AI Assisted Coding

Assignment-01

Name : G.Anoop Goud

Hallticket: 2303A51085

Batch-02

Task:01

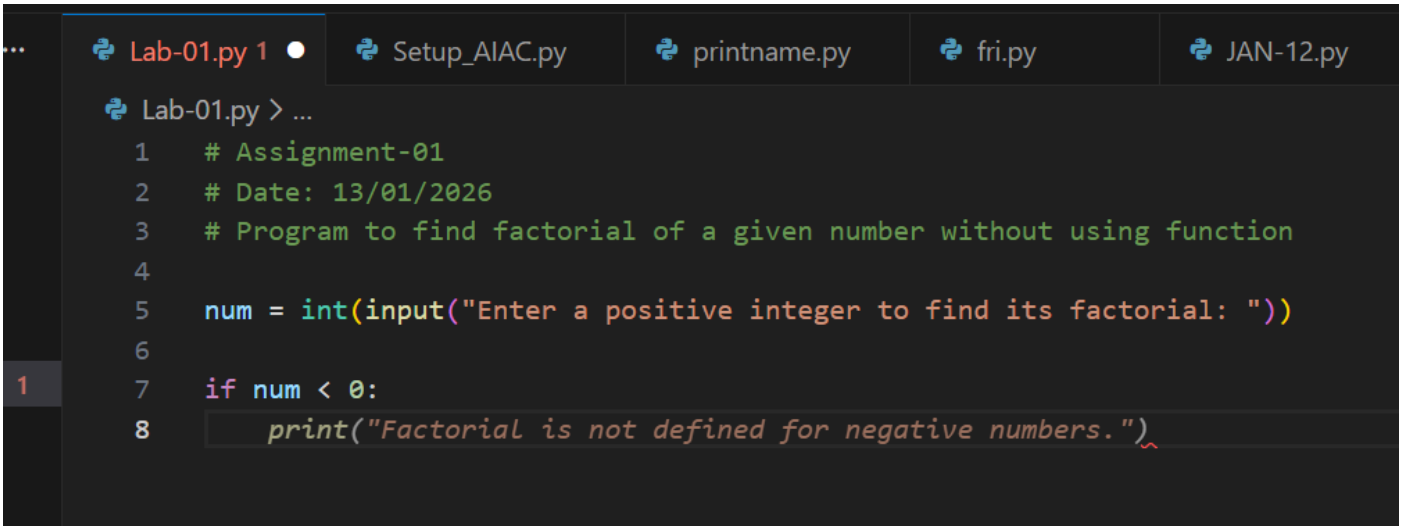
- **A working Python program generated with Copilot assistance**

```python
# Assignment-01
# Date: 13/01/2026
# Program to find factorial of a given number without using function

num = int(input("Enter a positive integer to find its factorial: "))

if num < 0:
    print("Please enter a positive integer")
else:
    factorial = 1
    for i in range(1, num + 1):
        factorial = factorial * i
    print("The factorial of", num, "is", factorial)
```

- **Screenshot(s) showing:**



Lab-01.py ● | Setup_AIAC.py | printname.py | fri.py | JAN-12.py

Lab-01.py > ...

```python
# Assignment-01
# Date: 13/01/2026
# Program to find factorial of a given number without using function

num = int(input("Enter a positive integer to find its factorial: "))

if num < 0:
    print("Please enter a positive integer")
else:
    factorial = 1
    for i in range(1, num + 1):
        factorial = factorial * i
    print("The factorial of", num, "is", factorial)

```

- **Sample input/output screenshots**

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\anoop\OneDrive\Desktop\AIAC> & C:/Users/anoop/A
ppData/Local/Programs/Python/Python314/python.exe c:/Users/
anoop/OneDrive/Desktop/AIAC/Lab-01.py
Enter a positive integer to find its factorial: 9
The factorial of 9 is 362880
PS C:\Users\anoop\OneDrive\Desktop\AIAC>
```

```
PS C:\Users\anoop\OneDrive\Desktop\AIAC> & C:/Users/anoop/AppData/L
ab-01.py
Enter a positive integer to find its factorial: 11
The factorial of 11 is 39916800
PS C:\Users\anoop\OneDrive\Desktop\AIAC>
```

- **Brief reflection (5–6 lines):**

GitHub Copilot helped me generate the program logic quickly without spending much time planning the structure. It encouraged a simple and organized coding style while keeping the code easy to read without using functions. By following its suggestions, I did not have to remember exact syntax, which allowed me to concentrate more on the main logic. Copilot also supported writing clean and efficient code. Overall, the development process became faster, easier, and more clear with less stress.

- **How helpful was Copilot for a beginner?**

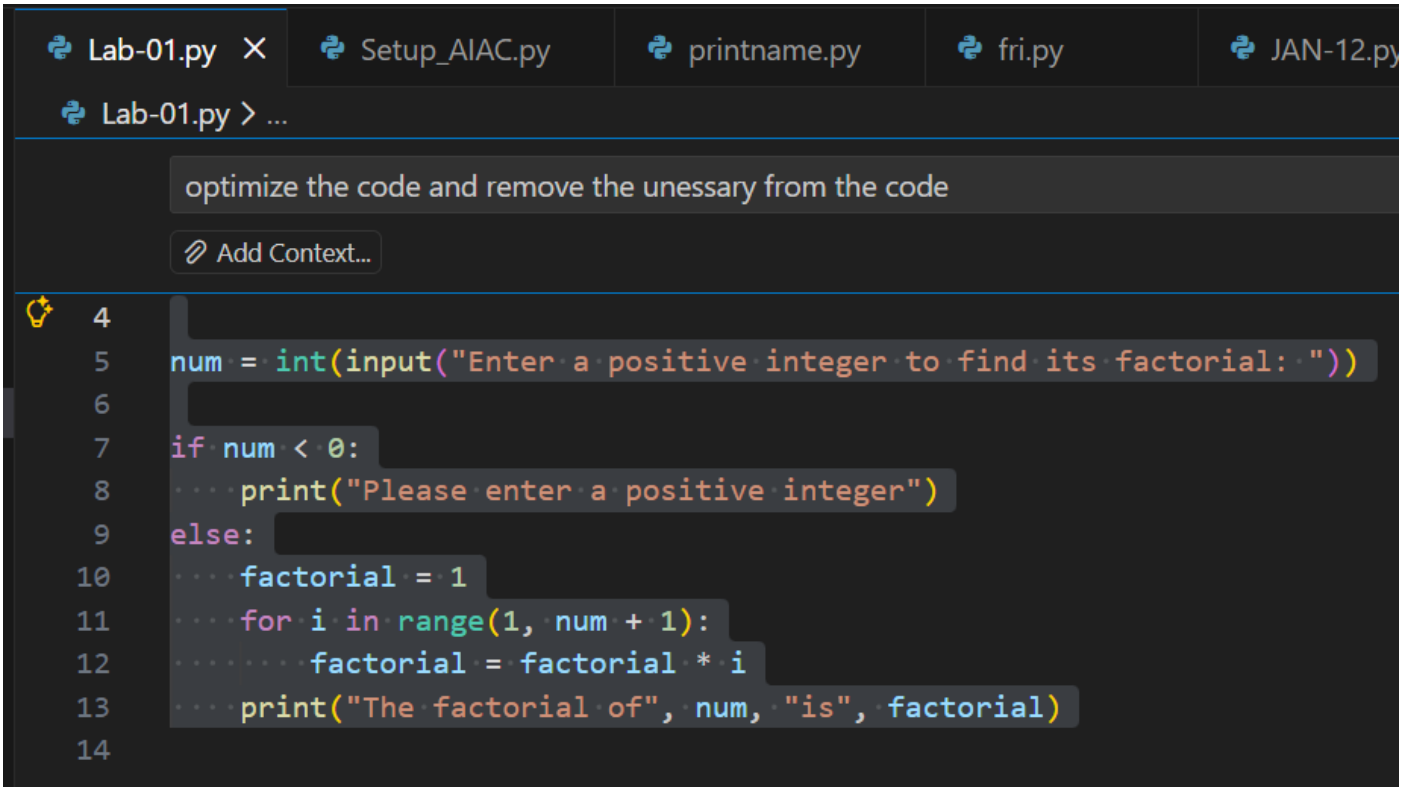
  ->copilot is very useful for beginners because it provides instant code suggestion when the user is stuck in middle. It will help beginner by suggesting the code so that user can understand the code and memorized and it makes learning programming easier and more confident

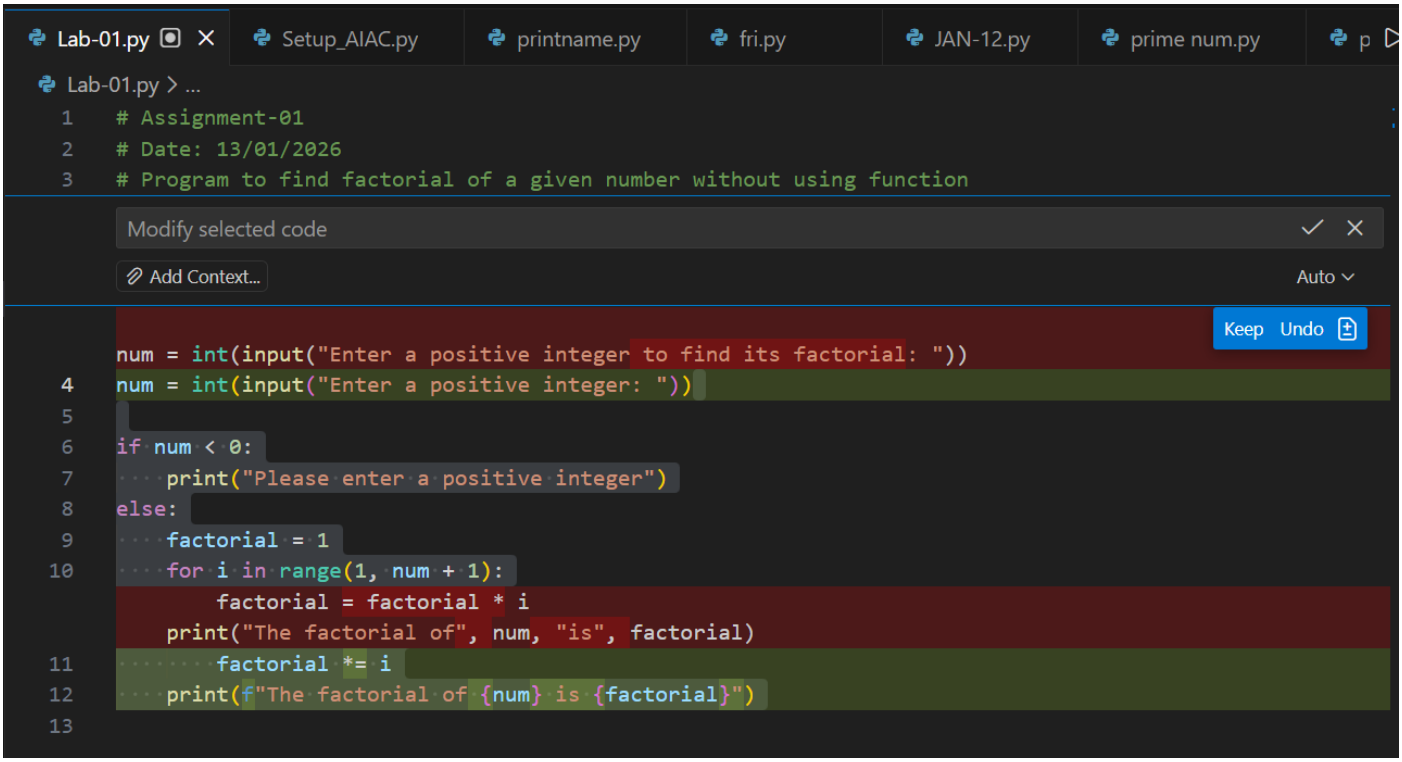
- **Did it follow best practices automatically?**

  ->Yes it follows best practices automatically for user by writing the code itself copilot will guess the next lines by that user can also more confident and user also get to know what next should be .It automatically best practices

Task-02

- **Screenshot(s) showing:**





code

```
# Assignment-01
# Date: 13/01/2026
# Program to find factorial of a given number without using function
num = int(input("Enter a positive integer: "))

if num < 0:
    print("Please enter a positive integer")
else:
    factorial = 1
    for i in range(1, num + 1):
        factorial *= i
    print(f"The factorial of {num} is {factorial}")
```

side by side comparition

```
n=int(input("Enter a number: "))
result = 1
if n < 0:
print("Factorial is not defined for negative numbers")
else:
for i in range(1, n + 1):
```

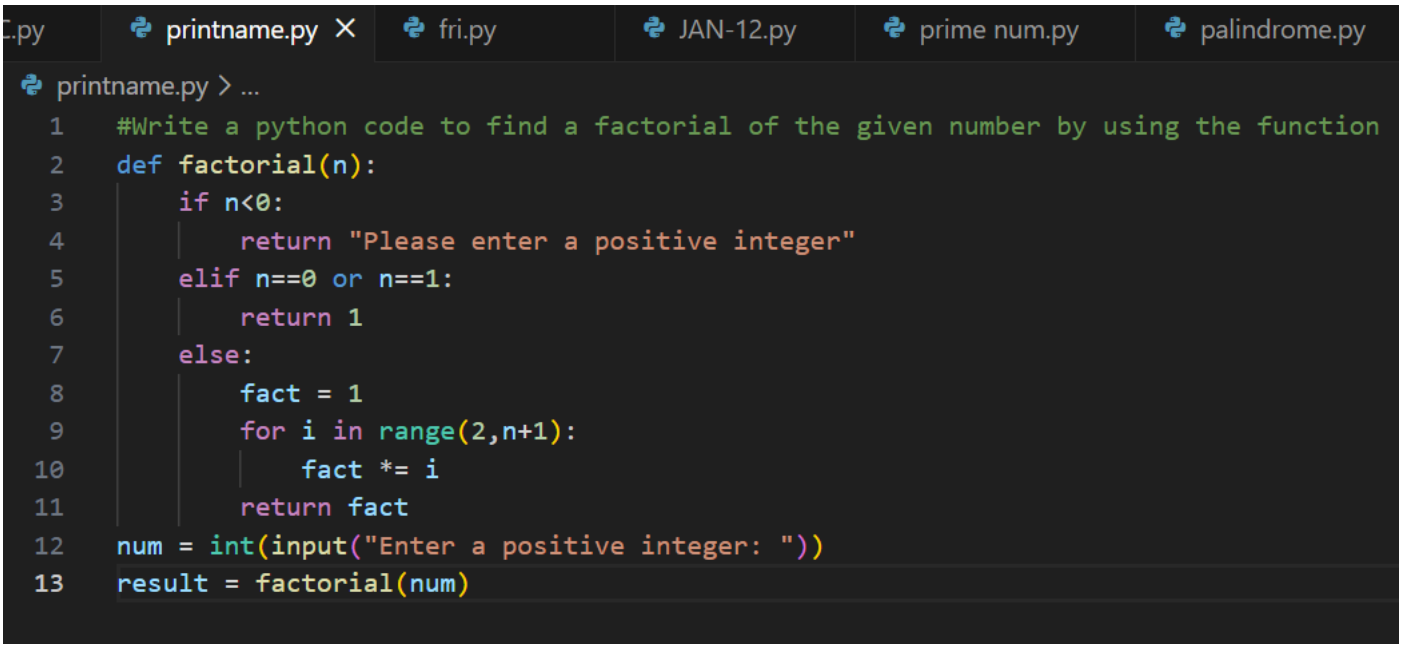
```
result = result * i
print("Factorial of", n, "is", result)
```

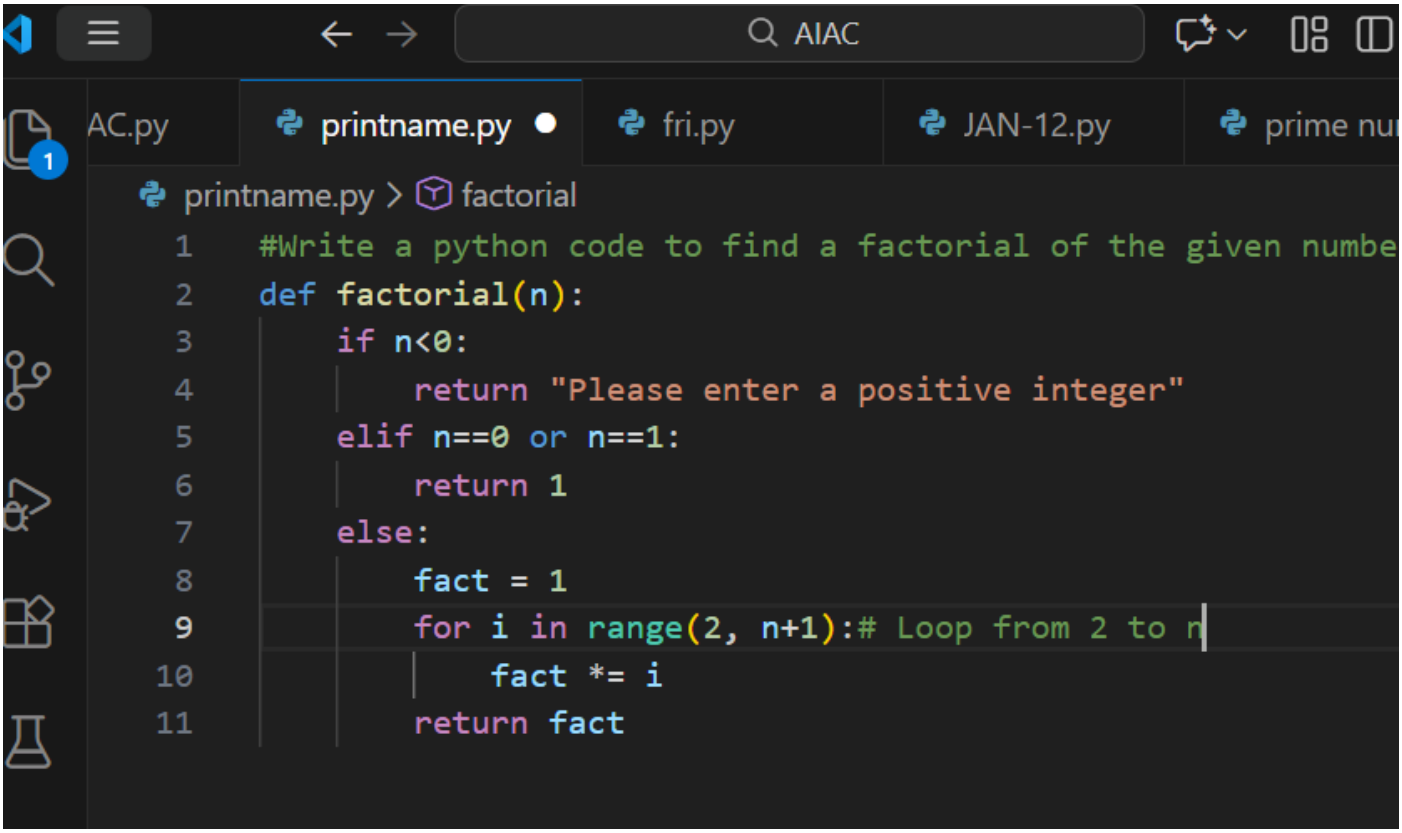
optimal code

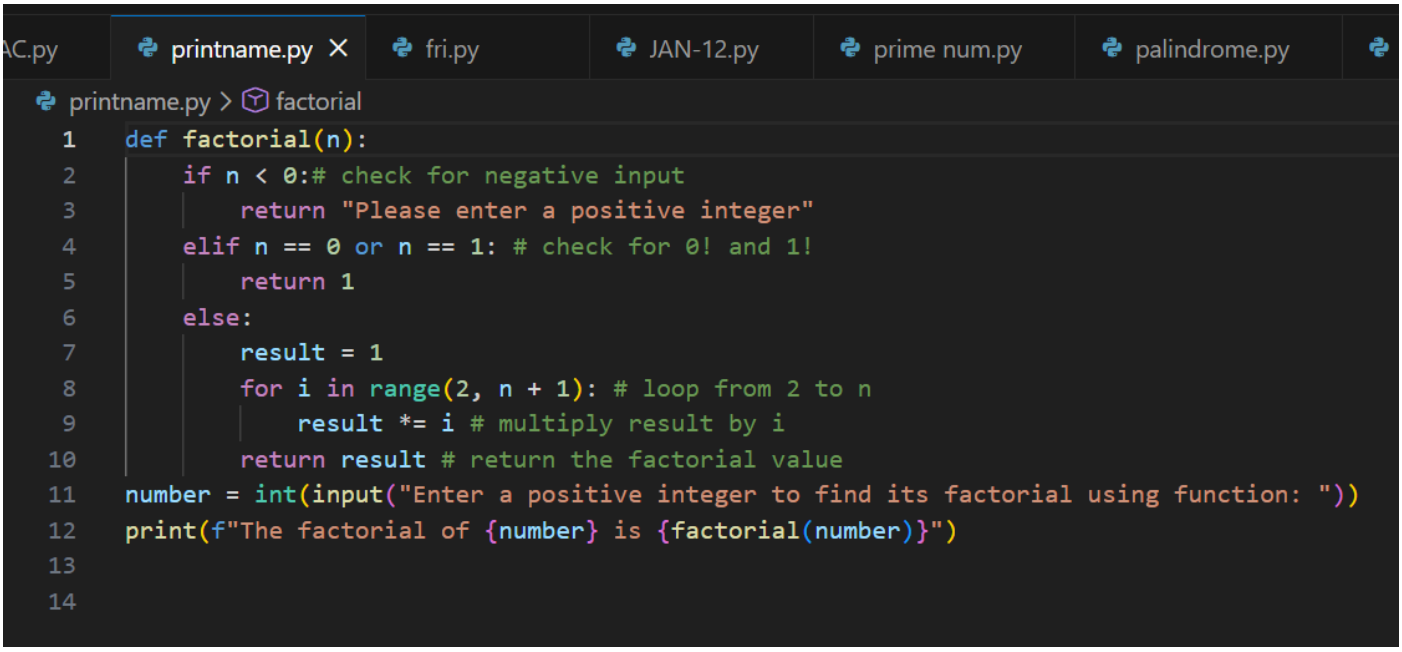
```
n = int(input("Enter a number: "))
if n < 0:
print("Factorial is not defined for negative numbers")
else:
result = 1
for i in range(2, n + 1):
result *= i
print(f"Factorial of {n} is {result}")
```

task-03

Modular Design Using AI Assistance (Factorial with Functions)








```python
#write a program to print factorial of the given number by using the fuction
def factorial(n):
    if n < 0:# check for negative input
        return "Please enter a positive integer"
    elif n == 0 or n == 1: # check for 0! and 1!
        return 1
    else:
        result = 1
```

```
        for i in range(2, n + 1): # loop from 2 to n
            result *= i # multiply result by i
        return result # return the factorial value
number = int(input("Enter a positive integer to find its factorial using
function: "))
print(f"The factorial of {number} is {factorial(number)}")
```

- **Sample input/output screenshots**

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\anoop\OneDrive\Desktop\AIAC> & C:/Users/anoop/AppData/Local/Programs/Python/Pyth
its factorial using function: 6
The factorial of 6 is 720
PS C:\Users\anoop\OneDrive\Desktop\AIAC> & C:/Users/anoop/AppData/Local/Programs/Python/Pyth
e c:/Users/anoop/OneDrive/Desktop/AIAC/printname.py
Enter a positive integer to find its factorial using function: 15
The factorial of 15 is 1307674368000
PS C:\Users\anoop\OneDrive\Desktop\AIAC>
```

Short note:

How modularity improves reusability.

Modularity allows a program to be organized into separate units that work independently. These units can be used again in other programs or reused wherever needed in the same program. This avoids writing the same code repeatedly and helps save time. It also makes programs easier to understand, update, and debug. Thus, modularity supports better code reuse and long-term scalability.

Task 4:

Comparative Analysis – Procedural vs Modular AI Code (With vs Without Functions

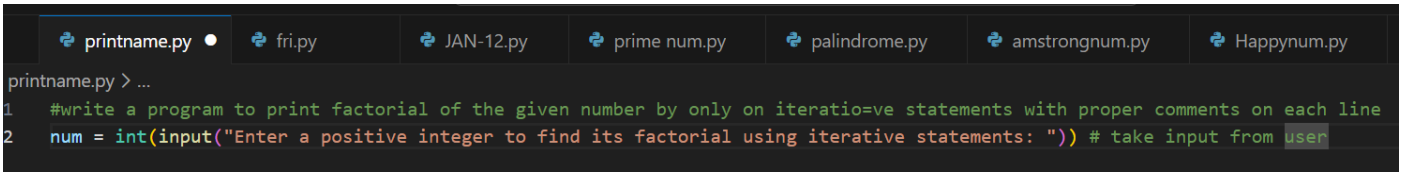
- **Comparative Analysis: Procedural vs Modular AI Code**

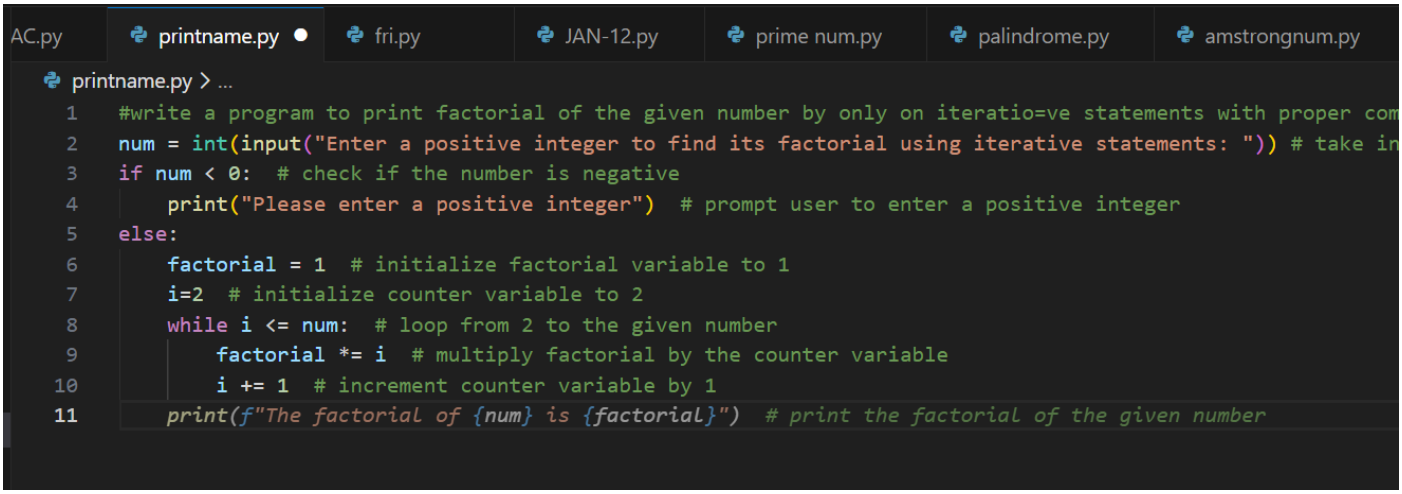
Criteria	Procedural Code (Without Functions)	Modular Code (With Functions)
Logic Clarity	The logic is written in a single block, which is easy to understand for	The logic is organized inside a function, making the program

	small programs but becomes difficult to follow as the code grows.	more structured and easier to understand.
Reusability	The code cannot be reused easily and must be rewritten if the same logic is needed again.	The function can be reused multiple times across the program or in other programs.
Debugging Ease	Debugging is harder because the entire code must be checked for errors.	Debugging is easier since errors are confined within the function.
Suitability for Large Projects	This approach is not suitable for large projects due to poor structure and maintainability.	This approach is suitable for large projects because modular design improves organization and scalability.
AI Dependency Risk	Since the logic is simple, dependency on AI is minimal.	Over-dependence on AI-generated functions without understanding the logic can introduce risks.

Task 5:

AI-Generated Iterative vs Recursive Thinking






```
#write a program to print factorial of the given number by only on
iteratio=ve statements with proper comments on each line
num = int(input("Enter a positive integer to find its factorial using only
iterative statements: ")) # take input from user
```

```python
if num < 0: # check for negative input
    print("Please enter a positive integer")
else: # if input is valid
    factorial = 1 # initialize factorial variable
    i = 2 # start from 2
    iteration_count = 0 # initialize iteration counter
    while i <= num: # loop until i is less than or equal to num
        factorial *= i # multiply factorial by i
        i += 1 # increment i by 1
        iteration_count += 1 # increment iteration counter
    print(f"The factorial of {num} is {factorial}") # print the result
    print(f"Number of iterations: {iteration_count}") # print iteration count
```

outputs

```
Enter a positive integer to find its factorial using only iterative statements: 20
The factorial of 20 is 2432902008176640000
Number of iterations: 19
PS C:\Users\anoop\OneDrive\Desktop\AIAC>
```

```
Enter a positive integer to find its factorial using only iterative statements: 100
The factorial of 100 is 933262154439441526816992388562667004907159682643816214685929638952175
8625369792082722375825118521091686400000000000000000000000
Number of iterations: 99
PS C:\Users\anoop\OneDrive\Desktop\AIAC>
```

Using Recursion:

Setup_AIAC.py | printname.py ● | fri.py | JAN-12.py | prime num

printname.py > ...

```python
#write a program to print factorial of the given number by using recursion
def factorial(n):
    if n<0:
        return "Factorial is not defined for negative numbers"
    elif n==0 or n==1:
        return 1
    else:
        return n * factorial(n-1)  # Recursive call to factorial function
# Get input from user
num = int(input("Enter a positive integer: "))
```

C.py | printname.py ● | fri.py | JAN-12.py | prime num.py

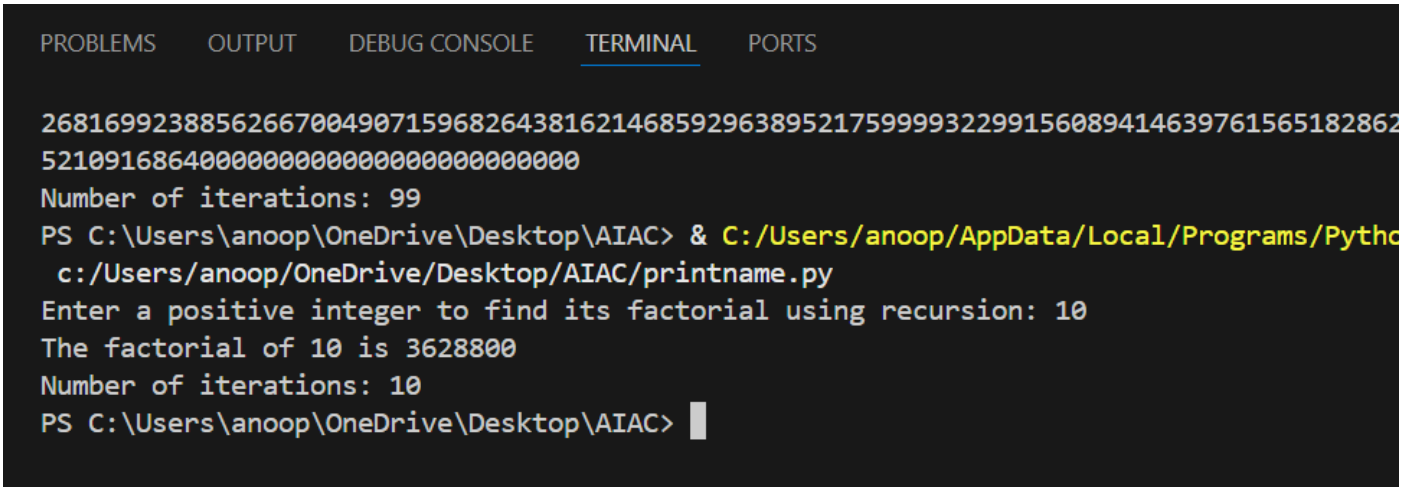
printname.py > ...

```python
def factorial(n):
    if n<0:
        return "Please enter a positive integer"
    elif n==0 or n==1:
        return 1
    else:
        return n * factorial(n-1)
num = int(input("Enter a positive integer to find its factorial: "))
result =(f"The factorial of {num} is {factorial(num)}")
print(result)
```

```python
#write a program to print factorail of the given number by using recursion with proper comments on each line
def factorial(n, count=[0]): # define factorial function with iteration counter
    count[0] += 1 # increment iteration counter
    if n < 0: # check for negative input
        return "Please enter a positive integer"
    elif n == 0 or n == 1: # base case for recursion
        return 1
    else:
        return n * factorial(n - 1, count) # recursive case
count = [0] # initialize iteration counter
number = int(input("Enter a positive integer to find its factorial using recursion: ")) # take input from user
print(f"The factorial of {number} is {factorial(number, count)}") # print the result
print(f"Number of iterations: {count[0]}") # print iteration count
```


```python
#write a program to print factorail of the given number by using recursion with proper comments on each line
def factorial(n, count=[0]): # define factorial function with iteration counter
    count[0] += 1 # increment iteration counter
    if n < 0: # check for negative input
        return "Please enter a positive integer"
    elif n == 0 or n == 1: # base case for recursion
        return 1
    else:
        return n * factorial(n - 1, count) # recursive case
count = [0] # initialize iteration counter
number = int(input("Enter a positive integer to find its factorial using recursion: ")) # take input from user
print(f"The factorial of {number} is {factorial(number, count)}") # print the result
print(f"Number of iterations: {count[0]}") # print iteration count
```


output:



Comparison: Iterative vs Recursive Approach

Criteria	Iterative Approach	Recursive Approach
Readability	Simple and clear, especially for beginners to follow the logic.	Code looks concise and elegant but can be hard for beginners to understand.
Stack Usage	Does not use call stack, memory usage is minimal.	Depends on the call stack for each call, leading to higher memory usage
Performance Implications	Faster and more efficient for large inputs.	Comparatively slower due to multiple function calls and stack overhead.
When Recursion Is Not Recommended	Suitable and safe even for large input values.	Avoided for large inputs because it may cause stack overflow or hit recursion limits.