

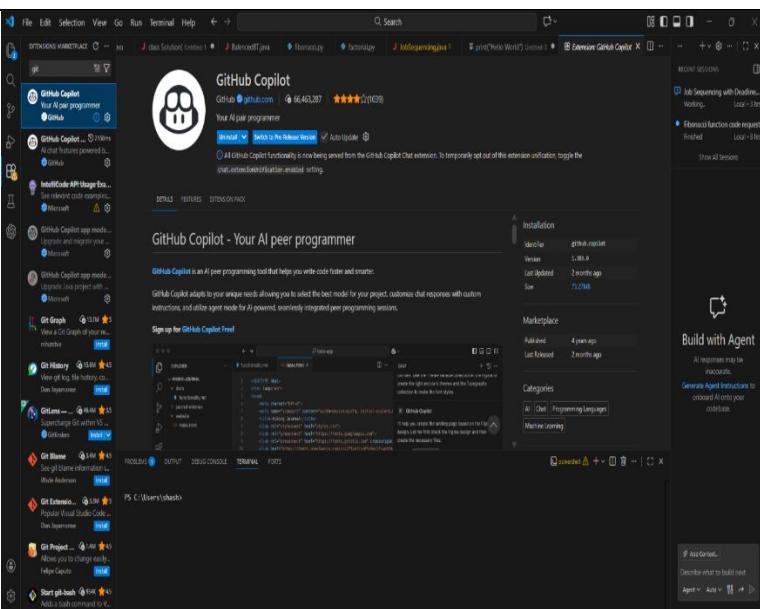
Name:G.Bhagath

H.No:2303A51807

Batch: 26

SCHOOL OF COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE		DEPARTMENT OF COMPUTER SCIENCE ENGINEERING	
Program Name: B. Tech		Assignment Type: Lab	
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

	<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):</p> <p>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. <hr/> <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.	
--	--	--



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?

```
C:\> java saves > task1.py > ...
1  # Simple command-line program to compute factorial of a number n
2  # Use a loop to calculate n! without any functions
3  # Take input from user, print result
4  n = int(input("Enter a number: "))
5  result = 1
6  for i in range(1, n + 1):
7      result *= i
8  print(f"The factorial of {n} is {result}")

PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL PORTS
path was included, verify that the path is correct and try again.
At line:1 char:1
+ conda activate Shashidhar
+ ~~~~~
+ CategoryInfo          : ObjectNotFound: (conda:String) [], CommandNotFoundException
+ FullyQualifiedErrorId : CommandNotFoundException

PS C:\java saves> & 'c:\Users\shash\anaconda3\envs\Shashidhar\python.exe' 'c:\User
ed\libs\debugpy\launcher' '50660' '--' 'c:\java saves\task1.py'
Enter a number: 5
The factorial of 5 is 120
PS C:\java saves>
```

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).
 -
 -

```
: > java saves > task1.py > ...
1 # Optimized factorial computation
2 n = int(input("Enter a number: "))
3 fact = 1
4 for num in range(1, n + 1):
5     fact *= num
6 print(f"Factorial of {n}: {fact}")5

PROBLEMS 2 OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS C:\java saves> & 'c:\Users\shash\anaconda3\envs\Shash
ed\libs\debugpy\launcher' '50660' '--' 'c:\java saves\tas
Enter a number: 5
The factorial of 5 is 120
PS C:\java saves> ^C
PS C:\java saves>
PS C:\java saves> c;; cd 'c:\java saves'; & 'c:\Users\sh
925.18.0-win32-x64\bundled\libs\debugpy\launcher' '49935'
Enter a number: 5
Factorial of 5: 120
PS C:\java saves>
```

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.

-

```

C:\java saves > task1.py ...
1 def factorial(n):
2     """Compute factorial of n using iteration."""
3     if n < 0:
4         return None # Handle negative input
5     result = 1
6     for i in range(1, n + 1):
7         result *= i # Multiply incrementally
8     return result
9
10 # Main execution
11 if __name__ == "__main__":
12     n = int(input("Enter a number: "))
13     fact = factorial(n)
14     if fact is not None:
15         print(f"Factorial of {n}: {fact}")
16     else:
17         print("Invalid input: Factorial not defined for negative numbers.")

PS C:\java saves>
PS C:\java saves> c:\cd 'c:\java saves'; & 'c:\Users\shash\anaconda3\envs\Shashidhar\python.exe' 'c:\java saves\task1.py'
025.18.0-win32-x64\bundled\libs\debugpy\launcher' '57609' '--' 'c:\java saves\task1.py'
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\Shashidhar\python.exe' 'c:\java saves\task1.py'
025.18.0-win32-x64\bundled\libs\debugpy\launcher' '57609' '--' 'c:\java saves\task1.py'
Enter a number: -3
Invalid input: Factorial not defined for negative numbers.
PS C:\java saves>

```

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).

The screenshot shows a terminal window with the following content:

```
C:\> java saves > task1.py> ...
1  # Optimized procedural factorial computation (no functions)
2  # Computes factorial inline for quick utility
3
4  n = int(input("Enter a number: "))
5
6  # Check for invalid input
7  if n < 0:
8      print("Invalid input: Factorial not defined for negative numbers.")
9  else:
10     fact = 1
11     for num in range(1, n + 1):
12         fact *= num # Multiply incrementally
13     print(f"Factorial of {n}: {fact}")

PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS

--- Quick Comparison Summary ---
Code Clarity: Modular > Inline (separation of concerns)
Reusability: Modular >> Inline (call function anywhere)
Debugging Ease: Modular > Inline (test function independently)
Suitability for Large-Scale: Modular >> Inline (promotes clean architecture)
PS C:\java saves> ^C
PS C:\java saves>
PS C:\java saves> c:&; cd 'c:\java saves'; & 'c:\Users\shash\anaconda3\envs\Shashidhar\py
025.18.0-win32-x64\bundled\libs\debugpy\launcher' '63490' '--' 'c:\java saves\task1.py'
Enter a number: 5
Factorial of 5: 120
PS C:\java saves>
```

The image shows a code editor interface with a dark theme. In the top navigation bar, there are tabs for 'DArray.java', 'Solution.py' (which is currently selected), 'Untitled-1', 'BalancedBT.java', 'fibonacci.py', and 'factorial.py'. Below the tabs, the code for 'task1.py' is displayed:

```

C: > java saves > task1.py > ...
1  # Modular factorial program using a function for reusability
2
3  def factorial(n):
4      """
5          Compute factorial of n using iteration.
6          Handles negative inputs gracefully.
7      """
8      if n < 0:
9          return None # Handle negative input
10     result = 1
11     for i in range(1, n + 1):
12         result *= i # Multiply incrementally
13     return result
14
15 # Main execution block
16 if __name__ == "__main__":
17     n = int(input("Enter a number: "))
18     fact = factorial(n)
19     if fact is not None:
20         print(f"Factorial of {n}: {fact}")
21     else:
22         print("Invalid input: Factorial not defined for negative numbers.")

```

Below the code editor is a terminal window showing the execution of the code. The terminal output is as follows:

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS C:\java saves>
PS C:\java saves> c;; cd 'c:\java saves'; & 'c:\Users\shash\anaconda3\envs\Shashidh
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\Shashidh
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.

```

java | J TwoDArray.java | J class Solution{ Untitled-1 | J BalancedBT.java | fib
...
C: > java saves > task1.py ...
1 # Iterative Version
2 def factorial_iter(n):
3     """Iterative factorial computation."""
4     if n < 0:
5         return None
6     result = 1
7     for i in range(1, n + 1):
8         result *= i
9     return result
10
11 if __name__ == "__main__":
12     n = int(input("Enter n for iterative: "))
13     print(f"Iterative: {factorial_iter(n)}")
14
15 # Recursive Version
16 def factorial_rec(n):
17     """Recursive factorial computation."""
18     if n < 0:
19         return None
20     if n == 0 or n == 1:
21         return 1 # Base case
22     return n * factorial_rec(n - 1) # Recursive step
23
24 if __name__ == "__main__":
25     n = int(input("Enter n for recursive: "))
26     print(f"Recursive: {factorial_rec(n)}")

PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL PORTS

025.18.0-win32-x64\bundled\libs\debugpy\launcher' '58956' '--' 'c:\java saves>
Enter a number: -3
Invalid input: Factorial not defined for negative numbers.
PS C:\java saves> ^C
PS C:\java saves>
PS C:\java saves> c:; cd 'c:\java saves'; & 'c:\Users\shash\anaconda3\envs\py
025.18.0-win32-x64\bundled\libs\debugpy\launcher' '59182' '--' 'c:\java saves>
Enter n for iterative: 5
Iterative: 120
Enter n for recursive: 5
Recursive: 120
PS C:\java saves>

```

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.