

# AI ASSISTED CODING

## LAB ASSIGNMENT-1

Name: JILLELA AKSHAYA

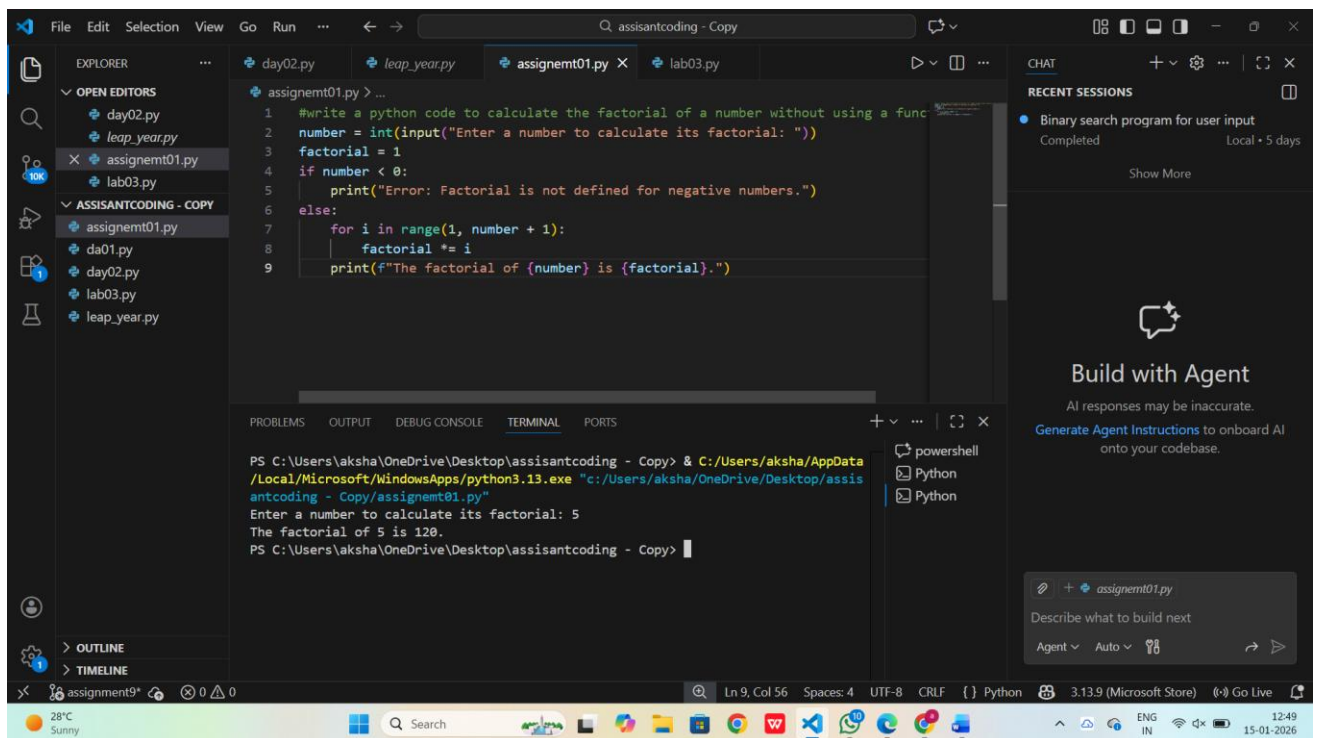
HT.NO: 2303A51629

Batch: 22

**Question-1:** AI-Generated Logic Without Modularization (Factorial without Functions)

**Prompt:** # generate a code to get factorial of a number without using functions

Code and Output Screenshot:



The screenshot displays the Visual Studio Code (VS Code) interface. The Explorer panel on the left shows a project named 'ASSISANTCODING - COPY' containing several Python files, with 'assignment01.py' selected. The main editor window shows the code for 'assignment01.py', which is a Python script to calculate the factorial of a number without using functions. The code includes a comment, an input prompt, a loop to calculate the factorial, and an error handling for negative numbers. The terminal at the bottom shows the command to run the script, the input '5', and the output 'The factorial of 5 is 120'. The right sidebar shows the 'CHAT' panel with 'RECENT SESSIONS' and a 'Build with Agent' section.

```
1 #write a python code to calculate the factorial of a number without using a func
2 number = int(input("Enter a number to calculate its factorial: "))
3 factorial = 1
4 if number < 0:
5     print("Error: Factorial is not defined for negative numbers.")
6 else:
7     for i in range(1, number + 1):
8         factorial *= i
9     print(f"The factorial of {number} is {factorial}.")
```

PS C:\Users\aksha\OneDrive\Desktop\assisantcoding - Copy> & C:/Users/aksha/AppData/Local/Microsoft/WindowsApps/python3.13.exe "c:/Users/aksha/OneDrive/Desktop/assisantcoding - Copy/assignment01.py"

Enter a number to calculate its factorial: 5

The factorial of 5 is 120.

PS C:\Users\aksha\OneDrive\Desktop\assisantcoding - Copy>

**Observation:**

GitHub Copilot was helpful for me being a beginner, it helped me with the right type of logic in loops. It shortened the time to consider syntax and basic control flow logic. Copilot made the things easy like initializing a variable properly and choosing good loop condition expressions. For new user it works more like an intelligent code assistant than an educator. Finally it improves confidence and quickness and must be done while also learning base skills.

## Question-2: AI Code Optimization & Cleanup (Improving Efficiency)

**Prompt:** # generate an optimized version code of Factorial of a given Number.

Code and Output Screenshot:

The screenshot displays the Visual Studio Code editor with a file explorer on the left showing a project named 'ASSISANTCODING - COPY'. The main editor window shows a Python file named 'assignment01.py' with the following code:

```
11 #task2
12 #generate a python code to calculate the factorial of a number without using an
13 number = int(input("Enter a number to calculate its factorial: "))
14 factorial = 1
15 if number < 0:
16     print("Error: Factorial is not defined for negative numbers.")
17 else:
18     for i in range(2, number + 1):
19         factorial *= i
20     print(f"The factorial of {number} is {factorial}.")
```

The bottom panel shows the TERMINAL output:

```
PS C:\Users\aksha\OneDrive\Desktop\assisantcoding - Copy> & C:/Users/aksha/AppData/Local/Microsoft/WindowsApps/python3.13.exe "c:/Users/aksha/OneDrive/Desktop/assisantcoding - Copy/assignment01.py"
Enter a number to calculate its factorial: 5
The factorial of 5 is 120.
PS C:\Users\aksha\OneDrive\Desktop\assisantcoding - Copy> & C:/Users/aksha/AppData/Local/Microsoft/WindowsApps/python3.13.exe "c:/Users/aksha/OneDrive/Desktop/assisantcoding - Copy/assignment01.py"
Enter a number to calculate its factorial: 6
The factorial of 6 is 720.
Enter a number to calculate its factorial: 
```

The right sidebar shows the CHAT panel with a 'RECENT SESSIONS' list and a 'Build with Agent' section.

## Observation:

Using GitHub Copilot for the optimized factorial code produced a more efficient and well-structured solution. The optimized logic reduced unnecessary computations and improved performance. Copilot suggested clear function design and concise implementation, making the code easy to read and reuse. Inline comments helped explain the optimized approach, encouraging good programming practices.

## Question-3: Modular Design Using AI Assistance (Factorial with Functions)

**Prompt:** # generate a code to get factorial of a number with using functions

Code and Output Screenshot:

```

24 #task3
25 #write a python code to calculate the factorial of a number with using function
26 def factorial(number):
27     factorial = 1
28     if number < 0:
29         return "Error: Factorial is not defined for negative numbers."
30     else:
31         for i in range(2, number + 1):
32             factorial *= i
33     return f"The factorial of {number} is {factorial}."

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```

The factorial of 6 is 720.
Enter a number to calculate its factorial: & C:/Users/aksha/AppData/Local/Microsoft/WindowsApps/python3.13.exe "c:/Users/aksha/OneDrive/Desktop/assisantcoding - Copy/assinent01.py"
Traceback (most recent call last):
  File "c:/Users/aksha/OneDrive/Desktop/assisantcoding - Copy/assinent01.py", line 13, in <module>
    number = int(input("Enter a number to calculate its factorial: "))
ValueError: invalid literal for int() with base 10: '& C:/Users/aksha/AppData/Local/Microsoft/WindowsApps/python3.13.exe "c:/Users/aksha/OneDrive/Desktop/assisantcoding - Copy/assinent01.py"'
PS C:/Users/aksha/OneDrive/Desktop/assisantcoding - Copy> & C:/Users/aksha/AppData/Local/Microsoft/WindowsApps/python3.13.exe "c:/Users/aksha/OneDrive/Desktop/assisantcoding - Copy/assinent01.py"
Enter a number to calculate its factorial: 7
The factorial of 7 is 5040.
Enter a number to calculate its factorial:

```

## Observation:

Using GitHub Copilot for a modular design made the code more structured and easier to understand. Copilot suggested meaningful function names and clear parameters, which improves readability. The separation of logic into a function allows the same factorial computation to be reused across multiple programs. Inline comments generated by Copilot helped clarify each step of the logic for beginners. Copilot naturally encourages good programming practices through function-based design.

## Question-4: Comparative Analysis – Procedural vs Modular AI Code (With vs Without Functions)

Prompt: No prompt

Code and Output Screenshot: No code

Comparison Table:

Features	Without Functions	With Functions
Code Structure	Simple and linear	Organized and Modular
Length of code	Shorter	Slightly long

Reusability	Cannot be reduced easily	Can be reused multiple times
Maintenance	Harder for large programs	Easy to debug and modify
Calling Mechanism	Runs directly	Function is called

### Technical Report:

or **logic clarity**, a procedural version (without functions) feels simple and direct for very small programs because everything is written in one continuous flow. Beginners can easily follow the steps from input to output. But as the program grows, this style quickly becomes messy and harder to understand. A modular version (using functions) improves clarity by putting the main logic into well-named functions, so anyone reading the code can understand its purpose at a glance.

For **debugging**, procedural code is easy to fix when the program is small. But in longer scripts, finding errors becomes confusing and time-consuming. Modular code makes debugging much easier because problems can usually be traced back to a specific function. This allows developers to test and fix parts of the program independently.

Regarding **AI dependency risk**, both approaches have risks if someone blindly trusts Copilot's suggestions. However, modular code slightly reduces this risk .

### Question-5: AI-Generated Iterative vs Recursive Thinking

Iterative:

**Prompt:** # generate a code to get factorial iteratively

Code and Output Screenshots:

The screenshot shows a VS Code editor with a file named `assignemt01.py` open. The code defines an iterative factorial function. The terminal output shows the program running successfully for the input 7, resulting in 5040. A subsequent attempt to run the program with input 8 results in a `ValueError: invalid literal for int() with base 10: '& C:/Users/aksha/AppData/Local/Microsoft/WindowsApps/python3.13.exe' c:/Users/aksha/OneDrive/Desktop/assisantcoding - Copy/assigne` error.

```
#task 5
# generate a code to get factorial iteratively
def factorial_iterative(number):
    result = 1
    for i in range(2, number + 1):
        result *= i
    return result

number = int(input("Enter a number to calculate its factorial: "))
print(f"The factorial of {number} is {factorial_iterative(number)}.")
```

Terminal Output:

```
The factorial of 7 is 5040.
Enter a number to calculate its factorial: & C:/Users/aksha/AppData/Local/Microsoft/WindowsApps/python3.13.exe "c:/Users/aksha/OneDrive/Desktop/assisantcoding - Copy/assignemt01.py"
Traceback (most recent call last):
  File "c:/Users/aksha/OneDrive/Desktop/assisantcoding - Copy/assignemt01.py", line 13, in <module>
    number = int(input("Enter a number to calculate its factorial: "))
ValueError: invalid literal for int() with base 10: '& C:/Users/aksha/AppData/Local/Microsoft/WindowsApps/python3.13.exe' c:/Users/aksha/OneDrive/Desktop/assisantcoding - Copy/assigne
PS C:/Users/aksha/OneDrive/Desktop/assisantcoding - Copy> & C:/Users/aksha/AppData/Local/Microsoft/WindowsApps/python3.13.exe "c:/Users/aksha/OneDrive/Desktop/assisantcoding - Copy/assignemt01.py"
Enter a number to calculate its factorial: 8
The factorial of 8 is 40320.
Enter a number to calculate its factorial: 
```

Recursive:

Prompt: # generate a code to get factorial recursively

Code and Output Screenshots:

The screenshot shows a VS Code editor with a file named `assignemt01.py` open. The code defines a recursive factorial function. The terminal output shows the program running successfully for the input 7, resulting in 5040. A subsequent attempt to run the program with input 8 results in a `ValueError: invalid literal for int() with base 10: '& C:/Users/aksha/AppData/Local/Microsoft/WindowsApps/python3.13.exe' c:/Users/aksha/OneDrive/Desktop/assisantcoding - Copy/assignemt01.py'` error.

```
# generate a code to get factorial recursively
def factorial_recursive(number):
    if number < 0:
        return "Error: Factorial is not defined for negative numbers."
    elif number == 0 or number == 1:
        return 1
    else:
        return number * factorial_recursive(number - 1)

number = int(input("Enter a number to calculate its factorial: "))
print(f"The factorial of {number} is {factorial_recursive(number)}.")
```

Terminal Output:

```
The factorial of 7 is 5040.
Enter a number to calculate its factorial: & C:/Users/aksha/AppData/Local/Microsoft/WindowsApps/python3.13.exe "c:/Users/aksha/OneDrive/Desktop/assisantcoding - Copy/assignemt01.py"
Traceback (most recent call last):
  File "c:/Users/aksha/OneDrive/Desktop/assisantcoding - Copy/assignemt01.py", line 13, in <module>
    number = int(input("Enter a number to calculate its factorial: "))
ValueError: invalid literal for int() with base 10: '& C:/Users/aksha/AppData/Local/Microsoft/WindowsApps/python3.13.exe' c:/Users/aksha/OneDrive/Desktop/assisantcoding - Copy/assignemt01.py'
PS C:/Users/aksha/OneDrive/Desktop/assisantcoding - Copy> & C:/Users/aksha/AppData/Local/Microsoft/WindowsApps/python3.13.exe "c:/Users/aksha/OneDrive/Desktop/assisantcoding - Copy/assignemt01.py"
Enter a number to calculate its factorial: 4
The factorial of 4 is 24.
Enter a number to calculate its factorial: 
```

Execution Flow Explanation:



In the **iterative approach**, the program starts with a value of 1 and uses a loop to multiply it with every number from 1 up to the given input. The result is updated step by step inside the same loop until the final factorial value is obtained.

In the **recursive approach**, the function solves the problem by breaking it into smaller parts. Each function call depends on the result of the next call, continuing until it reaches a base case (0 or 1). After reaching the base case, the function calls return one by one, multiplying the values together to produce the final factorial.

### Comparative Analysis:

#### **Readability:**

The iterative approach is usually easier for beginners to read and understand because the flow of execution is straightforward. Recursive code, although mathematically elegant, can be harder to follow since the function keeps calling itself, which makes tracing the execution more complex.

#### **Stack Usage:**

Iterative implementations use constant memory because they rely on a single loop. In contrast, recursive implementations consume extra stack memory for every function call, which increases memory usage.

#### **Performance Implications:**

Iterative solutions are generally faster and more memory-efficient. Recursive solutions introduce overhead due to repeated function calls and stack operations, which can slow down execution.

#### **When Recursion Is Not Recommended:**

Recursion should be avoided when dealing with very large inputs because it can cause stack overflow. It is also not ideal for performance-critical or memory-limited applications, and when the problem logic does not naturally suit a recursive approach.