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

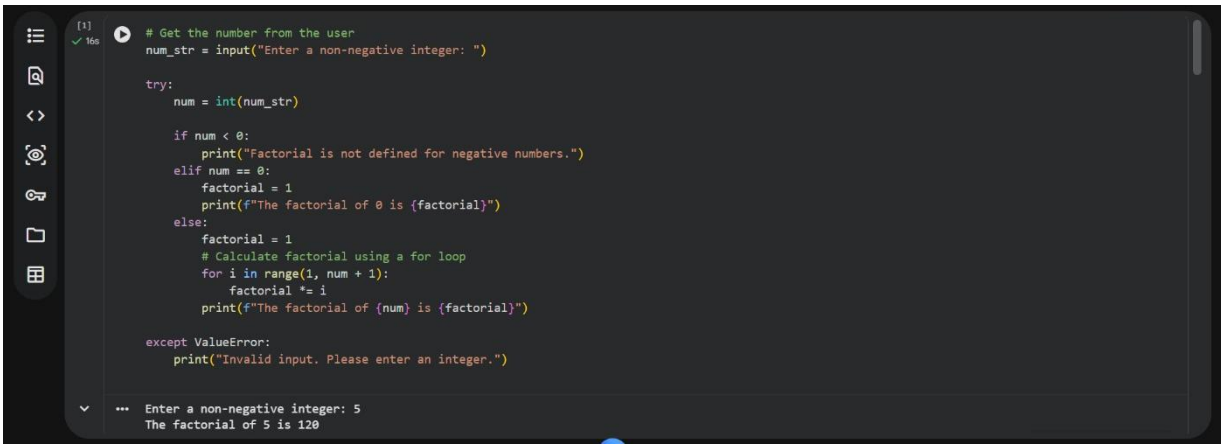
Name of Student : ENDLA ESHWAR PRADAD

Enrollment No. : 2303A52377

Batch No. : 33

Date :09/01/26

TASK _01



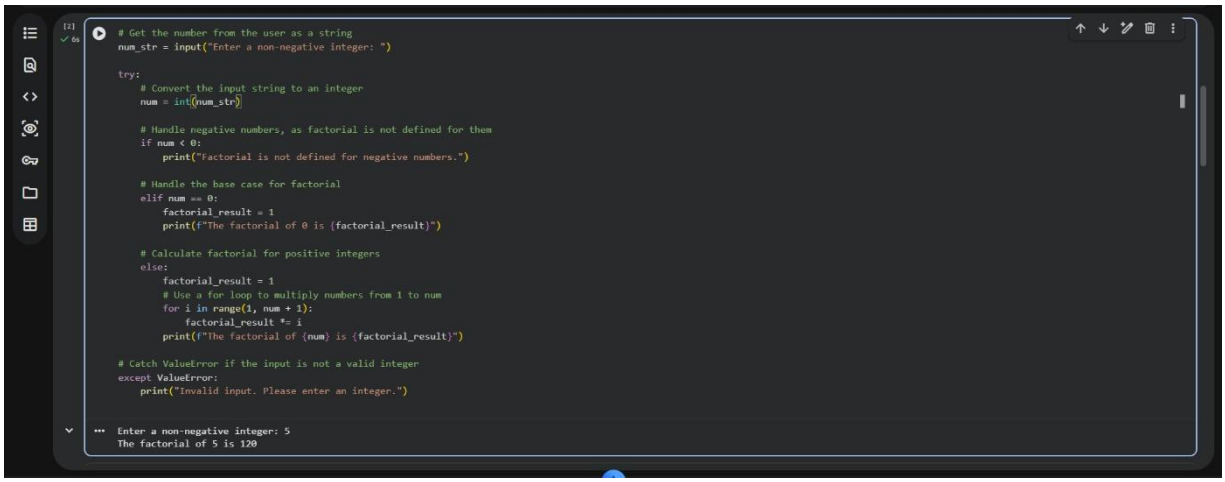
```
# Get the number from the user
num_str = input("Enter a non-negative integer: ")

try:
    num = int(num_str)

    if num < 0:
        print("Factorial is not defined for negative numbers.")
    elif num == 0:
        factorial = 1
        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
```

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:



```
# 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 factorial
    elif num == 0:
        factorial_result = 1
        print(f"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.")
```

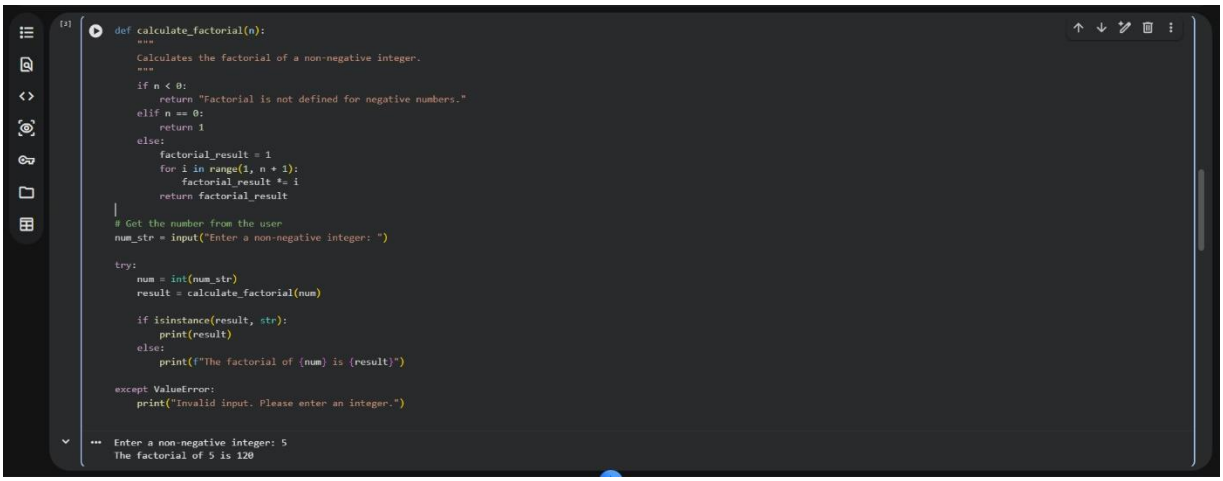
```
Enter a non-negative integer: 5
The factorial of 5 is 120
```

Explanation

- Better variable names
- Cleaner output

- More readable

TASK_03:



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.

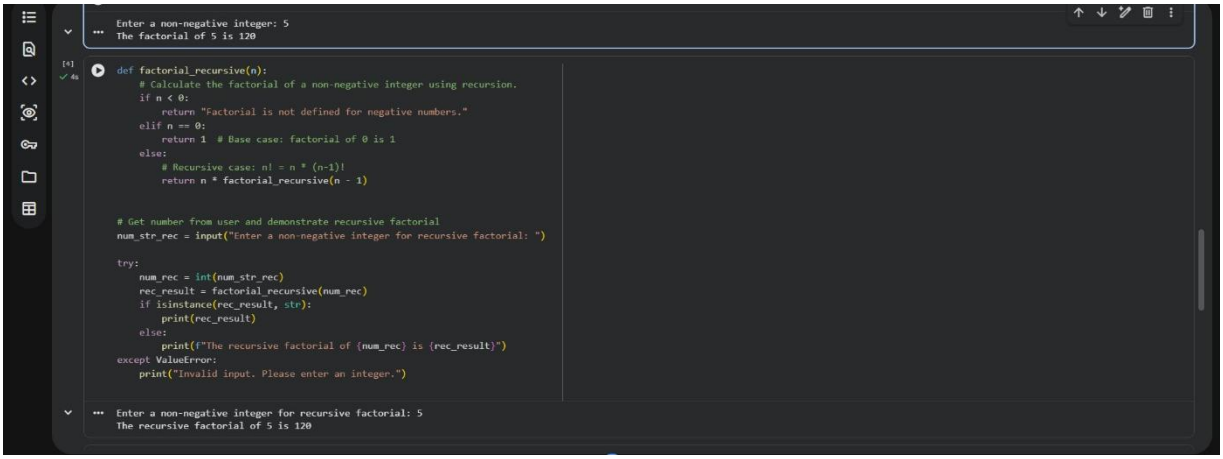
AI Dependency Risk:

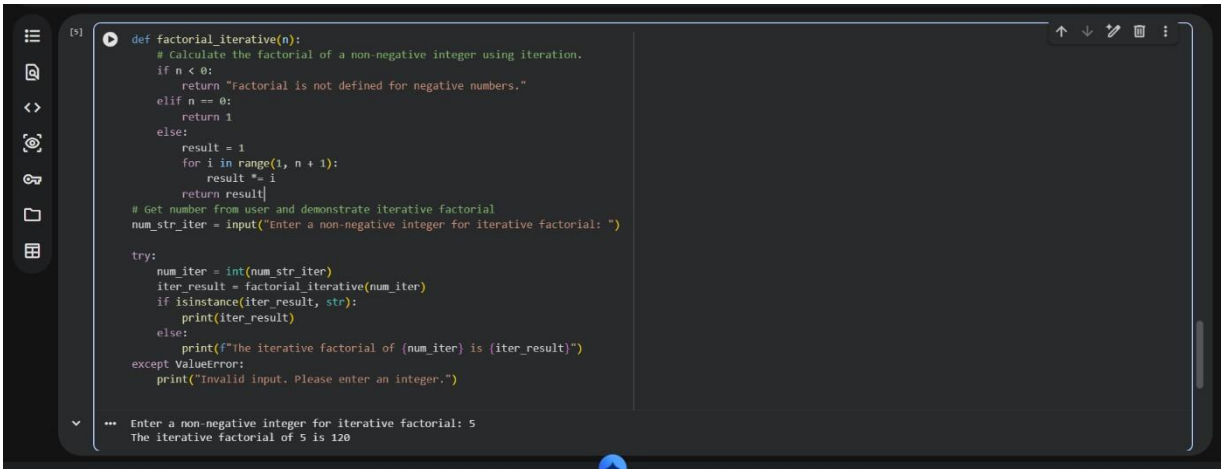
When using AI tools like Google Colab, procedural code may be generated quickly but often lacks proper structure. Modular code encourages better design practices, even when AI is used. This reduces the risk of poor-quality code.

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:





Explanation

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