

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 : CH.RAMCHARAN

Enrollment No. : 2303A52095

Batch No. : 33

TASK _01

The screenshot shows the Microsoft Visual Studio Code interface. The left sidebar has icons for Explorer, Search, Problems, Timeline, and Outline. The main area shows a file named 'lab1.py' with the following code:

```
1  # calculation without using functions
2  # loops and variables only
3
4  print("Enter a number to find factorial")
5
6  n = int(input())
7
8  factorial = 1
9
10 for i in range(1, n+1):
11     factorial *= i
12
13 print(f"Factorial of {n} is {factorial}")
```

The terminal below shows the output of running the script:

```
PS C:\Users\chirrr\Ram\ai lab> & "C:/Program Files/Python312/python.exe" "c:/Users/chirrr/Ram/ai lab/lab1.py"
Enter a number to find factorial: 5
Factorial of 5 is 120
PS C:\Users\chirrr\Ram\ai lab>
```

A Chat window on the right says: "write a python program to calculate factorial without using functions . use loops and variables only." Below it, a note says: "Done! I've written a Python program that calculates factorial using only loops and variables. The program:" followed by a bulleted list:

- Takes a number as input
- Uses a `while` loop to multiply all numbers from 1 to n
- Stores the result in the `factorial` variable
- Prints the result

At the bottom, there's a status bar with weather information (17°C, Partly sunny), system icons, and the date/time (10-01-2026).

This screenshot is nearly identical to the first one, showing the same code in 'lab1.py' and its execution output in the terminal. The code is:

```
1  # Factorial calculation without using functions
2  # Using loops and variables only
3
4  n = int(input("Enter a number to find factorial: "))
5
6  factorial = 1
7  i = 1
8
9  while i <= n:
10     factorial *= i
11     i += 1
12
13  print(f"Factorial of {n} is {factorial}")
```

The terminal output is:

```
PS C:\Users\chirrr\Ram\ai lab> & "C:/Program Files/Python312/python.exe" "c:/Users/chirrr/Ram/ai lab/lab1.py"
Enter a number to find factorial: 5
Factorial of 5 is 120
PS C:\Users\chirrr\Ram\ai lab>
```

The status bar at the bottom shows the date/time (10:00 AM, 10-01-2026).

EXPLANATION:

GitHub Copilot helped me quickly generate a working factorial program.

It followed basic Python syntax and logic correctly.

For beginners, it saves time and reduces syntax errors.

However, understanding the logic is still important.

Copilot gives suggestions but human review is needed

TASK_02:

The screenshot shows a Microsoft Visual Studio Code (VS Code) interface. The left sidebar has icons for Explorer, Search, Problems, and others. The main area shows an 'AI LAB' workspace with a file named 'lab1.py'. The code in the editor is:

```
def factorial(n):
    if n < 0:
        raise ValueError("Factorial is not defined for negative numbers")
    result = 1
    for i in range(1, n + 1):
        result *= i
    return result

try:
    num = int(input("Enter a number to find factorial: "))
    print(f"Factorial of {num} is {factorial(num)}")
except ValueError as e:
    print(e)
```

The status bar at the bottom indicates the code is running in a terminal window under 'ai lab' with Python 3.12.0. A tooltip from GitHub Copilot suggests optimizing the code for readability and efficiency. The tooltip content includes:

- ✓ Optimized factorial code for readability and efficiency
- ✓ Edited lab1.py +20 -10

Done! I've optimized the code with these improvements:

Efficiency:

- Replaced `while` loop with `for` loop (more efficient in Python)
- Uses `range()` for cleaner iteration
- Uses `*` operator instead of repeated multiplication

Readability:

- Added input validation to catch non-integer inputs
- Added error handling with try-except
- Added check for negative numbers
- Improved comments explaining each section
- Used f-string formatting for better output readability

```
# Factorial calculation without using functions
# Using loops and variables only

# Input validation
try:
    n = int(input("Enter a non-negative number to find factorial: "))

    # Validate input
    if n < 0:
        print("Factorial is not defined for negative numbers!")
    else:
        # Initialize result
        factorial = 1

        # Calculate factorial using for loop
        for i in range(1, n + 1):
            factorial *= i

        # Display result
        print(f"Factorial of {n} is {factorial}")

except ValueError:
    print("Invalid input! Please enter a valid integer.")
```

```
# Input validation
try:
    n = int(input("Enter a non-negative number to find factorial: "))

    # Validate input
    if n < 0:
        print("Factorial is not defined for negative numbers!")
    else:
        # Initialize result
        factorial = 1

        # Calculate factorial using for loop
        for i in range(1, n + 1):
            factorial *= i

        # Display result
        print(f"Factorial of {n} is {factorial}")

except ValueError:
    print("Invalid input! Please enter a valid integer.")
```

PS C:\Users\chirr\Ram\ai lab> & "C:/Program Files/Python312/python.exe" "c:/Users/chirr/Ram/ai lab/lab1.py"
Enter a non-negative number to find factorial: 5
Factorial of 5 is 120
PS C:\Users\chirr\Ram\ai lab>

Explanation

- Better variable names
- Cleaner output
- More readable

TASK_03:

The screenshot shows the Microsoft Visual Studio Code interface. The Explorer sidebar on the left shows a folder named 'AI LAB' containing a file 'lab1.py'. The main editor window displays the following Python code:

```
24
25
26 # Main program
27 try:
28     n = int(input("Enter a non-negative number to find factorial: "))
29     result = calculate_factorial(n)
30     print(f"Factorial of {n} is {result}")
31
32 except ValueError as e:
33     print(f"Error: {e}")
34 except Exception as e:
35     print(f"Invalid input! Please enter a valid integer.")
```

The terminal below the editor shows the command PS C:\Users\chirr\Ram\ai lab & "C:/Program Files/Python312/python.exe" "c:/Users/chirr/Ram/ai lab/lab1.py" and the output Enter a non-negative number to find factorial: 5 Factorial of 5 is 120.

A Chat window on the right has a message from AI: "write a python program to calculate factorial using a function". Below it, a summary says: "Done! I've rewritten the program to use a function. The improvements include:" followed by a bulleted list: "• Created `calculate_factorial()` function with proper documentation", "• Handles edge cases (0 and 1 return 1 immediately)", and "• Raises `ValueError` for negative numbers".

This screenshot shows the same Microsoft Visual Studio Code interface, but the code in the editor has been rewritten using a function-based approach:

```
1 # Factorial calculation using a function
2
3 def calculate_factorial(num):
4     """
5         Calculate factorial of a number using a loop.
6
7     Args:
8         num: An integer for which factorial is to be calculated
9
10    Returns:
11        The factorial of the input number
12    """
13    if num < 0:
14        raise ValueError("Factorial is not defined for negative numbers!")
15
16    if num == 0 or num == 1:
17        return 1
18
19    factorial = 1
20    for i in range(2, num + 1):
21        factorial *= i
22
23    return factorial
24
25
26 # Main program
27 try:
28     n = int(input("Enter a non-negative number to find factorial: "))
29     result = calculate_factorial(n)
30     print(f"Factorial of {n} is {result}")
31
32 except ValueError as e:
33     print(f"Error: {e}")
34 except Exception as e:
35     print(f"Invalid input! Please enter a valid integer.")
```

The terminal output remains the same: PS C:\Users\chirr\Ram\ai lab & "C:/Program Files/Python312/python.exe" "c:/Users/chirr/Ram/ai lab/lab1.py" and Enter a non-negative number to find factorial: 5 Factorial of 5 is 120.

The screenshot shows a Windows desktop environment with the Visual Studio Code (VS Code) application open. The code editor displays a Python file named 'lab1.py'. The code defines a function 'calculate_factorial' that calculates the factorial of a number using a loop. The terminal below the editor shows the execution of the script and its output, which is the factorial of 5 (120). The VS Code interface includes the Explorer sidebar on the left, the Command Palette at the top, and various status indicators at the bottom.

```
# Factorial calculation using a function
def calculate_factorial(num):
    """
    Calculate factorial of a number using a loop.

    Args:
        num: An integer for which factorial is to be calculated

    Returns:
        The factorial of the input number
    """
    if num < 0:
        return None
    factorial = 1
    for i in range(1, num + 1):
        factorial *= i
    return factorial

if __name__ == "__main__":
    num = int(input("Enter a non-negative number to find factorial: "))
    result = calculate_factorial(num)
    if result is not None:
        print(f"Factorial of {num} is {result}")
    else:
        print("Input must be a non-negative integer")
```

```
PS C:\Users\chirr\Ram\ai lab> & "C:/Program Files/Python312/python.exe" "c:/Users/chirr/Ram/ai lab/lab1.py"
Enter a non-negative number to find factorial: 5
Factorial of 5 is 120
PS C:\Users\chirr\Ram\ai lab>
```

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 GitHub Copilot, 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 VS Code interface with the following details:

- File Explorer:** Shows a folder named "AI LAB" containing "lab1.py".
- Editor:** Displays the content of "lab1.py".

```
54     print(f"Iterative approach")
55     print(f"Recursive approach")
56
57     except ValueError as e:
58         print(f"Error: {e}")
59     except RecursionError:
60         print("Error: Recursion depth exceeded for very large numbers!")
61     except Exception:
62         print("Invalid input! Please enter a valid integer.")
```
- Terminal:** Shows the output of running the script.

```
PS C:\Users\chiru\Ram\ai lab> & "C:/Program Files/Python312/python.exe" "c:/Users/chiru/Ram/ai lab/lab1.py"
Enter a non-negative number to find factorial: 5
Factorial of 5 is 120
PS C:\Users\chiru\Ram\ai lab>
```
- Chat:** A sidebar titled "CALCULATING FACTORIAL IN PYTHON WITHOUT FUNCTIONS" contains a message from an AI assistant.

```
write factorial using iteration and recursion
```

Created Python program for factorial calculations
Edited lab1.py +42 -15
Done! I've created a program with both approaches:

Iterative approach (factorial_iterative):

 - Uses a `for` loop to multiply numbers
 - More efficient with less memory usage
 - Better for large numbers

Recursive approach (factorial_recursive):

 - Calls itself with decreasing values
 - Uses base case (0 or 1 returns 1)
 - More elegant but uses more memory
 - Limited by recursion depth for large numbers

Main program:

 - Calculates factorial using both methods
 - Compares results
 - Handles errors including recursion depth limit
 - Shows the difference between the two approaches
- Bottom Status Bar:** Shows file path, line/col, spaces, encoding, and date/time.

The screenshot shows a VS Code interface with the following details:

- File Explorer:** Shows a folder named "AI LAB" containing "lab1.py".
- Editor:** Displays the content of "lab1.py".

```
1 # Factorial calculation using both iterative and recursive approaches
2
3 def factorial_iterative(num):
4     """
5         Calculate Factorial using Iteration (loop).
6
7     Args:
8         num: A non-negative integer
9
10    Returns:
11        The factorial of the input number
12    """
13    if num < 0:
14        raise ValueError("factorial is not defined for negative numbers!")
15
16    result = 1
17    for i in range(2, num + 1):
18        result *= i
19
20    return result
21
22
23 def factorial_recursive(num):
24     """
25         Calculate Factorial using recursion.
26
27     Args:
28         num: A non-negative integer
29
30    Returns:
31        The factorial of the input number
32    """
33    if num < 0:
34        raise ValueError("factorial is not defined for negative numbers!")
35
36    # Base case
37    if num == 0 or num == 1:
38        return 1
39
40    # Recursive case
41    return num * factorial_recursive(num - 1)
42
43 # Main program
44
45 try:
46     n = int(input("Enter a non-negative number to find factorial: "))
47
48     # Calculations using both methods
49     iterative_result = factorial_iterative(n)
50     recursive_result = factorial_recursive(n)
51
52     # Display results
53     print(f"Factorial of {n}:")
54     print(f"  Iterative approach: {iterative_result}")
55     print(f"  Recursive approach: {recursive_result}")
56
57     except ValueError as e:
58         print(f"Error: {e}")
59     except RecursionError:
60         print("Error: Recursion depth exceeded for very large numbers!")
61     except Exception:
62         print("Invalid input! Please enter a valid integer.")
```
- Bottom Status Bar:** Shows file path, line/col, spaces, encoding, and date/time.

The screenshot shows a Microsoft Visual Studio Code (VS Code) interface. The left sidebar has icons for Explorer, Search, Problems, and others. The main area shows a file named 'lab1.py' with the following code:

```
1 # Factorial calculation using both iterative and recursive approaches
2
3 def factorial_iterative(num):
4     """
5         Calculate factorial using iteration (loop).
6
7     Args:
8         num: A non-negative integer
9
10    Returns:
11        The factorial of the input number
12    """
13    if num < 0:
```

Below the code editor, there are tabs for PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL, and PORTS. The TERMINAL tab is active, showing command-line output:

```
PS C:\Users\chirr\Ram\ai lab> & "C:/Program Files/Python312/python.exe" "c:/Users/chirr/Ram/ai lab/lab1.py"
Enter a non-negative number to find factorial: 5

Factorial of 5:
Iterative approach: 120
Recursive approach: 120
PS C:\Users\chirr\Ram\ai lab>
```

The status bar at the bottom shows the file path 'C:\Users\chirr\Ram\ai lab\lab1.py', line 63, column 1, spaces: 4, encoding: UTF-8, Python 3.12.0, and the date/time '10-01-2026'. The taskbar at the bottom of the screen also displays various application icons.

Explanation

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