

# **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**

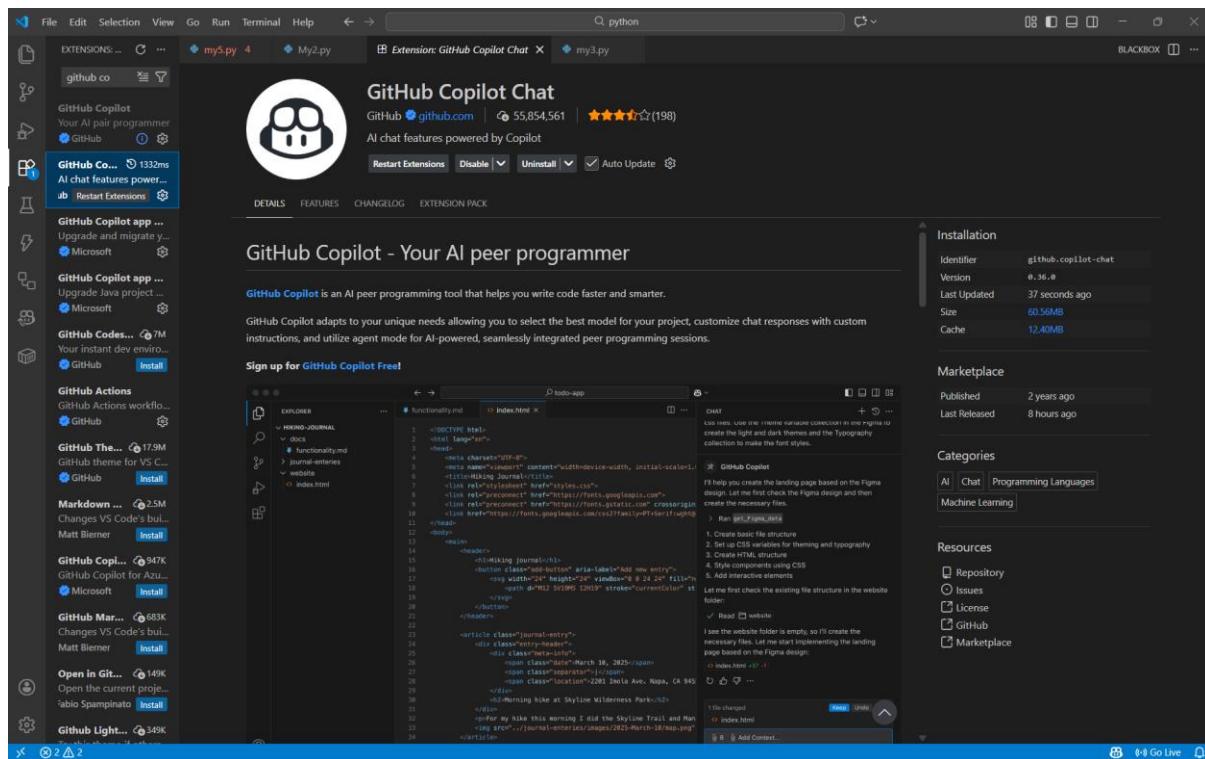
**Name of Student : R.Deekshith**

**Enrollment No. : 2303A52104**

**Batch No. : 33**

**Date : 09/01/26**

# Git Hub



# TASK \_01

CHAT + ↻ ⚙️ ⋮ | [ ] X

← CALCULATING FACTORIAL IN PYTHON WITHOUT FUNCTIONS

Write a Python program to calculate factorial of a number

Do not use functions

Use only loops and variables

my6.py

> Used 1 reference  
✓ Created factorial code snippet and formatted code

```
# ...existing code...
n_str = input("Enter a non-negative integer: ")
try:
    n = int(n_str)
except:
    print("Invalid input; please enter an integer.")
else:
    if n < 0:
        print("Factorial is not defined for negative numbers.")
    else:
        result = 1
        i = 1
        while i <= n:
            result *= i
            i += 1
        print(f"Factorial of {n} is {result}")
# ...existing code...
```

C ↺ ⌂ ⌂

my6.py

Explore and understand your code

Ask Auto ➡

Ln 17, Col 22 Spaces: 4 UTF-8 CRLF { } Python 3.14.0 Go Live

A screenshot of a code editor window titled "my6.py". The code is a Python script that prompts the user for a non-negative integer, calculates its factorial using a while loop, and prints the result. The code includes comments indicating existing code at the beginning and end.

```
1 # ...existing code...
2 n_str = input("Enter a non-negative integer: ")
3 try:
4     n = int(n_str)
5 except:
6     print("Invalid input; please enter an integer.")
7 else:
8     if n < 0:
9         print("Factorial is not defined for negative numbers.")
10    else:
11        result = 1
12        i = 1
13        while i <= n:
14            result *= i
15            i += 1
16        print(f"Factorial of {n} is {result}")
17 # ...existing code...
```

A screenshot of a terminal window titled "Python". It shows the execution of the "my6.py" script. The user enters "5" as input, and the program outputs "Factorial of 5 is 120".

```
PS C:\drive D\python> & C:/Python314/python.exe "c:/drive D/python/my6.py"
Enter a non-negative integer: 5
Factorial of 5 is 120
PS C:\drive D\python>
```

## 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 code editor interface with a dark theme. At the top, there's a header bar with a 'CHAT' tab, a file navigation section ('← CALCULATING FACTORIAL IN PYTHON WITHOUT FUNCTIONS'), and a close button ('X'). Below the header is a toolbar with icons for 'Restore Checkpoint' and other settings.

A central callout box contains the text 'Optimize this code' and 'Simplify logic and improve readability'. Below this, there's a 'Click to Edit' button and a file icon labeled 'my6.py'.

On the left side of the editor, there's a list of code analysis results:

- > Used 1 reference
- ✓ Optimized factorial calculation and provided optimization

Below the analysis results, the text 'Optimized version:' is followed by the Python code:

```
# ...existing code...
s = input("Enter a non-negative integer: ").strip()
try:
    n = int(s)
except ValueError:
    print("Invalid input; please enter an integer.")
else:
    if n < 0:
        print("Factorial is not defined for negative numbers.")
    else:
        result = 1
        for i in range(2, n + 1):
            result *= i
        print(f"Factorial of {n} is {result}")
# ...existing code...
```

At the bottom of the editor, there are several small icons: a refresh circle, a double arrow, a square, and a question mark. The status bar at the bottom shows the file name 'my6.py' and some navigation controls: 'Ask' and 'Auto' dropdowns, and arrows for navigating between files.

The screenshot shows a code editor window with a dark theme. The file tab at the top says "my6.py". The code itself is:

```
1 s = input("Enter a non-negative integer: ").strip()
2 try:
3     n = int(s)
4 except ValueError:
5     print("Invalid input; please enter an integer.")
6 else:
7     if n < 0:
8         print("Factorial is not defined for negative numbers.")
9     else:
10        result = 1
11        for i in range(2, n + 1):
12            result *= i
13        print(f"Factorial of {n} is {result}")
```

The screenshot shows a terminal window with a dark theme. The title bar says "TERMINAL". The terminal output is:

```
PS C:\drive D\python> & C:/Python314/python.exe "c:/drive D/python/my6.py"
Enter a non-negative integer: 5
Factorial of 5 is 120
PS C:\drive D\python> & C:/Python314/python.exe "c:/drive D/python/my6.py"
● Enter a non-negative integer: 7
Factorial of 7 is 5040
○ PS C:\drive D\python>
```

## Explanation

- Better variable names
- Cleaner output
- More readable

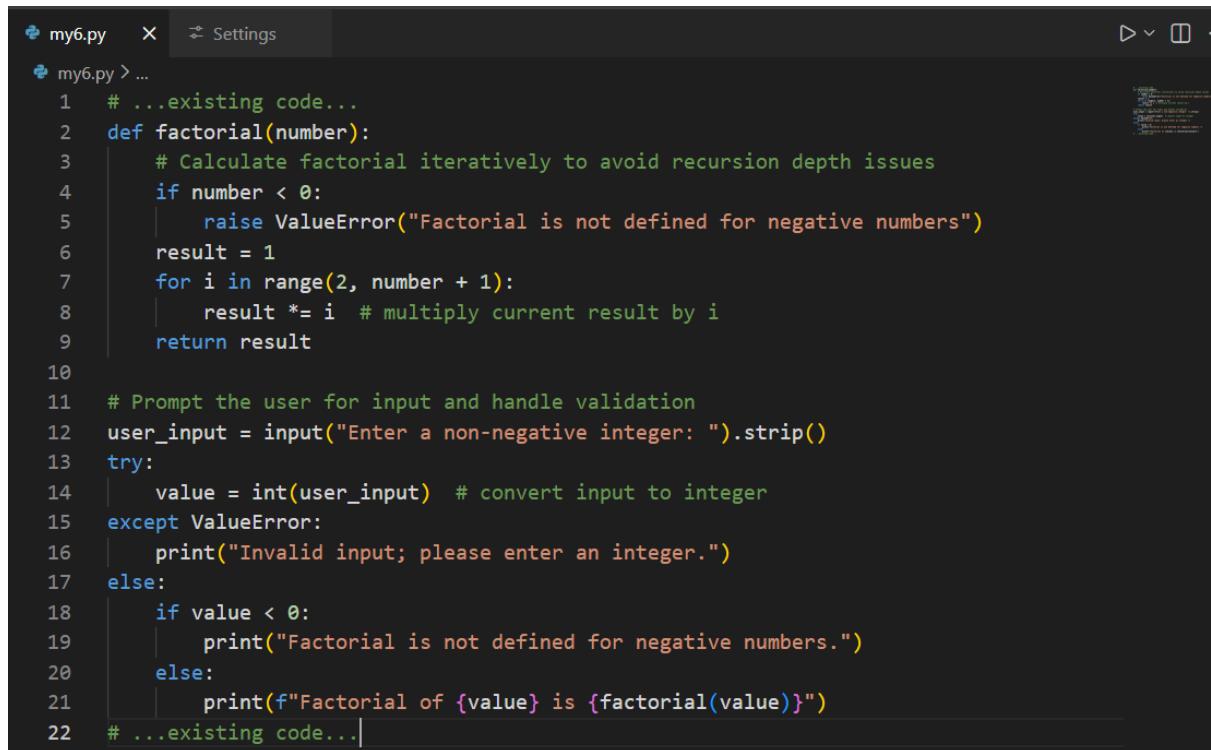
## TASK\_03:

The screenshot shows a code editor interface with the following details:

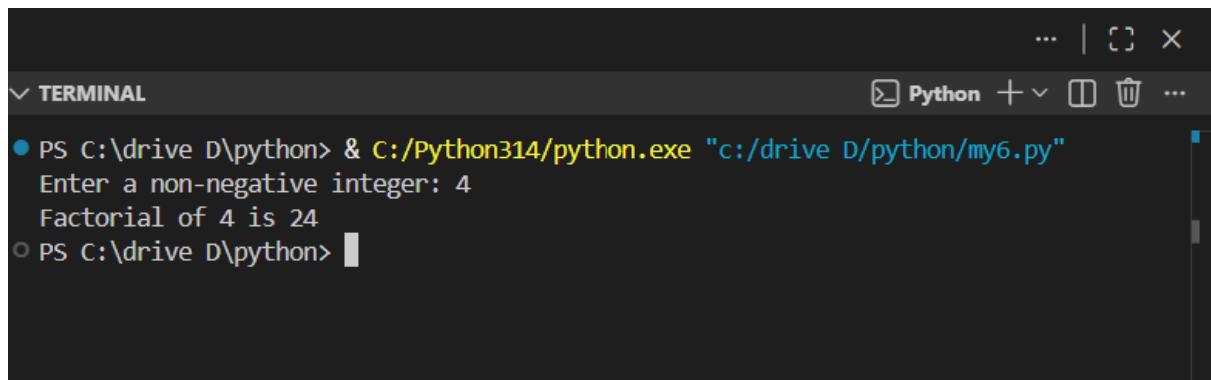
- Header:** CHAT, CALCULATING FACTORIAL IN PYTHON WITHOUT FUNCTIONS
- Task Instructions:**
  - Write a Python program to calculate factorial using a function
  - Use meaningful variable names
  - Add inline comments
- File:** my6.py
- Code Content:**

```
# ...existing code...
def factorial(number):
    # Calculate factorial iteratively to avoid recursion depth issues
    if number < 0:
        raise ValueError("Factorial is not defined for negative numbers")
    result = 1
    for i in range(2, number + 1):
        result *= i  # multiply current result by i
    return result

# Prompt the user for input and handle validation
user_input = input("Enter a non-negative integer: ").strip()
try:
    value = int(user_input)  # convert input to integer
except ValueError:
    print("Invalid input; please enter an integer.")
else:
    if value < 0:
        print("Factorial is not defined for negative numbers.")
    else:
        print(f"Factorial of {value} is {factorial(value)}")
# ...existing code...
```
- Bottom Bar:** my6.py, Explore and understand your code, Ask ▾ Auto ▾, and two right-pointing arrows.



```
my6.py > ...
1 # ...existing code...
2 def factorial(number):
3     # Calculate factorial iteratively to avoid recursion depth issues
4     if number < 0:
5         raise ValueError("Factorial is not defined for negative numbers")
6     result = 1
7     for i in range(2, number + 1):
8         result *= i # multiply current result by i
9     return result
10
11 # Prompt the user for input and handle validation
12 user_input = input("Enter a non-negative integer: ").strip()
13 try:
14     value = int(user_input) # convert input to integer
15 except ValueError:
16     print("Invalid input; please enter an integer.")
17 else:
18     if value < 0:
19         print("Factorial is not defined for negative numbers.")
20     else:
21         print(f"Factorial of {value} is {factorial(value)}")
22 # ...existing code...
```



```
... | [] X
TERMINAL Python + - X ...
● PS C:\drive D\python> & C:/Python314/python.exe "c:/drive D/python/my6.py"
Enter a non-negative integer: 4
Factorial of 4 is 24
○ PS C:\drive D\python>
```

## 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:

← ITERATIVE FACTORIAL PROGRAM IN PYTHON

### Generate iterative factorial program in Python

my6.py

> Used 1 reference

```
def factorial(n):
    """
    Calculate factorial of n using iterative approach.

    Args:
        n: Non-negative integer

    Returns:
        Factorial of n
    """
    if n < 0:
        raise ValueError("Factorial not defined for negative numbers")

    result = 1
    for i in range(2, n + 1):
        result *= i

    return result

# Test cases
if __name__ == "__main__":
    print(f"5! = {factorial(5)}")
    print(f"0! = {factorial(0)}")
    print(f"10! = {factorial(10)}")
```

This program calculates factorials iteratively by multiplying numbers from 2 to n. It includes error handling for negative numbers and test cases to verify functionality.

my6.py

Explore and understand your code

Ask ▾ Auto ▾

Ln 1, Col 1 Spaces: 4 UTF-8 CRLF { } Python 3.14.0 Go Live

```
File Edit Selection View Go Run Terminal Help ⏪ ⏫ ⏪ ⏫ ... 🔍 python
my6.py × Settings
my6.py > ...
1 def factorial(n):
2     """
3         Calculate factorial of n using iterative approach.
4
5     Args:
6         n: Non-negative integer
7
8     Returns:
9         Factorial of n
10    """
11    if n < 0:
12        raise ValueError("Factorial not defined for negative numbers")
13
14    result = 1
15    for i in range(2, n + 1):
16        result *= i
17
18    return result
19
20
21 # Test cases
22 if __name__ == "__main__":
23     print("5! = " + str(factorial(5)))
24     print("0! = " + str(factorial(0)))
25     print("10! = " + str(factorial(10)))
```

```
... | []
TERMINAL
PS C:\drive D\python> & C:/Python314/python.exe "c:/drive D/python/my6.py"
5! = 120
0! = 1
10! = 3628800
PS C:\drive D\python> []
```

## Explanation

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

