

# **School of Computer Science and Artificial Intelligence**

---

## **Lab Assignment # 1.2**

---

**Program : B. Tech (CSE)**

**Specialization :AIML**

**Course Title : AI Assisted Coding**

**Course Code : 23CS002PC304**

**Semester : VI**

**Academic Session : 2025-2026**

**Name of Student : V.Mokshitha**

**Enrollment No. : 2303A52122**

**Batch No. : 33**

**Date :09/01/26**

# TASK \_01

The screenshot shows a Google Colab notebook titled "LAB-01.ipynb". The code cell contains the following Python script:

```
# Get the number from the user
num_str = input("Enter a non-negative integer: ")

try:
    num = int(num_str)
    if num < 0:
        print("Factorial is not defined for negative numbers.")
    elif num == 0:
        print(f"The factorial of 0 is {factorial}")
    else:
        factorial = 1
        # Calculate factorial using a for loop
        for i in range(1, num + 1):
            factorial *= i
        print(f"The factorial of {num} is {factorial}")
except ValueError:
    print("Invalid input. Please enter an integer.")

*** Enter a non-negative integer: 5
The factorial of 5 is 120
```

To the right of the code cell is a Gemini AI sidebar. The user asks: "write a python program to calculate factorial without using functions use loops and variables only". Gemini responds with a Python program that calculates the factorial of 5 using a for loop and variables only. The sidebar also includes options to test the code with edge cases, explain efficiency, show how to calculate using recursion, and build a model.

## EXPLANATION:

- Input:** It prompts the user to enter a non-negative integer.
- Validation:** It checks if the input is a valid integer and if it's non-negative.
- Initialization:** It sets factorial to 1, as  $0!$  (zero factorial) is 1.
- Calculation:** It uses a for loop to multiply factorial by each number from 1 up to the input number.
- Output:** Finally, it prints the calculated factorial of the given number.

## TASK\_02:

The screenshot shows a Google Colab notebook titled "LAB-01.ipynb". The code cell contains a script to calculate the factorial of a non-negative integer. The script includes comments for readability and handles edge cases like negative numbers and zero. The output shows the factorial of 5 is 120. To the right of the code cell is a "Gemini" sidebar with AI-generated comments and suggestions.

```
# Get the number from the user as a string
num_str = input("Enter a non-negative integer: ")

try:
    # Convert the input string to an integer
    num = int(num_str)

    # Handle negative numbers, as factorial is not defined for them
    if num < 0:
        print("Factorial is not defined for negative numbers.")
    # Handle the base case for 0 factorial
    elif num == 0:
        factorial_result = 1 # Factorial of 0 is 1
        print("The factorial of 0 is {factorial_result}")
    # Calculate factorial for positive integers
    else:
        factorial_result = 1
        # Use a for loop to multiply numbers from 1 to num
        for i in range(1, num + 1):
            factorial_result *= i
        print(f"The factorial of {num} is {factorial_result}")

    # Catch ValueError if the input is not a valid integer
except ValueError:
    print("Invalid input. Please enter an integer.")

... Enter a non-negative integer: 5
The factorial of 5 is 120
```

**Gemini**

- optimize this code to improve readability and efficiency
- I've added some comments to the existing code to improve its readability. The current loop-based approach is already quite efficient for calculating factorials without using dedicated functions.

The code executed successfully with the added comments. The factorial of 5 is still correctly calculated as 120. The changes primarily aimed at making the code easier to understand.

Test the code with edge cases like 0 or a larger number  
Explain the efficiency of the current factorial calculation  
Show me how to calculate factorial using recursion

What can I help you build?

Gemini can make mistakes so double-check it and use code with caution. Learn more

## Explanation

- Better variable names
- Cleaner output
- More readable

## TASK\_03:

The screenshot shows a Google Colab notebook titled "LAB-01.ipynb". The code cell contains a Python function named `calculate\_factorial` that calculates the factorial of a non-negative integer. It includes error handling for negative numbers and invalid input. The output shows the factorial of 5 is 120. To the right, the Gemini AI interface is open with a query to write a Python program to calculate factorial using a function. Gemini provides a sample code and notes that it calculated the factorial of 5 as 120 using the provided function.

```
def calculate_factorial(n):
    """
    Calculates the factorial of a non-negative integer.
    """
    if n < 0:
        return "Factorial is not defined for negative numbers."
    elif n == 0:
        return 1
    else:
        factorial_result = 1
        for i in range(1, n + 1):
            factorial_result *= i
        return factorial_result

# Get the number from the user
num_str = input("Enter a non-negative integer: ")

try:
    num = int(num_str)
    result = calculate_factorial(num)

    if isinstance(result, str):
        print(result)
    else:
        print(f"The factorial of {num} is {result}")
except ValueError:
    print("Invalid input. Please enter an integer.")

... Enter a non-negative integer: 5
The factorial of 5 is 120
```

Gemini

S: write a python program to calculate factorial using a function  
◆ Okay, here's a Python program that calculates the factorial using a function:  
The code cell executed successfully. It calculated the factorial of 5 as 120, using the `calculate_factorial` function. This demonstrates how to implement the factorial calculation within a function.

## **EXPLANATION:**

Using functions improves reusability.

The same function can be used in many programs.

Code becomes cleaner and easier to maintain.

## TASK\_04:

### **Comparative Analysis – Procedural vs Modular AI Code**

#### ***Procedural (Without Functions) vs Modular (With Functions)***

In Task 1, the factorial program was written using a procedural approach, where all the logic was implemented directly in the main execution flow without using any user-defined functions. In Task 3, the same logic was rewritten using a modular approach by creating a

separate function to calculate the factorial. Both approaches produce the same output, but they differ significantly in terms of design quality and usability.

### **Logic Clarity:**

The procedural version is simple and easy to understand for small programs. However, as the program grows, the logic becomes harder to follow because everything is written in one place. In contrast, the modular version separates the factorial logic into a function, making the code more organized and easier to read.

### **Reusability:**

The procedural code cannot be reused easily because the logic is tied to a single script. The modular version allows the factorial function to be reused in multiple programs without rewriting the same code, which saves time and effort.

### **Debugging Ease:**

Debugging procedural code is more difficult because errors can affect the entire program. In modular code, each function can be tested separately, making it easier to find and fix errors.

### **Suitability for Large Projects:**

Procedural code is suitable only for small, simple programs. For large projects, modular code is preferred because it supports better structure, teamwork, and maintenance.

### **AI Dependency Risk:**

When using AI tools like Google Colab, procedural code may be generated quickly but often lacks proper structure. Modular code encourages better design practices, even when AI is used. This reduces the risk of poor-quality code.

### **Conclusion:**

While procedural programming is useful for quick tasks and learning basics, modular programming is more efficient, reusable, and suitable for real-world software development. Using functions improves clarity, maintainability, and scalability, making modular code the better choice for professional projects.

## TASK\_05:

The screenshot shows a Google Colab notebook titled "LAB-01.ipynb". In the code cell, there is a recursive factorial function and a user input section. The output shows the factorial of 5 is 120. To the right, the Gemini AI interface provides a response about the recursive factorial code.

```

def factorial_recursive(n):
    """
    Calculates the factorial of a non-negative integer using recursion.
    """
    if n < 0:
        return "Factorial is not defined for negative numbers."
    elif n == 0:
        return 1 # Base case: factorial of 0 is 1
    else:
        # Recursive case: n! = n * (n-1)!
        return n * factorial_recursive(n - 1)

# Get number from user and demonstrate recursive factorial
num_str_rec = input("Enter a non-negative integer for recursive factorial: ")
try:
    num_rec = int(num_str_rec)
    rec_result = factorial_recursive(num_rec)
    if isinstance(rec_result, str):
        print(rec_result)
    else:
        print(f"The recursive factorial of {num_rec} is {rec_result}")
except ValueError:
    print("Invalid input. Please enter an integer.")

```

Output: Enter a non-negative integer for recursive factorial: 5  
The recursive factorial of 5 is 120

Gemini response:

- write factorial using iteration and recursion
- Certainly! Here are Python programs demonstrating factorial calculation using both iteration and recursion.

The screenshot shows a Google Colab notebook titled "LAB-01.ipynb". In the code cell, there is an iterative factorial function and a user input section. The output shows the factorial of 5 is 120. To the right, the Gemini AI interface provides a response about the iterative factorial code.

```

def factorial_iterative(n):
    """
    Calculates the factorial of a non-negative integer using iteration.
    """
    if n < 0:
        return "Factorial is not defined for negative numbers."
    elif n == 0:
        return 1
    else:
        result = 1
        for i in range(1, n + 1):
            result *= i
        return result

# Get number from user and demonstrate iterative factorial
num_str_iter = input("Enter a non-negative integer for iterative factorial: ")
try:
    num_iter = int(num_str_iter)
    iter_result = factorial_iterative(num_iter)
    if isinstance(iter_result, str):
        print(iter_result)
    else:
        print(f"The iterative factorial of {num_iter} is {iter_result}")
except ValueError:
    print("Invalid input. Please enter an integer.")

```

Output: Enter a non-negative integer for iterative factorial: 5  
The iterative factorial of 5 is 120

Gemini response:

- write factorial using iteration and recursion
- Certainly! Here are Python programs demonstrating factorial calculation using both iteration and recursion.

## **Explanation**

- Iterative uses loop
- Recursive calls itself
- Recursion uses more memory
- Iterative is faster