

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 : k.sathwika

Enrollment No. : 2303A52096

Batch No. : 33

Date :09/01/26

TASK _01

CHAT + ⚙️ ⋮ | [] X
← CALCULATING FACTORIAL IN PYTHON WITHOUT FUNCTIONS

Write a Python program to calculate factorial without using functions.

Use loops and variables only.

Enable Claude Haiku 4.5 for all clients

Enable access to the latest Claude Haiku 4.5 model from Anthropic. [Learn more about how GitHub Copilot serves Claude Haiku 4.5.](#)

✓ Selected "Enable"

✓ Created factorial.py file

I've created a Python program that calculates factorial without using functions. The program:

- Takes user input for a number
- Handles edge cases (negative numbers, 0, and 1)
- Uses a `while` loop and variables to multiply numbers from 2 to the input number

factorial.py

Describe what to build next

Agent ▾ Auto ▾ 🛠

4 UTF-8 CRLF { } Python 3.11.9 (Microsoft Store) (●) Go Live 🔔

A screenshot of the Visual Studio Code interface. The top bar shows the menu: File, Edit, Selection, View, Go, Run, ... A search bar contains the text "AI coding". The left sidebar has sections for EXPLORER, OPEN EDITORS, and AI CODING. Under AI CODING, there is a folder "Assignment 1.2" containing a file "factorial.py". The main editor area shows the following Python code:

```
Assignment 1.2 > factorial.py > ...
1 # Calculate factorial without using functions
2 # Using only loops and variables
3
4 number = int(input("Enter a number to calculate factorial: "))
5
6 # Check if number is negative
7 if number < 0:
8     print("Factorial is not defined for negative numbers.")
9 elif number == 0 or number == 1:
10    print(f"Factorial of {number} is 1")
11 else:
12    # Calculate factorial using a loop
13    factorial = 1
14    counter = 2
15
16    while counter <= number:
17        factorial = factorial * counter
18        counter = counter + 1
19
20    print(f"Factorial of {number} is {factorial}")
21
```

The terminal tab is active in the bottom navigation bar. The terminal window displays the following output:

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS +  
PS C:\AI coding> & C:/Users/sathvika/AppData/Local/Microsoft/WindowsApps/python  
PS C:\AI coding> & C:/Users/sathvika/AppData/Local/Microsoft/WindowsApps/python  
3.11.exe "c:/AI coding/Assignment 1.2/factorial.py"  
Enter a number to calculate factorial: 5  
● Factorial of 5 is 120
```

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:

Optimize this code to improve readability and efficiency

- ✓ Read factorial.py file

I'll optimize the code for better readability and efficiency:

- ✓ Edited `+ factorial.py +5 -8`

Now let me test the optimized code:

```
∅ ⚡ cd "c:\AI coding\Assignment 1.2"; echo  
"5" | python factorial.py
```

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

Readability:

- Removed redundant `elif` condition for 0 and 1 (the loop naturally handles these cases)
- Used `for` loop with `range()` instead of `while` loop (more Pythonic)
- Simplified the control flow from 3 branches to 2

  `+ factorial.py`

Describe what to build next

Agent ▾ Auto ▾ 

 

A screenshot of a code editor window titled "factorial.py". The code is a Python script that calculates the factorial of a given number. It includes validation for negative numbers and uses a for-range loop for calculation. The code is color-coded for syntax.

```
1 # Calculate factorial without using functions
2 # Using only loops and variables
3
4 number = int(input("Enter a number to calculate factorial: "))
5
6 # Validate input
7 if number < 0:
8     print("Factorial is not defined for negative numbers.")
9 else:
10    # Initialize result
11    factorial = 1
12
13    # Calculate factorial using a for-range loop (more efficient than a while loop)
14    for i in range(2, number + 1):
15        factorial *= i
16
17    print(f"Factorial of {number} is {factorial}")
18
```

A screenshot of the VS Code terminal tab. The terminal shows the command "PS C:\AI coding> & C:/Users/sathvika/AppData/Local/Microsoft/WindowsApps/python3.11.exe "c:/AI coding/Assignment 1.2/factorial.py"" being run. The output shows the user entering "5" and the program outputting "Factorial of 5 is 120".

```
● PS C:\AI coding> & C:/Users/sathvika/AppData/Local/Microsoft/WindowsApps/python3.11.exe "c:/AI coding/Assignment 1.2/factorial.py"
Enter a number to calculate factorial: 5
Factorial of 5 is 120
```

Explanation

- Better variable names
- Cleaner output
- More readable

TASK_03:

← CALCULATING FACTORIAL IN PYTHON WITHOUT FU... ☒

Restore Checkpoint

Write a Python program to calculate factorial using a function

✓ Created factorial_with_function.py file

- cd "c:\AI coding\Assignment 1.2";
- ⊖ echo "5" | python factorial_with_function.py

Created `+ factorial_with_function.py` with a function-based approach:

Features:

- `calculate_factorial()` function with docstring
- Input validation (returns `None` for negative numbers)
- Uses a `for` loop with `range()` for efficiency
- Clean separation of function logic and main program
- Easy to reuse and test

This complements the original loop-based version without functions.

`factorial_with_function.py`

Describe what to build next

Agent ▾ Auto ▾ 🎨

⟳ ➡

```
factorial.py          factorial_with_function.py X
Assignment 1.2 > factorial_with_function.py > ...
1  # Calculate factorial using a function
2
3  def calculate_factorial(n):
4      """
5          Calculate the factorial of a number using a loop.
6
7          Args:
8              n: A non-negative integer
9
10         Returns:
11             The factorial of n
12             """
13
14         if n < 0:
15             return None
16
17         result = 1
18         for i in range(2, n + 1):
19             result *= i
20
21         return result
22
23
24     # Main program
25     number = int(input("Enter a number to calculate factorial: "))
26
27     factorial_result = calculate_factorial(number)
28
29     if factorial_result is None:
30         print("Factorial is not defined for negative numbers.")
31     else:
32         print(f"Factorial of {number} is {factorial_result}")
```

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS
+  ...  []
powershell...
Python
/factorial_with_function.py ...
● PS C:\AI coding> & C:/Users/sathvika/AppData/Local/Microsoft/WindowsApps/python
3.11.exe "c:/AI coding/Assignment 1.2/factorial_with_function.py"
Enter a number to calculate factorial: 5
Factorial of 5 is 120
○ PS C:\AI coding>
```

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:

← CALCULATING FACTORIAL IN PYTHON WITHOUT FUNCTIONS

Restore Checkpoint

Write factorial using iteration and recursion

- ✓ Read factorial_with_function.py
- ✓ Edited `+ factorial_with_function.py +30 -8`

Run `pwsh` command?

```
cd "c:\AI coding\Assignment 1.2"; echo "5" | python factorial_with_function.py
```

Allow | Skip

`factorial_with_function.py`

Describe what to build next

Agent ▾ Auto ▾

↶ ➤

factorial.py factorial_with_function.py

```
Assignment 1.2 > factorial_with_function.py > ...
1 # Calculate factorial using iteration and recursion
2
3 # ===== ITERATIVE APPROACH =====
4 def factorial_iterative(n):
5     """
6         Calculate factorial using iteration (loop).
7
8     Args:
9         n: A non-negative integer
10
11    Returns:
12        The factorial of n, or None if negative
13    """
14    if n < 0:
15        return None
16
17    result = 1
18    for i in range(2, n + 1):
19        result *= i
20
21    return result
```

factorial.py factorial_with_function.py

```
Assignment 1.2 > factorial_with_function.py > ...
24 # ===== RECURSIVE APPROACH =====
25 def factorial_recursive(n):
26     """
27         Calculate factorial using recursion.
28
29     Args:
30         n: A non-negative integer
31
32    Returns:
33        The factorial of n, or None if negative
34    """
35    if n < 0:
36        return None
37    elif n == 0 or n == 1:
38        return 1
39    else:
40        return n * factorial_recursive(n - 1)
41
42
43 # Main program
44 number = int(input("Enter a number to calculate factorial: "))
45
46 if number < 0:
47     print("Factorial is not defined for negative numbers.")
48 else:
49     iterative_result = factorial_iterative(number)
50     recursive_result = factorial_recursive(number)
51
52     print(f"Iterative: Factorial of {number} is {iterative_result}")
53     print(f"Recursive: Factorial of {number} is {recursive_result}")
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
hon3.11.exe "c:/AI coding/Assignment 1.2/factorial_with_function.py"
Enter a number to calculate factorial: 5
Iterative: Factorial of 5 is 120
Recursive: Factorial of 5 is 120
PS C:\AI coding> []
```

Explanation

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