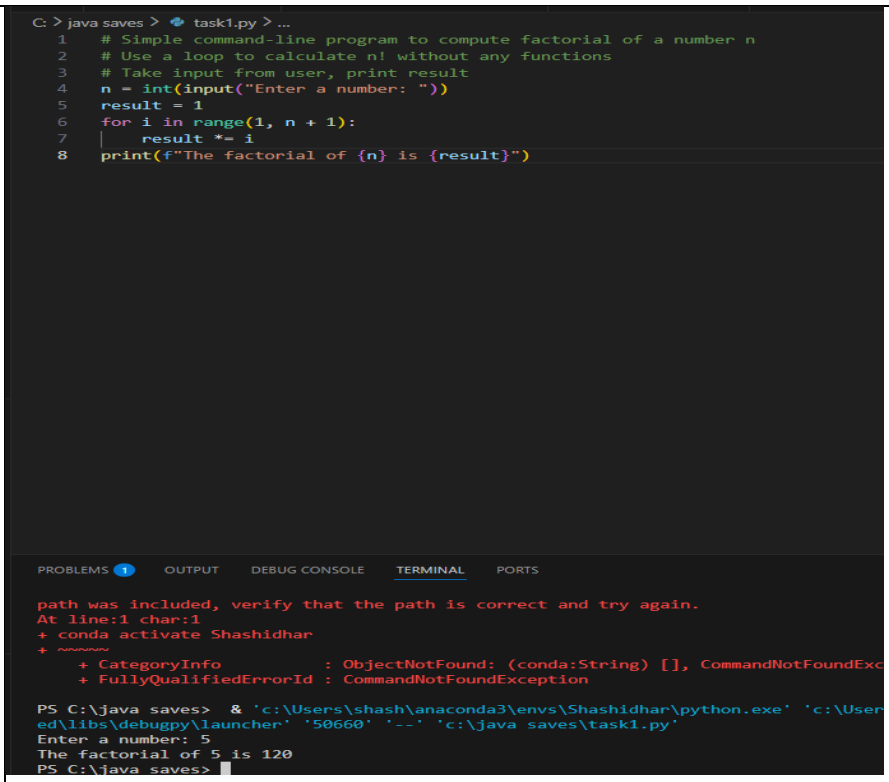
Use GitHub Copilot to generate a Python program that computes a mathematical product-based value (factorial-like logic) directly in the main execution flow, without using any user-defined functions.

- **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

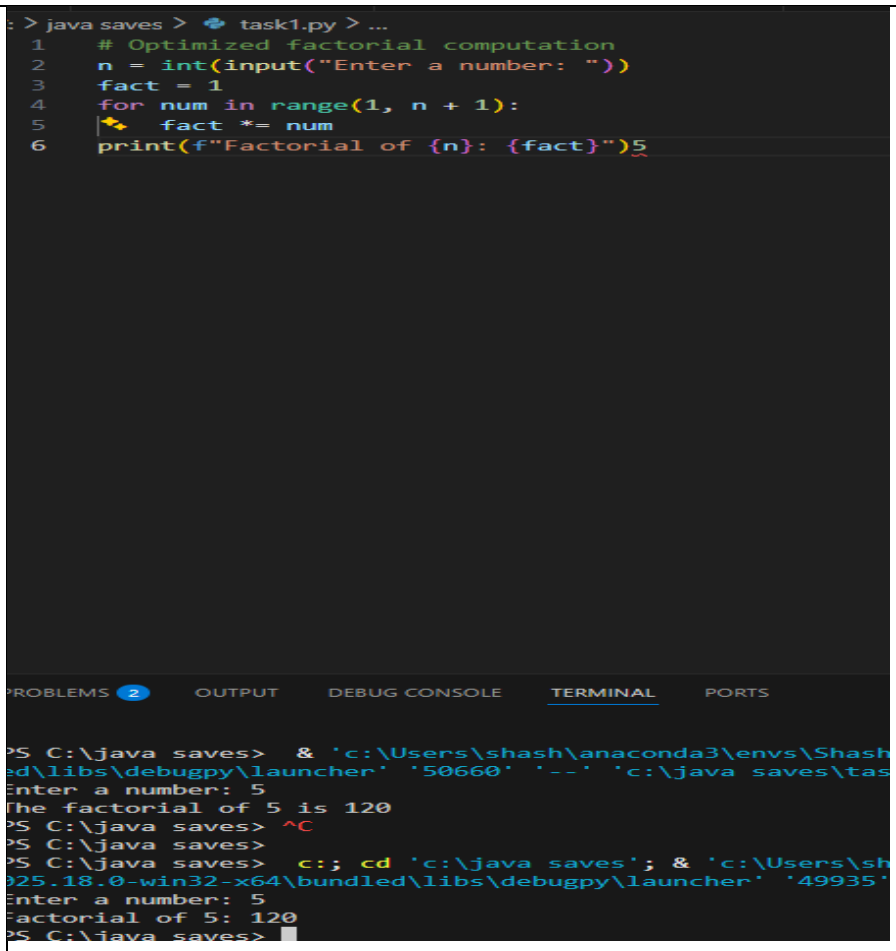
Hint:

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

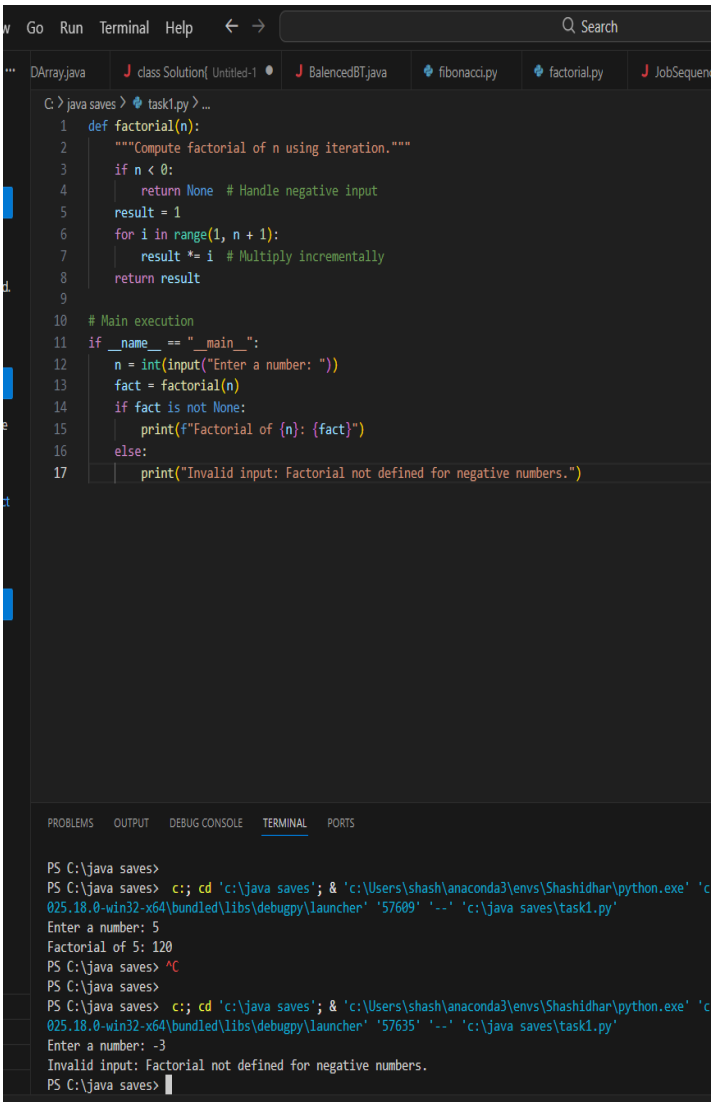
❖ **Constraints**

- 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

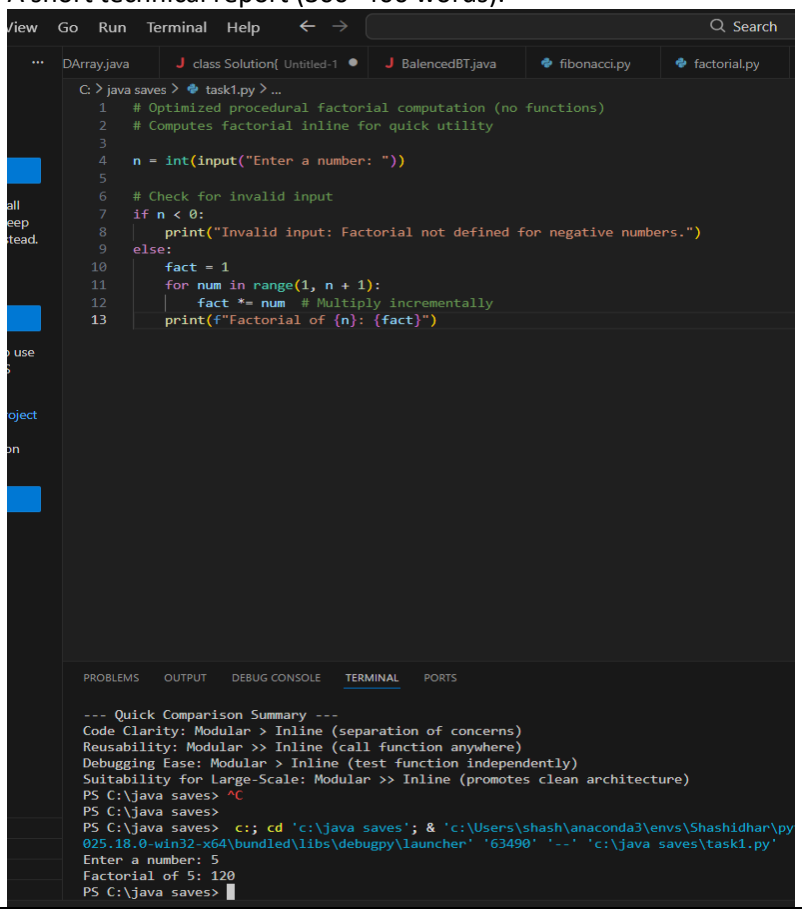
❖ **Expected Deliverables**

Choose **one**:

- A comparison table

  OR

- A short technical report (300–400 words).



DArray.java | class Solution{ Untitled-1 | BalencedBT.java | fibonacci.py | factorial.p

C: > java saves > task1.py > ...

```python
# Modular factorial program using a function for reusability

def factorial(n):
    """
    Compute factorial of n using iteration.
    Handles negative inputs gracefully.
    """
    if n < 0:
        return None  # Handle negative input
    result = 1
    for i in range(1, n + 1):
        result *= i  # Multiply incrementally
    return result

# Main execution block
if __name__ == "__main__":
    n = int(input("Enter a number: "))
    fact = factorial(n)
    if fact is not None:
        print(f"Factorial of {n}: {fact}")
    else:
        print("Invalid input: Factorial not defined for negative numbers.")
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\java saves>
PS C:\java saves>  c:; cd 'c:\java saves'; & 'c:\Users\shash\anaconda3\envs\Shashid
025.18.0-win32-x64\bundled\libs\debugpy\launcher' '63490' '--' 'c:\java saves\task1
Enter a number: 5
Factorial of 5: 120
PS C:\java saves> ^C
PS C:\java saves>
PS C:\java saves>  c:; cd 'c:\java saves'; & 'c:\Users\shash\anaconda3\envs\Shashid
025.18.0-win32-x64\bundled\libs\debugpy\launcher' '63554' '--' 'c:\java saves\task1
Enter a number: 5
Factorial of 5: 120
PS C:\java saves>
```

➢

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**

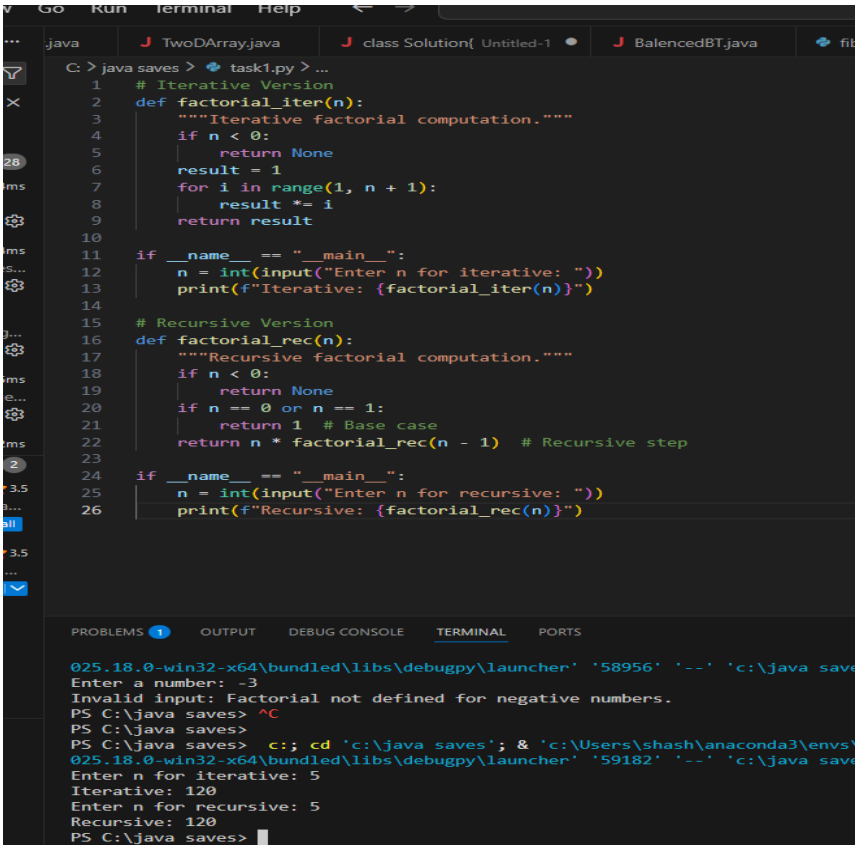
Two AI-generated implementations

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.