AI ASSISTED CODING

ASSIGNMENT – 01

Name :- CH VISHWANATH RAO

HT NO:- 2303A51095

BATCH:- 02

TASK: 01

Screenshots showing:-

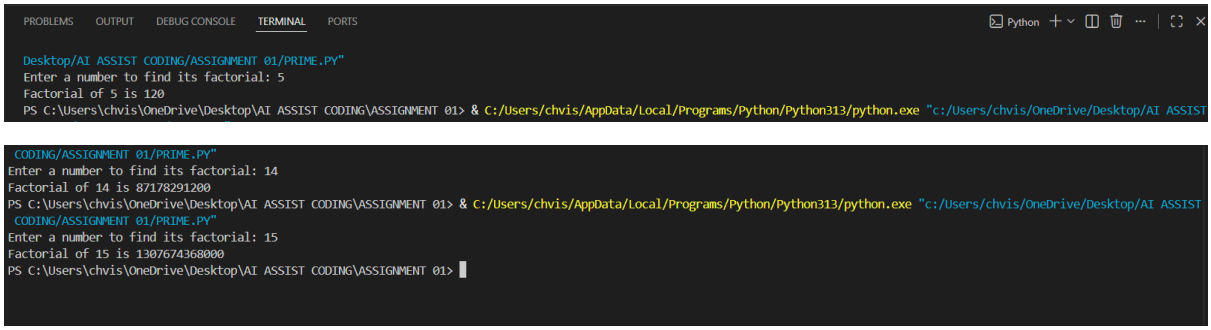
```
#ASSIGNMENT:01
#DATE:-12/1/2026
#Write a python code to print the factorial of a given number without using the recursion function
num = int(input("Enter a number to find its factorial: "))
factorial = 1
if num < 0:
    print("Factorial is not defined for negative numbers")
```

```
#ASSIGNMENT:01
#DATE:-12/1/2026
#Write a python code to print the factorial of a given number without using the recursion function
num = int(input("Enter a number to find its factorial: "))
factorial = 1
if num < 0:
    print("Factorial is not defined for negative numbers")
elif num == 0:
    print("Factorial of 0 is 1")
else:
    for i in range(1,num+1):
        factorial = factorial * i
    print("Factorial of",num,"is",factorial)
```

WORKING PYTHON PROGRAM GENERATED WITH COPILOT ASSIST

```
#ASSIGNMENT:01
#DATE:-12/1/2026
#Write a python code to print the factorial of a given number without using the recursion function
num = int(input("Enter a number to find its factorial: "))
factorial = 1
if num < 0:
    print("Factorial is not defined for negative numbers")
elif num == 0:
    print("Factorial of 0 is 1")
else:
    for i in range(1,num+1):
        factorial = factorial * i
    print("Factorial of",num,"is",factorial)
```

OUTPUT:-



GitHub Copilot Usage:
GitHub Copilot helped me generate the program logic quickly without spending much time on syntax or structure. It promoted a clean and simple coding approach, making the code easy to read even without functions. By suggesting ready-to-use code, Copilot allowed to focus more on understanding the logic rather than memorizing exact syntax. It also improved coding efficiency and Overall, Copilot made the process faster, clearer, and more enjoyable.

Usefulness of Copilot for Beginners:
Copilot is very useful for beginners because it provides instant code suggestions whenever the user gets stuck. It helps beginners understand how the code works, making it easier to learn and remember concepts. This guidance boosts confidence and supports faster learning in programming.

Following Best Coding Practices:
Yes, Copilot follows good coding practices on its own. It guesses and writes the next correct lines of code. This helps us learn how to write proper and neat code while improving our understanding of programming.

TASK – 02:

Screenshot showing

```
num = int(input("Enter a number to find its factorial: "))
factorial = 1
if num < 0:
    print("Factorial is not defined for negative numbers")
elif num == 0:
    print("Factorial of 0 is 1")
else:
    for i in range(1,num+1):
        factorial = factorial * i
    print("Factorial of",num,"is",factorial)
```

optimize the code and remove unwanted variables

Add Context...

```
num = int(input("Enter a number to find its factorial: "))
if num < 0:
    print("Factorial is not defined for negative numbers")
elif num == 0:
    print("Factorial of 0 is 1")
else:
    factorial = 1
    for i in range(1, num + 1):
        factorial *= i
    print(f"Factorial of {num} is {factorial}")
    result = math.factorial(num)
    print(f"Factorial of {num} is {result}")
```

CODE:

```
#ASSIGNMENT:01
#DATE:-12/1/2026
#Write a python code to print the factorial of a given number without using the recursion function

    num = int(input("Enter a number to find its factorial: "))
    if num < 0:
        print("Factorial is not defined for negative numbers")
    elif num == 0:
        print("Factorial of 0 is 1")
    else:
        factorial = 1
        for i in range(1, num + 1):
            factorial *= i
        print(f"Factorial of {num} is {factorial}")
        result = math.factorial(num)
        print(f"Factorial of {num} is {result}")
```

Optimal code:

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

TASK 3:

Modular Design Using AI Assistance (Factorial with Functions)

```
#write  program to print factorial of the given number by using function
def factorial(n):
    if n == 0 or n == 1:
        return 1
    else:
        return n * factorial(n - 1)
num = int(input("Enter a number to find its factorial: "))
result = factorial(num)
print(f"The factorial of {num} is {result}")
```

```
#write  program to print factorial of the given number by using function
def factorial(n):
    result = 1
    for i in range(2, n + 1):
        result *= i
    return result
num = int(input("Enter a num to find its factorial: "))
fact = factorial(num)
print(f"The factorial of {num} is {fact}")
```

OPTIMAL CODE:

```python
def factorial(n):  # Define a function named factorial
    if n == 0 or n == 1:  # Base case: if n is 0 or 1, the factorial is 1
        return 1
    else:
        return n * factorial(n - 1)  # n multiplied by factorial of (n-1)

num = int(input("Enter a num to find its factorial: "))
fact = factorial(num)  # Call the fact func with the user's input & store the result in fact
print(f"The factorial of {num} is {fact}")  # Print the result
```

OUTPUT:

```
Enter a number to find its factorial: 6
The factorial of 6 is 720
PS C:\Users\chvis\OneDrive\Desktop\AI ASSIST CODING\ASSIGNMENT 01> & C:/Users/chvis/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/chvis/OneDrive/
Desktop/AI ASSIST CODING/ASSIGNMENT 01/USE FUNC.py"
Enter a number to find its factorial: 16
The factorial of 16 is 20922789888000
PS C:\Users\chvis\OneDrive\Desktop\AI ASSIST CODING\ASSIGNMENT 01> & C:/Users/chvis/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/chvis/OneDrive/
Desktop/AI ASSIST CODING/ASSIGNMENT 01/USE FUNC.py"
Enter a number to find its factorial: 32
The factorial of 32 is 263130836933693530167218012160000000
PS C:\Users\chvis\OneDrive\Desktop\AI ASSIST CODING\ASSIGNMENT 01> & C:/Users/chvis/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/chvis/OneDrive/
```

Importance of Modularity

Modularity increases reusability by structuring software into clearly separated blocks that perform dedicated tasks. Each block can be independently used in different programs or reused within the same application whenever needed. This eliminates unnecessary code repetition and improves productivity. Because modules are isolated, testing and fixing errors becomes easier, and modifications can be made without affecting the whole system. Thus, modularity plays a key role in creating reliable, maintainable, and reusable software.

TASK 4:

Comparative Analysis – Procedural vs Modular AI code (with vs without functions)

Criteria	Procedural Code (Without Functions)	Modular Code (With Functions)
Logic Clarity	Logic is written in a single block. It is easy for small programs but becomes difficult as the code grows.	Logic is divided into functions, making the program clear and well structured.
Reusability	Code cannot be reused easily and must be rewritten when needed again.	Functions can be reused many times in the same or different programs.
Debugging Ease	Debugging is difficult because the whole code must be checked.	Debugging is easier since errors are limited to functions.
Suitability for Large Projects	Not suitable due to poor structure and maintenance issues.	Suitable because modular design improves organization and scalability.
AI Dependency Risk	Low risk as the logic is simple and easy to understand.	Higher risk if AI-generated functions are used without understanding logic.

TASK 5:

```
#write a Program to find factorial using only iterative statements

num = int(input("Enter a number: "))   # take input

if num < 0:                            # check for negative input
    print("Enter a positive number")
else:
    fact = 1                           # initialize factorial variable
    i = 1                              # start from 1
    while i <= num:                    # loop till i <= num
        fact *= i                      # multiply factorial by i
        i += 1                         # increment i by 1
    print(f"Factorial of {num} = {fact}")   # print result
```

Optimal code:

```python
num = int(input("Enter a number: "))   # take input

if num < 0:                            # check for negative input
    print("Enter a positive number")
else:
    fact = 1                           # initialize factorial variable
    i = 1                              # start from 1
    while i <= num:                    # loop till i <= num
        fact *= i                      # multiply factorial by i
        i += 1                         # increment i by 1
    print(f"Factorial of {num} = {fact}")   # print result
```

Outputs:

Using recursion:

```python
# Program to calculate factorial of a number using recursion

# Define recursive function
def find_fact(n):
    if n <= 1:                      # base condition
        return 1
    else:
        return n * find_fact(n - 1) # recursive call

# Take user input
val = int(input("Enter a number: "))

# Check if input is valid
if val < 0:
    print("Enter a positive integer.")
else:
    ans = find_fact(val)            # call recursive function
    print(f"Factorial of {val} = {ans}")
```

```python
def find_fact(n):
    if n <= 1:
        return 1
    else:
        return n * find_fact(n - 1)

val = int(input("Enter a number: "))

if val < 0:
    print("Enter a positive integer.")
else:
    ans = find_fact(val)
    print(f"Factorial of {val} = {ans}")
```

Optimal code:

```python
# Program to find factorial of a number using recursion

def find_fact(n):                           # define recursive function
    if n <= 1:                              # base case: factorial of 0 or 1 is 1
        return 1
    else:
        return n * find_fact(n - 1)         # recursive call to function

val = int(input("Enter a number: "))         # take input from user

if val < 0:                                 # check for negative input
    print("Enter a positive integer.")
else:
    ans = find_fact(val)                    # call function and store result
    print(f"Factorial of {val} = {ans}")    # display the result
```

Iterative approach vs recursion approach:

Feature	Iterative Approach	Recursive Approach
Definition	Uses loops (for, while) repeatedly	Function calls itself repeatedly
Termination	Stops when loop condition fails	Stops when base condition is met
Memory Usage	Uses less memory	Uses more memory (function call stack)
Speed	Generally faster	Slightly slower due to function calls
Readability	Easier for simple problems	Easier for problems with natural recursion (like factorial, Fibonacci)
Example	Uses for or while loop	Uses recursive function call