Assignment-1

Name: Yashaswini Nagireddi

Ht.no: 2303A51418

Batch. No: 21

1Q) Task 1: AI-Generated Logic Without Modularization (Factorial without

Functions)

Prompt used:

write a python code to calculate the factorial of a number without using any functions and input should be taken from user.

Code:

```
code1.py > ...
1  # write a python code to calculate the factorial of a number with out using any
2  n=int(input("enter a number: "))
3  factorial=1
4  for i in range(1,n+1):
5      factorial=factorial*i
6  print("factorial of",n,"is",factorial)
7
```

Output:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS
PS D:\AI assist> python -u "d:\AI assist\code1.py"
enter a number: 5
factorial of 5 is 120
```

Justification:

By doing this task, the factorial of a number is calculated without using any functions.
The program takes input from the user and uses a single for loop to compute the factorial.

All steps are written directly in the main program using iterative logic.

2Q) Task 2: AI Code Optimization & Cleanup (Improving Efficiency)

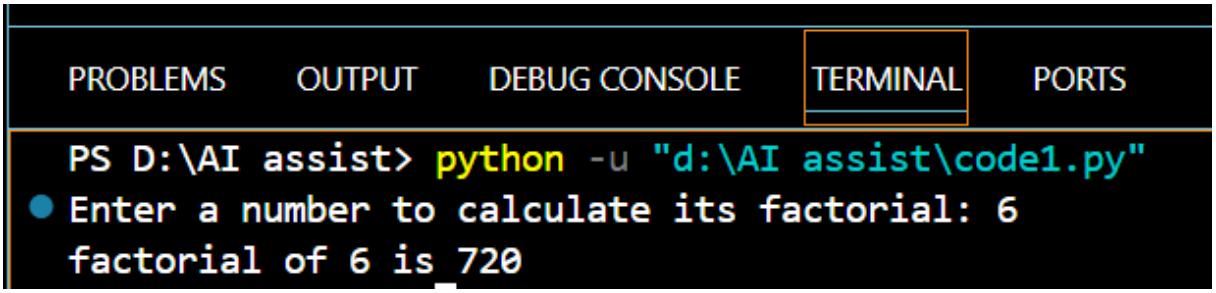
Prompt used:

optimize the above code by adding comments, readability and with improved logic.

Code:

```
# optimize the above code by adding comments, readability and with improved logic.
n=int(input("Enter a number to calculate its factorial: "))
factorial=1
for i in range(1,n+1):
    factorial=factorial*i
print("factorial of",n,"is",factorial)
```

Output:



Justification:

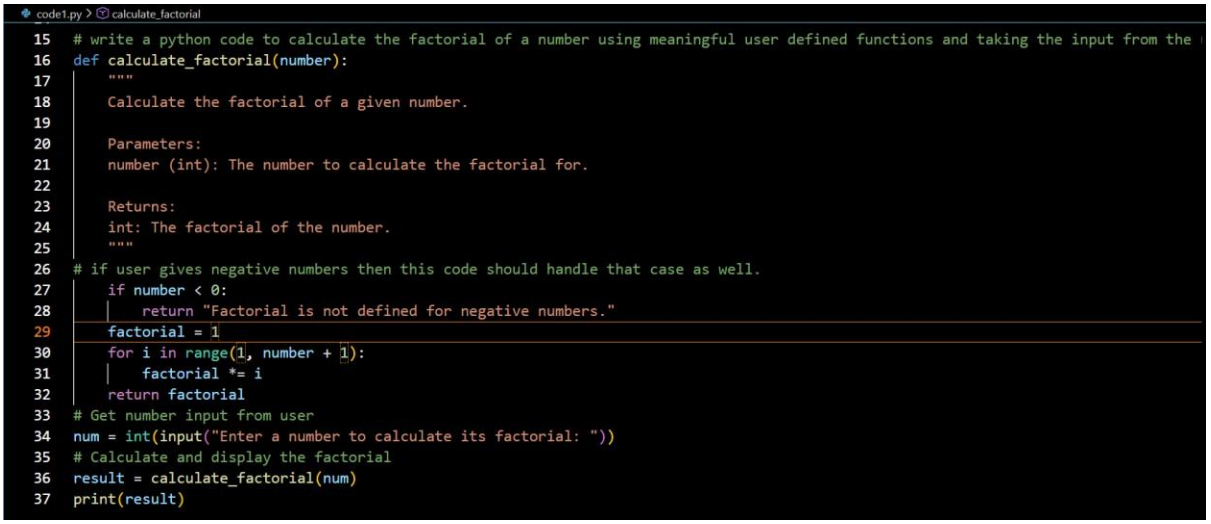
By doing this task, the factorial program is optimized for better readability and understanding.
Meaningful comments and clear variable usage are added while keeping the logic simple and efficient.
The program still uses a single loop and user input, with all steps written in the main program.

3Q) Task 3: Modular Design Using AI Assistance (Factorial with Functions)

Prompt used:

- write a python code to calculate the factorial of a number using meaningful user defined functions and taking the input from the user.
- if user gives negative numbers then this code should handle that case as well.

Code:

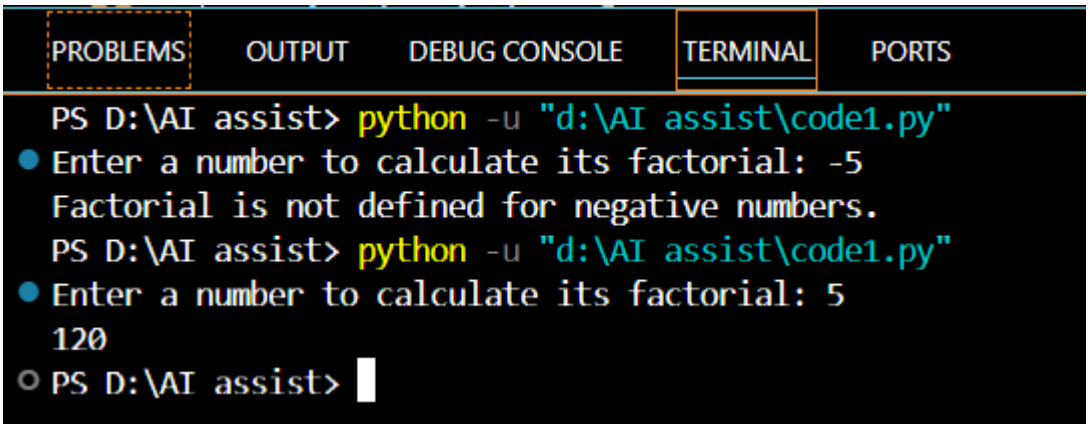


```python
# write a python code to calculate the factorial of a number using meaningful user defined functions and taking the input from the
def calculate_factorial(number):
    """
    Calculate the factorial of a given number.

    Parameters:
    number (int): The number to calculate the factorial for.

    Returns:
    int: The factorial of the number.
    """
# if user gives negative numbers then this code should handle that case as well.
    if number < 0:
        return "Factorial is not defined for negative numbers."
    factorial = 1
    for i in range(1, number + 1):
        factorial *= i
    return factorial
# Get number input from user
num = int(input("Enter a number to calculate its factorial: "))
# Calculate and display the factorial
result = calculate_factorial(num)
print(result)
```

Output:



```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS
PS D:\AI assist> python -u "d:\AI assist\code1.py"
Enter a number to calculate its factorial: -5
Factorial is not defined for negative numbers.
PS D:\AI assist> python -u "d:\AI assist\code1.py"
Enter a number to calculate its factorial: 5
120
PS D:\AI assist>
```

Justification:

By doing this task, the factorial logic is implemented using a meaningful user-defined function.
The program takes input from the user and handles negative numbers with proper validation.

Modular design improves code readability, reusability, and error handling.

4Q) Task 4: Comparative Analysis – Procedural vs Modular AI Code (With vs Without Functions)

Criteria	Without Functions (Procedural Code)	With Functions (Modular Code)
Logic Clarity	Logic is written linearly in one place, which is easy to follow for small programs but becomes cluttered as code grows.	Logic is clearly separated into meaningful functions, making the flow easier to understand.
Reusability	Code cannot be reused easily; the same logic must be rewritten if needed again.	Functions can be reused multiple times across the program or in other programs.
Debugging Ease	Debugging is harder because all logic is mixed in the main program.	Easier to debug since errors can be isolated within specific functions.
Suitability for Large Projects	Not suitable for large projects due to poor structure and maintainability.	Highly suitable for large projects because of clean structure and modular design.
AI Dependency Risk	Higher risk, as AI-generated logic may be copied blindly without understanding due to lack of structure.	Lower risk, as modular design encourages understanding of logic, inputs, and outputs.

Conclusion:
Procedural code is useful for small, simple tasks and beginners, while

modular code using functions is better suited for scalable, maintainable, and professional software development.

5Q) Task 5: AI-Generated Iterative vs Recursive Thinking

- **An iterative version of the logic**

Prompt used:

write a python code to calculate the factorial of a number using iterative approach with meaningful user defined functions and taking the input from the user.

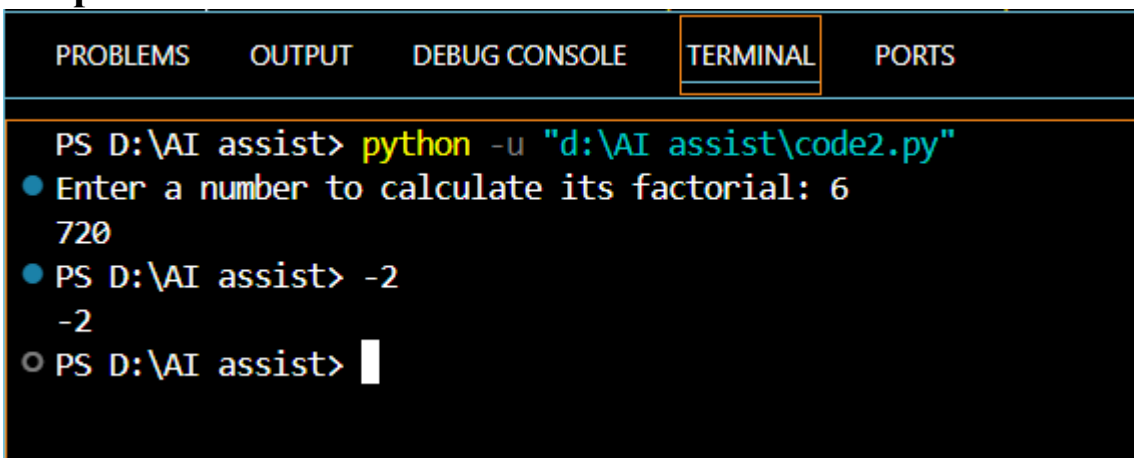
Code:

```
# write a python code to calculate the factorial of a number using iterative approach with meaningful
def calculate_factorial_iterative(number):
    """
    Calculate the factorial of a given number using an iterative approach.

    Parameters:
    number (int): The number to calculate the factorial for.

    Returns:
    int: The factorial of the number.
    """
    if number < 0:
        return "Factorial is not defined for negative numbers."
    factorial = 1
    for i in range(1, number + 1):
        factorial *= i
    return factorial
# Get number input from user
num = int(input("Enter a number to calculate its factorial: "))
# Calculate and display the factorial
result = calculate_factorial_iterative(num)
print(result)
```

Output:



- **A recursive version of the same logic**
  Prompt used:
  write a python code to calculate the factorial of a number using recursive approach with meaningful user defined functions and taking the input from the user.

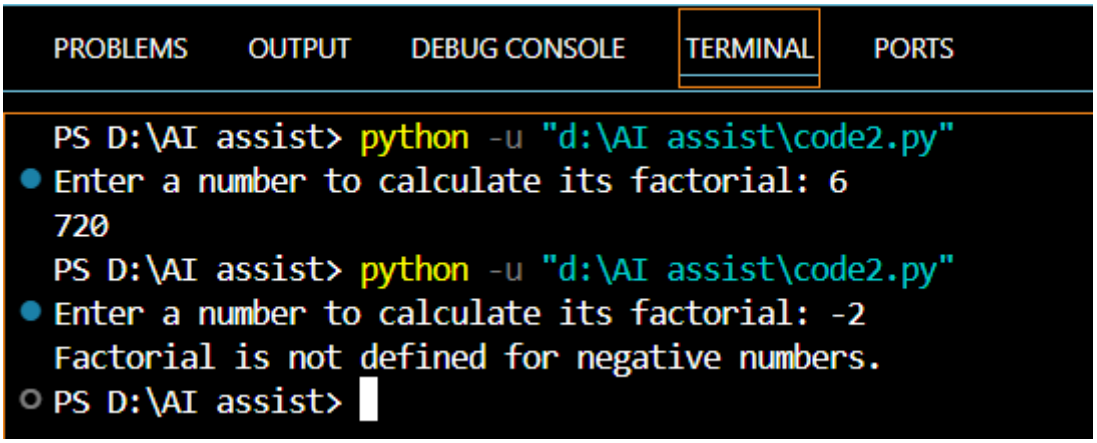
Code:

```
# write a pyhton code to calculate the factorial of a number using recursive approach with meaningful
def calculate_factorial_recursive(number):
    """
    Calculate the factorial of a given number using a recursive approach.

    Parameters:
    number (int): The number to calculate the factorial for.

    Returns:
    int: The factorial of the number.
    """
    if number < 0:
        return "Factorial is not defined for negative numbers."
    if number == 0 or number == 1:
        return 1
    return number * calculate_factorial_recursive(number - 1)
# Get number input from user
num = int(input("Enter a number to calculate its factorial: "))
# Calculate and display the factorial
result = calculate_factorial_recursive(num)
print(result)
```

Output:



Justification:

By doing this task, factorial logic is implemented using both iterative and recursive approaches.
The iterative version uses a loop for step-by-step multiplication, while

the recursive version uses function calls and base conditions.
This comparison helps understand differences in readability, stack usage, performance, and when recursion should be avoided.