

# **School of ComputerScience and Artificial Intelligence**

---

## **Lab Assignment # 1.2**

---

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

**Specialization :AIML**

**Course Title : AI Assisted Coding**

**Course Code : 23CS002PC304**

**Semester : VI**

**Enrollment No. : 2303A52131**

**Batch No. : 34**

**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:

S write a python program to calculate factorial without using functions use loops and variables only

Gemini responds:

Sure, here's a Python program to calculate the factorial of a number using only loops and variables:

```
The code successfully calculated the factorial of 5, which is 120. The program correctly handled the input and produced the expected output.
```

Below the response are three buttons: "Test the code with edge cases like 0 or a larger number", "Explain the efficiency of the current factorial calculation", and "Show me how to calculate factorial using recursion".

At the bottom of the sidebar, it says "What can I help you build?" and "Gemini 2.5 Flash". A note at the bottom states "Gemini can make mistakes so double-check it and use code with caution. [Learn more](#)".

## EXPLANATION:

- 1. Input:** It prompts the user to enter a non-negative integer.
- 2. Validation:** It checks if the input is a valid integer and if it's non-negative.
- 3. Initialization:** It sets factorial to 1, as  $0!$  (zero factorial) is 1.
- 4. Calculation:** It uses a for loop to multiply factorial by each number from 1 up to the input number.
- 5. 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 Python script for calculating 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, a Gemini AI sidebar provides feedback on the code's readability and efficiency.

```
# 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.")
```

**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.

What can I help you build?

+ Gemini 2.5 Flash ▶

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 Jupyter Notebook in Google Colab. 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 user input validation. The AI panel on the right, titled "Gemini", provides a response to a query about writing a Python program to calculate factorial using a function. It includes a generated code snippet, a success message, and several interactive buttons for further exploration.

```
[1]: 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
```

## **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 screenshots show two Jupyter notebooks running on Google Colab. The top screenshot displays the 'Factorial using Recursion' notebook, which contains the following 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_rec = int(input("Enter a non-negative integer for recursive factorial: "))
try:
    rec_result = factorial_recursive(num_rec)
    print(rec_result)
except ValueError:
    print("The recursive factorial of (num_rec) is (rec_result)")

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

The bottom screenshot displays the 'Factorial using Iteration' notebook, which contains the following 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_iter = int(input("Enter a non-negative integer for iterative factorial: "))
try:
    iter_result = factorial_iterative(num_iter)
    print(iter_result)
except ValueError:
    print("The iterative factorial of (num_iter) is (iter_result)")

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

In both notebooks, the Gemini AI sidebar provides context and assistance for the code, such as explaining the concepts of iteration and recursion, comparing iterative and recursive performance, and suggesting ways to help build the function.

## **Explanation**

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