

# **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

The screenshot shows the GitHub Copilot Chat extension interface within a Visual Studio Code window. The interface is divided into several sections:

- Left Sidebar (Extensions):** Lists installed extensions, including GitHub Copilot, GitHub Copilot app, GitHub Codespaces, GitHub Actions, GitHub Themes, GitHub Copilot for VS Code, GitHub Copilot for Azure, and GitHub Light.
- Main Panel:**
  - GitHub Copilot Chat Header:** Displays the GitHub Copilot logo, the name "GitHub Copilot Chat", the GitHub profile link, a star count of 55,854,561, and a rating of 4.5 stars (198 reviews). It includes buttons for "Restart Extensions", "Disable", "Uninstall", and "Auto Update".
  - Details Section:** Describes GitHub Copilot as "Your AI peer programmer" and "an AI peer programming tool that helps you write code faster and smarter." It mentions that it adapts to unique needs, allows selecting the best model, and utilizes agent mode for AI-powered sessions.
  - Sign up for GitHub Copilot Free:** A link to sign up for the free version.
  - Code Editor:** Shows a snippet of HTML code for a "journal.html" file, including a header, a list of entries, and a footer.
  - Chat Panel:** Displays a chat conversation with GitHub Copilot. The user's prompt is: "I'll help you create the landing page based on the Figma design. Let me first check the Figma design and then create the necessary files." The assistant's response is: "I'll help you create the landing page based on the Figma design. Let me first check the Figma design and then create the necessary files." The chat panel also shows a list of suggested actions: "1. Create basic file structure", "2. Set up CSS variables for theming and typography", "3. Create HTML structure", "4. Style components using CSS", "5. Add interactive elements".
- Right Sidebar (Marketplace):**
  - Installation:** Shows the identifier "github.copilot-chat", version "0.36.0", last updated "37 seconds ago", size "40.50MB", and cache "12.40MB".
  - Marketplace:** Shows the extension was published "2 years ago" and last released "8 hours ago".
  - Categories:** Lists "AI", "Chat", "Programming Languages", and "Machine Learning".
  - Resources:** Includes links to "Repository", "Issues", "License", "GitHub", and "Marketplace".

# TASK\_01

CHAT

+ v ⚙️ ... | [ ] ×

← 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...
```

↻ ↺ 👍 🗨

⌵

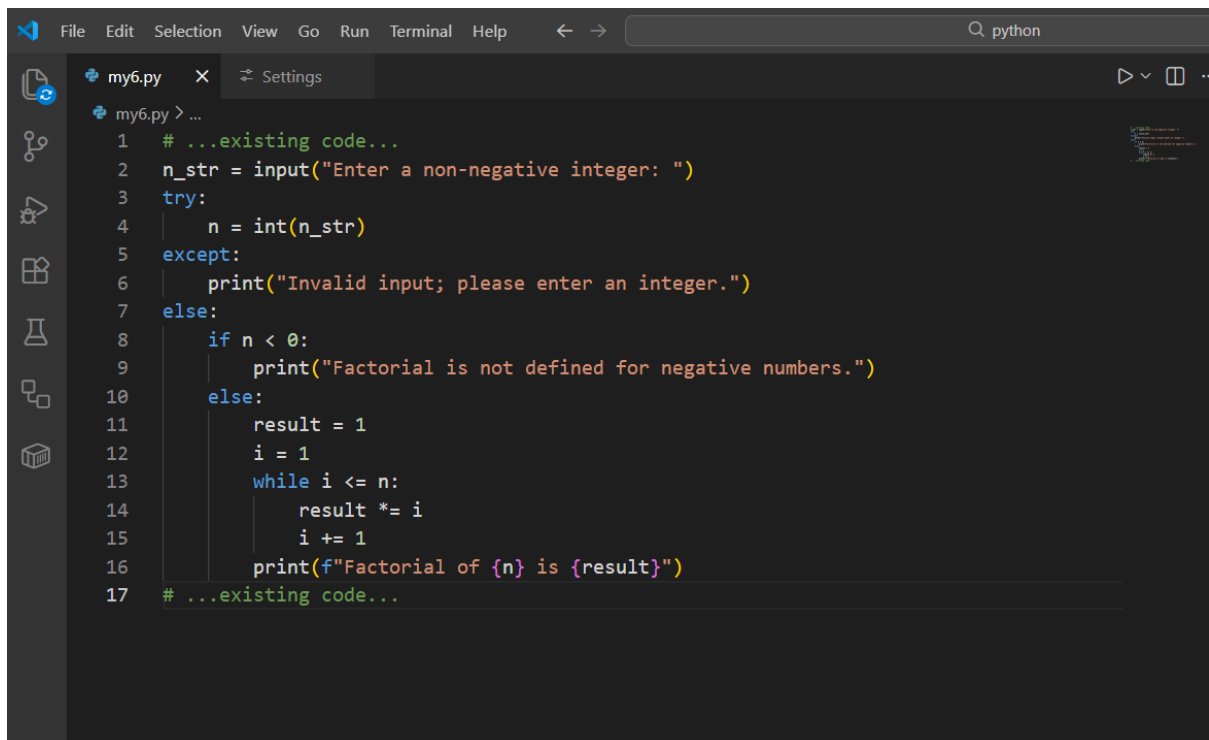
🔗 × 🔗 my6.py

Explore and understand your code

Ask ▾ Auto ▾

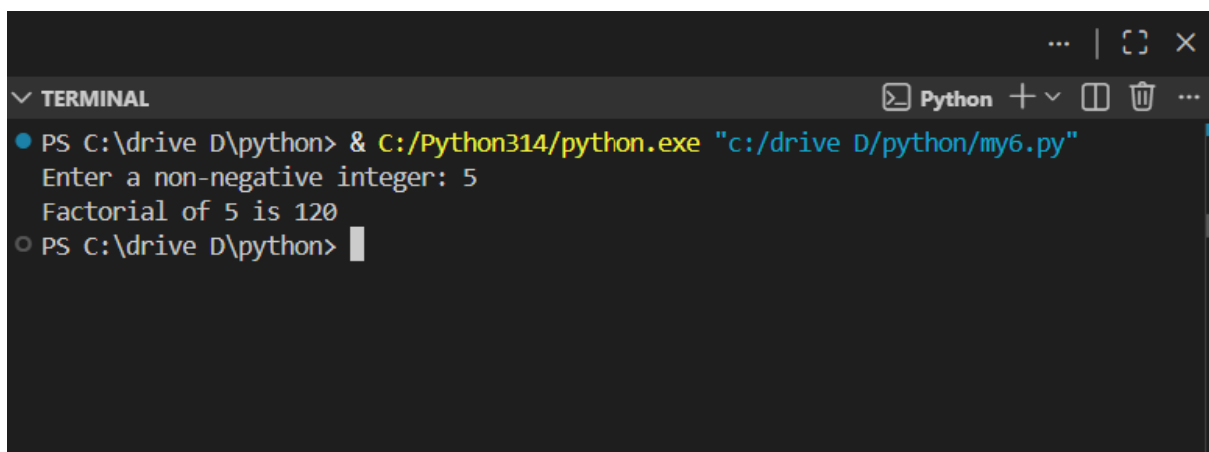
↩ ➡

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



The screenshot shows a code editor with a dark theme. The menu bar includes File, Edit, Selection, View, Go, Run, Terminal, and Help. A search bar on the right contains the text 'python'. The file explorer on the left shows a file named 'my6.py'. The main editor area displays the following Python code:

```
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...
```



The screenshot shows a terminal window with a dark theme. The title bar includes a dropdown menu with 'Python' and icons for adding, removing, and refreshing tabs. The terminal content is as follows:

```
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:

CHAT

+ v ⚙️ ... | [ ] ×

← CALCULATING FACTORIAL IN PYTHON WITHOUT FUNCTIONS

[ ]

🔖 Restore Checkpoint

Optimize this code

Simplify logic and improve readability

Click to Edit

🔗 my6.py

> Used 1 reference

✓ Optimized factorial calculation and provided optimization

Optimized version:

```
# ...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...
```

🔄 ↶ 👍 🗨

🔗 × 🔗 my6.py

Explore and understand your code

Ask v Auto v

➡ ➤

```
my6.py X Settings
my6.py > ...
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}")
```

```
... | [ ] X
TERMINAL Python + [ ] [ ] ...
● 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:

CHAT

← CALCULATING FACTORIAL IN PYTHON WITHOUT FUNCTIONS

Write a Python program to calculate factorial using a function

Use meaningful variable names

Add inline comments

my6.py

> Used 1 reference

```
# ...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

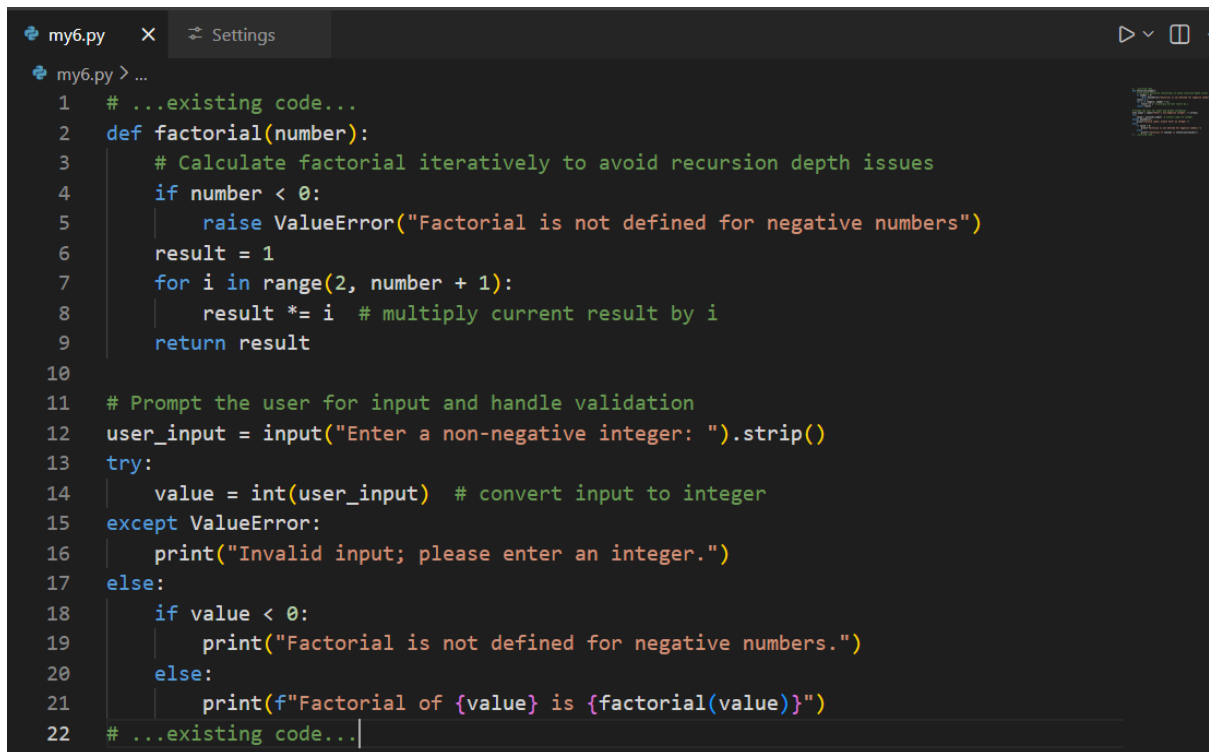
# Prompt the user for input and handle validation
user_input = input("Enter a non-negative integer: ").strip()
try:
    value = int(user_input) # convert input to integer
except ValueError:
    print("Invalid input; please enter an integer.")
else:
    if value < 0:
        print("Factorial is not defined for negative numbers.")
    else:
        print(f"Factorial of {value} is {factorial(value)}")
# ...existing code...
```

my6.py

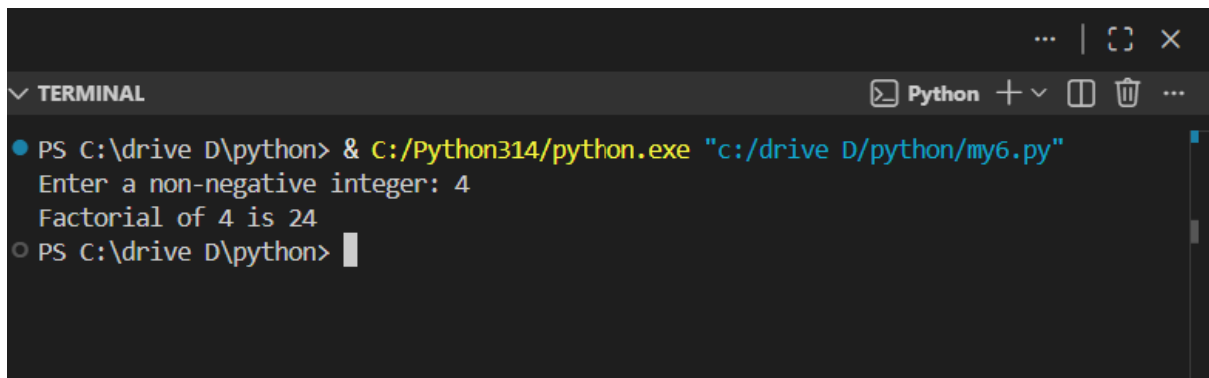
Explore and understand your code

Ask Auto





```
my6.py X Settings
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 +v {} ...
● 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.

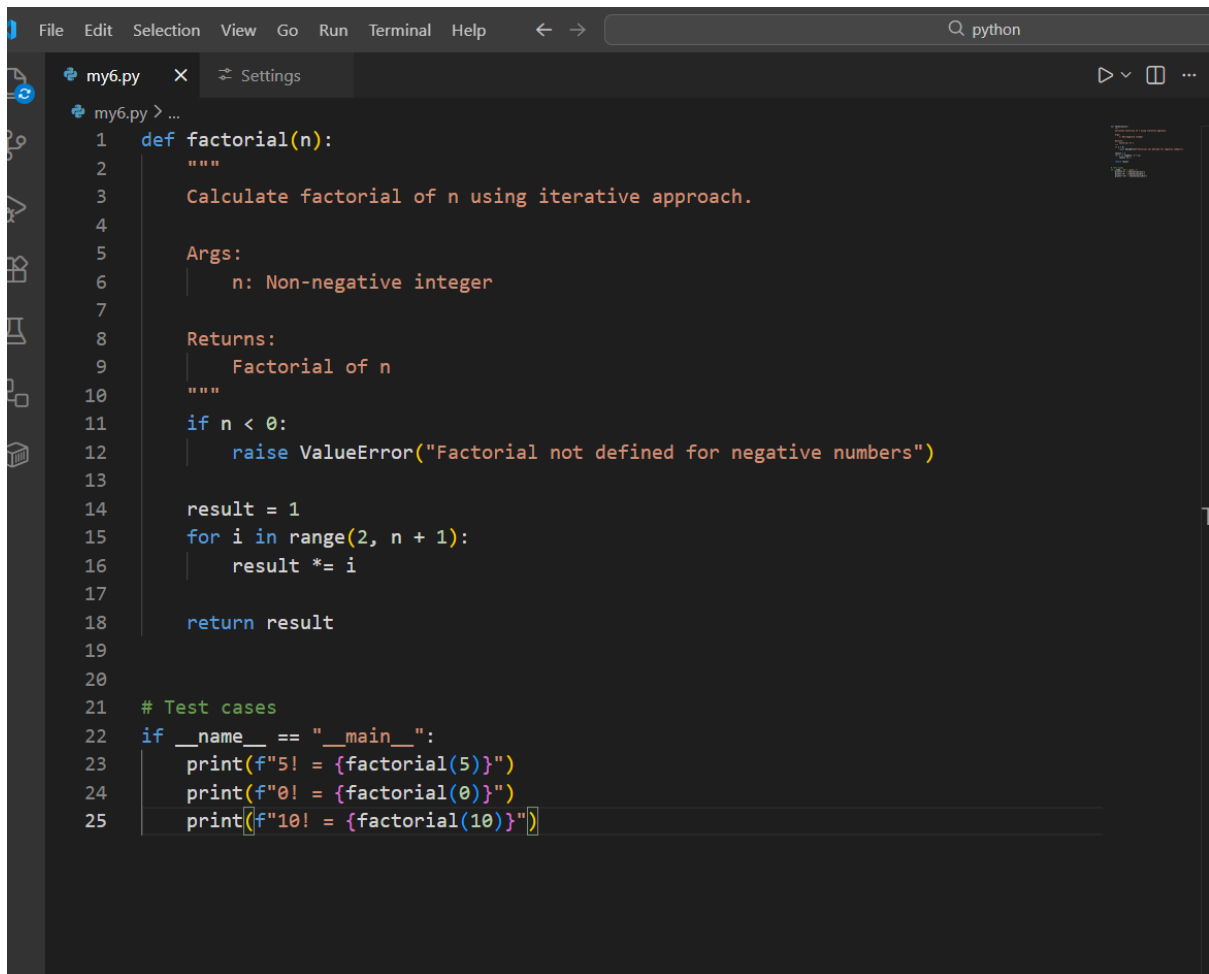
  my6.py

Explore and understand your code

Ask  Auto 

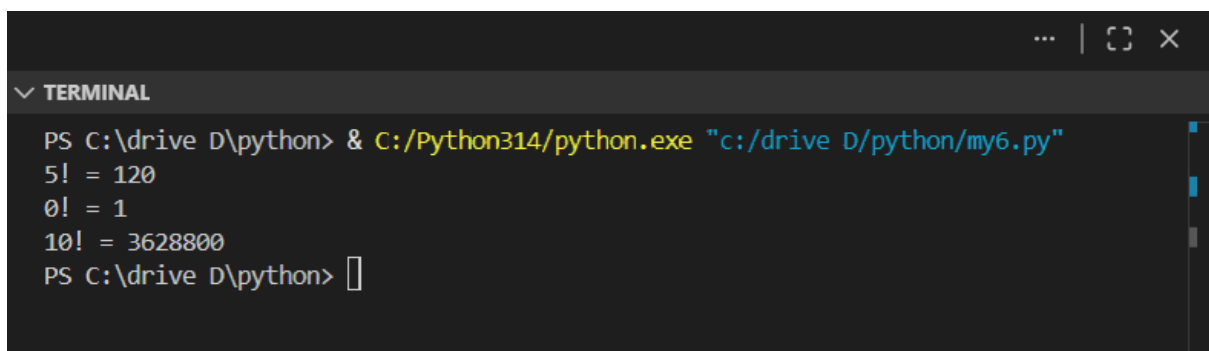
 

Ln 1, Col 1 Spaces: 4 UTF-8 CRLF { } Python  3.14.0  Go Live 



The screenshot shows a code editor with a dark theme. The menu bar at the top includes File, Edit, Selection, View, Go, Run, Terminal, and Help. A search bar on the right contains the text 'python'. The file explorer on the left shows a file named 'my6.py'. The main editor area displays the following Python code:

```
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(f"5! = {factorial(5)}")
24     print(f"0! = {factorial(0)}")
25     print(f"10! = {factorial(10)}")
```



The screenshot shows a terminal window with a dark background. The title bar at the top right has three icons: a menu icon, a maximize icon, and a close icon. The terminal content is as follows:

```
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

