

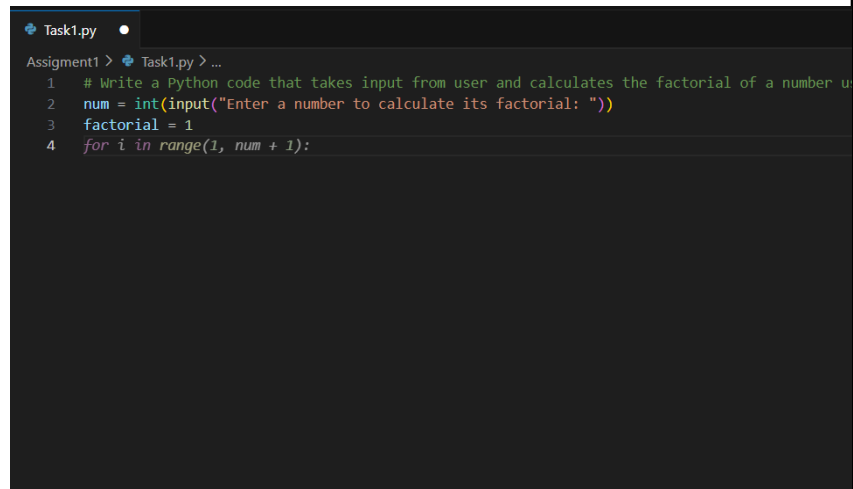
SCHOOL OF COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE		DEPARTMENT OF COMPUTER SCIENCE ENGINEERING	
Program Name: B. Tech		Assignment Type: Lab	Academic Year:2025-2026
Course Coordinator Name		Dr. Rishabh Mittal	
Instructor(s) Name		Mr. S Naresh Kumar	
		Ms. B. Swathi	
		Dr. Sasanko Shekhar Gantayat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

	<ul style="list-style-type: none"><li>● To install and configure GitHub Copilot in Visual Studio Code.</li><li>● To explore AI-assisted code generation using GitHub Copilot.</li><li>● To analyze the accuracy and effectiveness of Copilot's code suggestions.</li><li>● To understand prompt-based programming using comments and code context</li></ul> <p><b>Lab Outcomes (LOs):</b> After completing this lab, students will be able to:</p> <ul style="list-style-type: none"><li>● Set up GitHub Copilot in VS Code successfully.</li><li>● Use inline comments and context to generate code with Copilot.</li><li>● Evaluate AI-generated code for correctness and readability.</li><li>● Compare code suggestions based on different prompts and programming styles.</li></ul> <hr/> <p>Task 0</p> <ul style="list-style-type: none"><li>● Install and configure GitHub Copilot in VS Code. Take screenshots of each step.</li></ul> <p>Expected Output</p> <ul style="list-style-type: none"><li>● Install and configure GitHub Copilot in VS Code. Take screenshots of each step.</li></ul> <hr/> <p>Task 1: AI-Generated Logic Without Modularization (Factorial without Functions)</p> <ul style="list-style-type: none"><li>● <b>Scenario</b> You are building a <b>small command-line utility</b> for a startup intern onboarding task. The program is simple and must be written quickly without modular design.</li><li>● <b>Task Description</b> 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.</li><li>● <b>Constraint:</b><ul style="list-style-type: none"><li>☐ Do not define any custom function</li><li>☐ Logic must be implemented using loops and variables only</li></ul></li><li>● <b>Expected Deliverables</b></li></ul>	
--	--	--

- A working Python program generated with Copilot assistance

```
num = int(input("Enter a number to calculate its factorial: "))
factorial = 1
for i in range(1, num + 1):
    factorial *= i
print(f"The factorial of {num} is {factorial}")
```

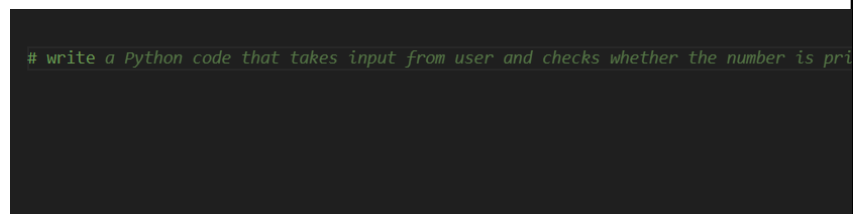
- Screenshot(s) showing:



- The prompt you typed

```
Write a python code using for loop to calculate
the factorial of a given number without using
functions and it should take input from the
user.without using any functions...
```

- Copilot's suggestions



- Sample input/output screenshots

```
● Enter a number to calculate its factorial: 5
The factorial of 5 is 120
```

- Brief reflection (5–6 lines):

An input number `is` given to it.  
The user `is` prompted by `the` program to enter an integer.  
A variable `is` initialized to 1 to hold `the factorial result`.  
The `result is` multiplied by each integer `in` sequence `as` it loops `from` the entered number down to 1.  
The calculated `factorial` value `for the` given number `is` printed out `finally`.

- How helpful was Copilot for a beginner?

Prompt `is` written `in` just seconds `and` it can write code easily. Efficient engagement `is` there `and` learning `is` greatly helped by it.

- Did it follow best practices automatically?

Yes, it automatically implemented several coding best practices

---

## 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

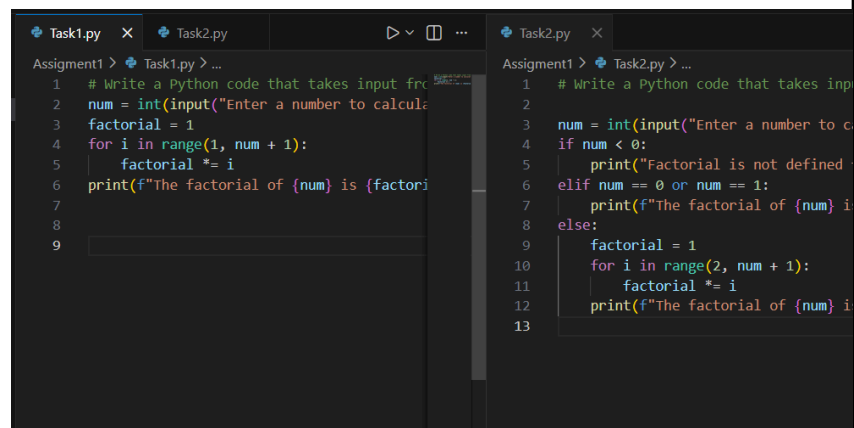
### □ Original AI-generated code

```
num = int(input("Enter a number to calculate its  
factorial: "))  
factorial = 1  
for i in range(1, num + 1):  
    factorial *= i  
print(f"The factorial of {num} is {factorial}")
```

### □ Optimized version of the same code

```
num = int(input("Enter a number to calculate its  
factorial: "))  
if num < 0:  
    print("Factorial is not defined for negative  
numbers.")  
elif num == 0 or num == 1:  
    print(f"The factorial of {num} is 1")  
else:  
    factorial = 1  
    for i in range(2, num + 1):  
        factorial *= i  
    print(f"The factorial of {num} is  
{factorial}")
```

### □ Side-by-side comparison



□ Written explanation:

▪ What was improved?

```
In the optimized code it first checks for negative numbers and if the user enters 1 or 0 it runs faster than the original code.
```

▪ Why the new version is better (readability, performance, maintainability).

```
using if-else blocks for edge cases like 0 and 1 it makes faster to run compared to original code
```

---

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

```
def calculate_factorial(num):  
    if num < 0:  
        return "Factorial is not defined for negative numbers."  
    elif num == 0 or num == 1:  
        return 1  
    else:  
        factorial = 1
```

```

        for i in range(2, num + 1):
            factorial *= i
        return factorial
num = int(input("Enter a number to calculate its
factorial: "))
result = calculate_factorial(num)
print(f"The factorial of {num} is {result}")

```

#### □ Screenshots showing:

- Prompt evolution
- Copilot-generated function logic

```

Assignment1 > Task3.py > ...
1  # Write a Python code that takes input from user and calculates the factorial of
2  def calculate_factorial(num):
3      if num < 0:
         return "Factorial is not defined for negative numbers."
         elif num == 0 or num == 1:
             return 1
         else:
             factorial = 1
             for i in range(2, num + 1):
                 factorial *= i
             return factorial
4

```

#### □ Sample inputs/outputs

```

Enter a number to calculate its factorial: -6
The factorial of -6 is Factorial is not defined for negative numbers.

```

#### □ Short note:

- How modularity improves reusability  
modules that can be reused across projects without rewriting, reducing duplication and enhancing maintainability.

### 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

```
==== PROCEDURAL VERSION ====
Enter a number to calculate its factorial: 5
The factorial of 5 is 120
⌚ Execution Time (Procedural): 2207.452900 ms

=====

=== MODULAR VERSION ===
Enter a number to calculate its factorial: 5
The factorial of 5 is 120
⌚ Execution Time (Modular): 1376.439900 ms

=====

PERFORMANCE ANALYSIS
=====
Procedural Version: 2207.452900 ms
Modular Version:    1376.439900 ms

Modular was faster by 831.013000 ms
```

Feature	Non-Function Approach (Task 1)	Function-Based Approach (Task 2)
Structure	<b>Monolithic:</b> Logic, input, and output are mixed in one global block.	<b>Modular:</b> Logic is encapsulated in a specific function, separating it from I/O.
Reusability	<b>Low:</b> Code must be copied and pasted to be used elsewhere.	<b>High:</b> Function can be imported and called by any other script or module.
Error Handling	<b>Basic:</b> Uses simple <code>print</code> statements (e.g., "Error: Negative number").	<b>Robust:</b> Uses Exceptions ( <code>raise ValueError</code> ) allowing programs to catch and manage errors.
Testing	<b>Difficult:</b> Requires manual user input for every test case.	<b>Easy:</b> Can be automated using Unit Tests to verify logic instantly.
Maintainability	<b>Poor:</b> Changes to logic might break the input/output flow.	<b>Excellent:</b> You can upgrade the math logic without touching the rest of the code.

**OR**

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

#### ❖ 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

##### *Iterative*

```
def factorial_iterative(n):  
    if n < 0:  
        return "Factorial is not defined for negative  
numbers."  
    elif n == 0 or n == 1:  
        return 1  
    else:  
        result = 1  
        for i in range(2, n + 1):  
            result *= i  
        return result  
  
num = int(input("Enter a number to calculate its  
factorial: "))  
factorial_result = factorial_iterative(num)  
print(f"The factorial of {num} is  
{factorial_result}")
```

### *recursive*

```
def factorial_recursive(n):  
    if n < 0:  
        return "Factorial is not defined for negative  
numbers."  
    elif n == 0 or n == 1:  
        return 1  
    else:  
        return n * factorial_recursive(n - 1)  
num = int(input("Enter a number to calculate its  
factorial: "))  
factorial_result = factorial_recursive(num)  
print(f"The factorial of {num} is  
{factorial_result}")
```

#### □ Stack usage

an iterative one using a loop for factorial calculation and a recursive one..both handling negatives and base cases with user input and output.

#### □ Performance implications

**Time Complexity:** Both iterative and recursive approaches are  $O(n)$   
**Space Complexity:** Iterative uses  $O(1)$  and Recursive uses  $O(n)$  space due to call stack.

#### □ When recursion is *not* recommended.

when input  $n$  is large as it can cause stack overflow due to deep call stacks

---

### 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.

	<b>Note: Report should be submitted as a word document for all tasks in a single document with prompts, comments &amp; code explanation, and output and if required, screenshots.</b>	
--	---	--