

# **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**

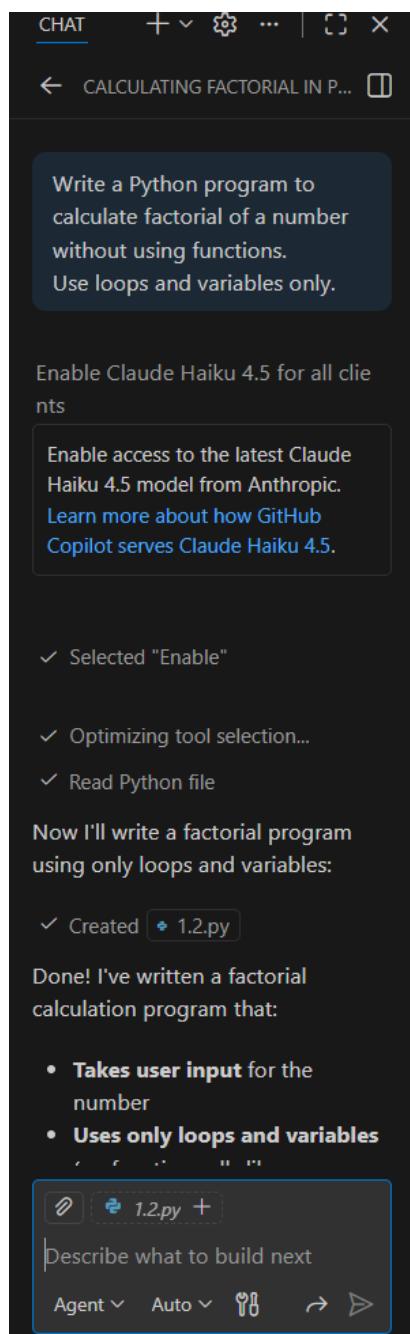
**Name of Student : P. Sricharan Goud**

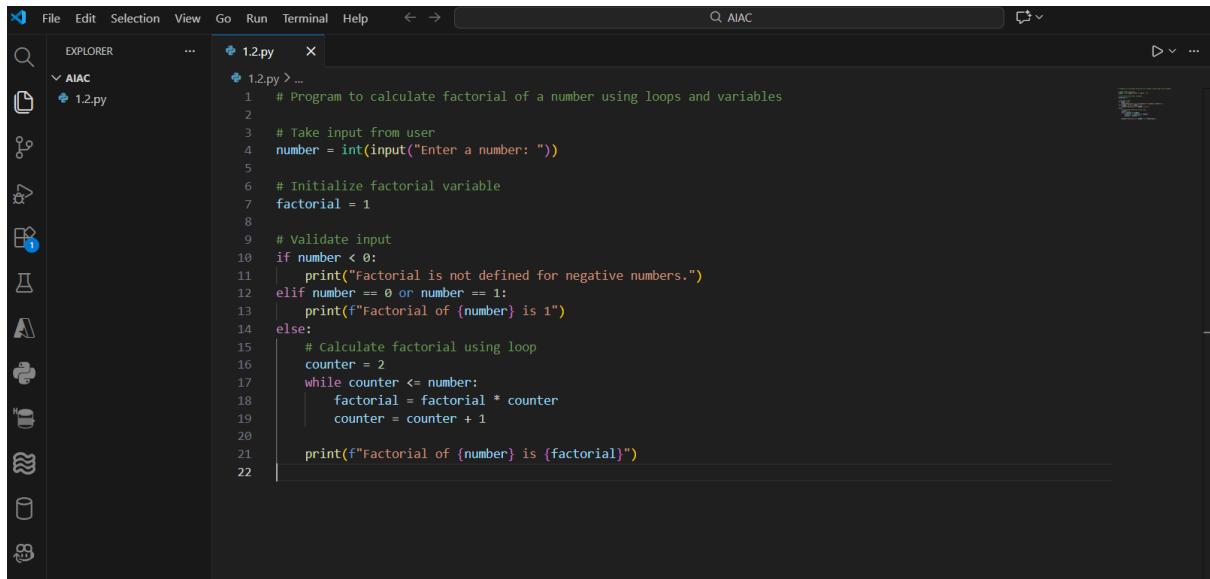
**Enrollment No. : 2303A52100**

**Batch No. : 33**

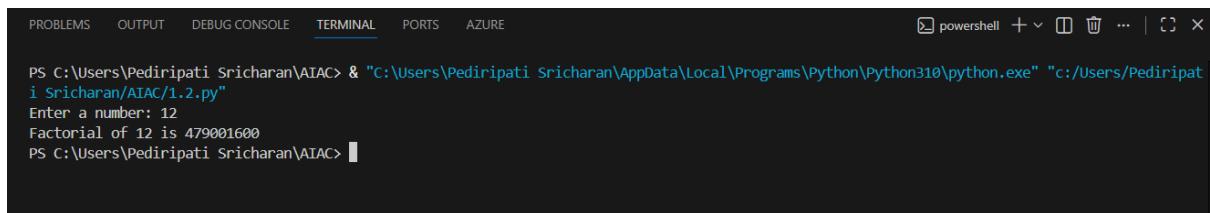
**Date :09/01/26**

**TASK \_01**





```
1 # Program to calculate factorial of a number using loops and variables
2
3 # Take input from user
4 number = int(input("Enter a number: "))
5
6 # Initialize factorial variable
7 factorial = 1
8
9 # Validate input
10 if number < 0:
11     print("Factorial is not defined for negative numbers.")
12 elif number == 0 or number == 1:
13     print(f"Factorial of {number} is 1")
14 else:
15     # Calculate factorial using loop
16     counter = 2
17     while counter <= number:
18         factorial = factorial * counter
19         counter = counter + 1
20
21     print(f"Factorial of {number} is {factorial}")
```



```
PS C:\Users\Pediripati Sricharan\AIAC & "C:\Users\Pediripati Sricharan\AppData\Local\Programs\Python\Python310\python.exe" "c:/Users/Pediripati Sricharan/AIAC/1.2.py"
Enter a number: 12
Factorial of 12 is 479001600
PS C:\Users\Pediripati Sricharan\AIAC>
```

## 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:

CHAT + ⚙️ ... | ☰ X

← CALCULATING FACTORIAL IN P...

Optimize and simplify this code to improve readability.

✓ Read Python file

The file appears to be empty. Let me recreate it with an optimized and simplified version:

✓ Created 1.2.py

I've optimized the code with these improvements:

- **Removed unnecessary condition** for 0 and 1 (the loop handles them correctly)
- **Used compound operators** (`*=` and `+=`) for cleaner code
- **Simplified logic** by merging edge cases
- **Reduced lines** from 20 to 13 while maintaining the same functionality
- **Kept readability** with clear variable names and structure

The program still uses only loops and variables, and produces the same results.

0 1.2.py +

Describe what to build next

Agent Auto 🕵️ ➔

This screenshot shows a user interface for AI-assisted code optimization. At the top, there's a header with 'CHAT' and other icons. Below it, a breadcrumb navigation shows the current task: 'CALCULATING FACTORIAL IN P...'. A prominent callout box in the center says 'Optimize and simplify this code to improve readability.' Below this, a list indicates that the Python file has been read. The main text area explains that the file is empty and provides a summary of the optimizations made to the code. It lists five bullet points detailing how the code was simplified: removing unnecessary conditions, using compound assignment operators, simplifying logic, reducing lines of code, and maintaining readability. At the bottom, there's a section for describing the next steps, with dropdown menus for 'Agent' and 'Auto' and some status indicators.

File Edit Selection View Go Run Terminal Help ← → 🔍 AIAC

EXPLORER ... 1.2.py X

1.2.py > ...

```
1 # Factorial calculation using loops
2
3 number = int(input("Enter a number: "))
4 factorial = 1
5
6 if number < 0:
7     print("Factorial is not defined for negative numbers.")
8 else:
9     counter = 2
10    while counter <= number:
11        factorial *= counter
12        counter += 1
13
14    print(f"Factorial of {number} is {factorial}")
15
```

This screenshot shows a code editor window with a dark theme. The menu bar includes 'File', 'Edit', 'Selection', 'View', 'Go', 'Run', 'Terminal', and 'Help'. On the left is an 'EXPLORER' sidebar with icons for files, folders, and other project components. The main pane displays a Python script named '1.2.py'. The code itself is a simple loop that calculates the factorial of a given number using compound assignment operators. The editor interface includes standard controls like back/forward arrows and a search bar at the top.

A screenshot of a terminal window titled "powershell". The command entered is "C:\Users\Pediripati Sricharan\AIAC> & "C:\Users\Pediripati Sricharan\AppData\Local\Programs\Python\Python310\python.exe" "c:/Users/Pediripati Sricharan/AIAC/1.2.py"" followed by "Enter a number: 12" and "Factorial of 12 is 479001600". The prompt "PS C:\Users\Pediripati Sricharan\AIAC>" is visible at the bottom.

## Explanation

- Better variable names
- Cleaner output
- More readable

## TASK\_03:

A screenshot of a task summary card. The title is "CALCULATING FACTORIAL IN P...". The task description is: "Write a Python program to calculate factorial using a function with proper comments.". A note says "✓ Read and created Python file". The main text states: "Done! I've written a factorial program using a function with comprehensive comments:". The "Key features:" section lists:

- `calculate_factorial()` function with proper parameter and return documentation
- **Docstring** explaining the function's purpose, arguments, and return value
- **Inline comments** describing each step of the logic
- **Error handling** for negative numbers
- **Clean main code** that calls the function and displays results
- **Reusable** - the function can be called multiple times

At the bottom, there are icons for copy, share, and report, and a note to "Describe what to build next".

1.2.py X

C:\Users\Pediripati Sricharan\AIAC\1.2.py

```
1 # Program to calculate factorial using a function with proper comments
2
3 def calculate_factorial(n):
4     """
5         Calculate the factorial of a number.
6
7     Args:
8         n (int): The number to calculate factorial for
9
10    Returns:
11        int: The factorial of n, or None if invalid input
12    """
13
14    # Check for invalid input (negative numbers)
15    if n < 0:
16        print("Error: Factorial is not defined for negative numbers.")
17        return None
18
19    # Base case: factorial of 0 or 1 is 1
20    if n == 0 or n == 1:
21        return 1
22
23    # Initialize result variable
24    result = 1
25
26    # Multiply all numbers from 2 to n
27    counter = 2
28    while counter <= n:
29        result *= counter
30        counter += 1
31
32
33
34    # Main program - Get input and display result
35    number = int(input("Enter a number: "))
36    factorial_result = calculate_factorial(number)
37    Q
38    if factorial_result is not None:
39        print(f"Factorial of {number} is {factorial_result}")
```

## **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 software interface with the following details:

- Top Bar:** CHAT, +, gear icon, ..., |, file icon, X.
- Title Bar:** ← CALCULATING FACTORIAL IN P... □
- Task Description:** Generate iterative and recursive factorial programs in Python.
- Status:** ✓ Read and created Python file
- Text:** Created a comprehensive factorial program with both approaches:
- Section:** **Iterative Method:**
  - Uses a `for` loop to multiply numbers sequentially
  - More efficient ( $O(1)$  space)
  - Better for large numbers
- Section:** **Recursive Method:**
  - Calls itself with decremented value
  - More elegant but uses call stack
  - Less efficient for large numbers ( $O(n)$  space)
- Section:** **Main Program:**
  - Demonstrates both methods side-by-side
  - Validates both produce the same result
  - Includes error handling for invalid input
- Bottom Bar:** ⌂ ↻ ⌛ ⌚
- File List:** Ø 1.2.py +
- Text Input:** Describe what to build next
- Bottom Buttons:** Agent ▾ Auto ▾ ⌂ ⌛ ⌚ ⌚ ⌚

```
1.2.py X
C:\Users\Pediripati Sriharan\AIAC\1.2.py
1 # Factorial Programs: Iterative and Recursive Approaches
2
3 # ===== ITERATIVE APPROACH =====
4 def factorial_iterative(n):
5     """
6         Calculate factorial using iteration (loop).
7         Time Complexity: O(n)
8         Space Complexity: O(1)
9     """
10    # Handle invalid input
11    if n < 0:
12        return None
13
14    # Initialize result
15    result = 1
16
17    # Multiply numbers from 2 to n
18    for i in range(2, n + 1):
19        result *= i
20
21    return result
22
23
24 # ===== RECURSIVE APPROACH =====
25 def factorial_recursive(n):
26     """
27         Calculate factorial using recursion.
28         Time Complexity: O(n)
29         Space Complexity: O(n) - due to call stack
30     """
31    # Handle invalid input
32    if n < 0:
33        return None
34
35    # Base case: factorial of 0 or 1 is 1
36    if n == 0 or n == 1:
37        return 1
```

```
38    # Recursive case: n! = n * (n-1)!
39    return n * factorial_recursive(n - 1)
40
41
42
43 # ===== MAIN PROGRAM =====
44 def main():
45     """Main program to demonstrate both approaches."""
46
47     try:
48         number = int(input("Enter a number: "))
49
50         # Calculate using iterative approach
51         iter_result = factorial_iterative(number)
52
53         # Calculate using recursive approach
54         recur_result = factorial_recursive(number)
55
56         # Display results
57         if iter_result is None:
58             print("Error: Factorial is not defined for negative numbers.")
59         else:
60             print(f"\nFactorial of {number}:")
61             print(f" Iterative Result: {iter_result}")
62             print(f" Recursive Result: {recur_result}")
63             print(f" Both methods match: {iter_result == recur_result}")
64
65     except ValueError:
66         print("Error: Please enter a valid integer.")
67
68
69 # Run the program
70 if __name__ == "__main__":
71     main()
```

# Explanation

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