## School of Computer Science and Artificial Intelligence

---

# Lab Assignment # 9

---

**Name of Student** : **GINNE SAI TEJA**
**Enrollment No.** : **2303A51526**
**Batch No.** : **22**

---

### Task Description -1 (Documentation – Function Summary Generation)

**Task:**

**Use AI to generate concise functional summaries for each Python function in a given script.**

**Instructions:**

- **Provide a Python script to the AI.**
- **Ask the AI to write a short summary describing the purpose of each function.**
- **Ensure summaries are brief and technically accurate.**
- **Do not include code implementation details.**

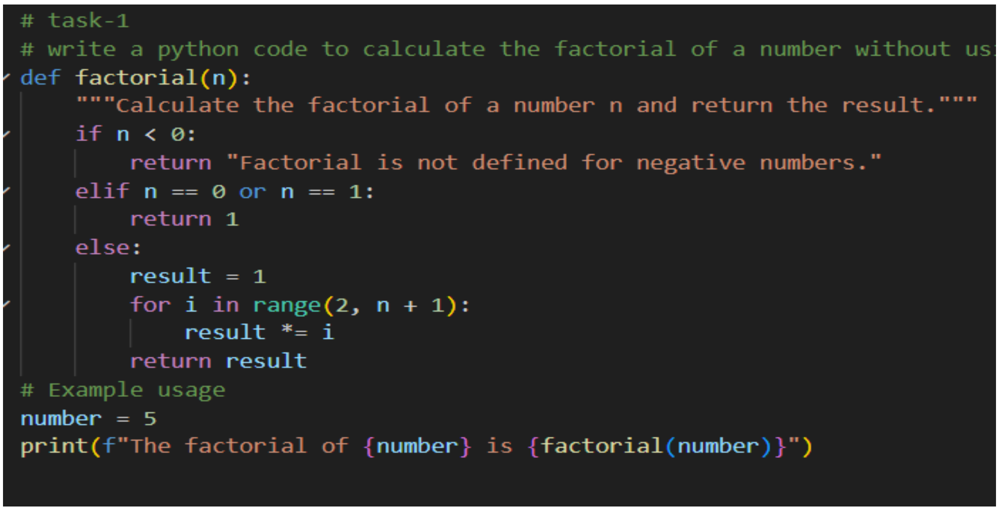
**Expected Output -1:**

**A Python script where each function contains a clear and concise summary explaining its purpose.**

**Prompt:-**

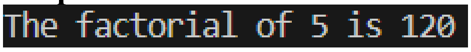
write a python code to calculate the factorial of a number without using functions and write a function docstring concise summary.

**Code:-**



```
# task-1
# write a python code to calculate the factorial of a number without us
def factorial(n):
    """Calculate the factorial of a number n and return the result."""
    if n < 0:
        return "Factorial is not defined for negative numbers."
    elif n == 0 or n == 1:
        return 1
    else:
        result = 1
        for i in range(2, n + 1):
            result *= i
        return result
# Example usage
number = 5
print(f"The factorial of {number} is {factorial(number)}")
```

**Output:-**



```
The factorial of 5 is 120
```

**Justification:-**

The function docstring provides a concise summary of what the function does, which is to calculate the factorial of a number n and return the result. It also includes information about how the function handles negative numbers and the base cases for 0 and 1, making it clear and informative for users who may want to use or modify the function in the future.

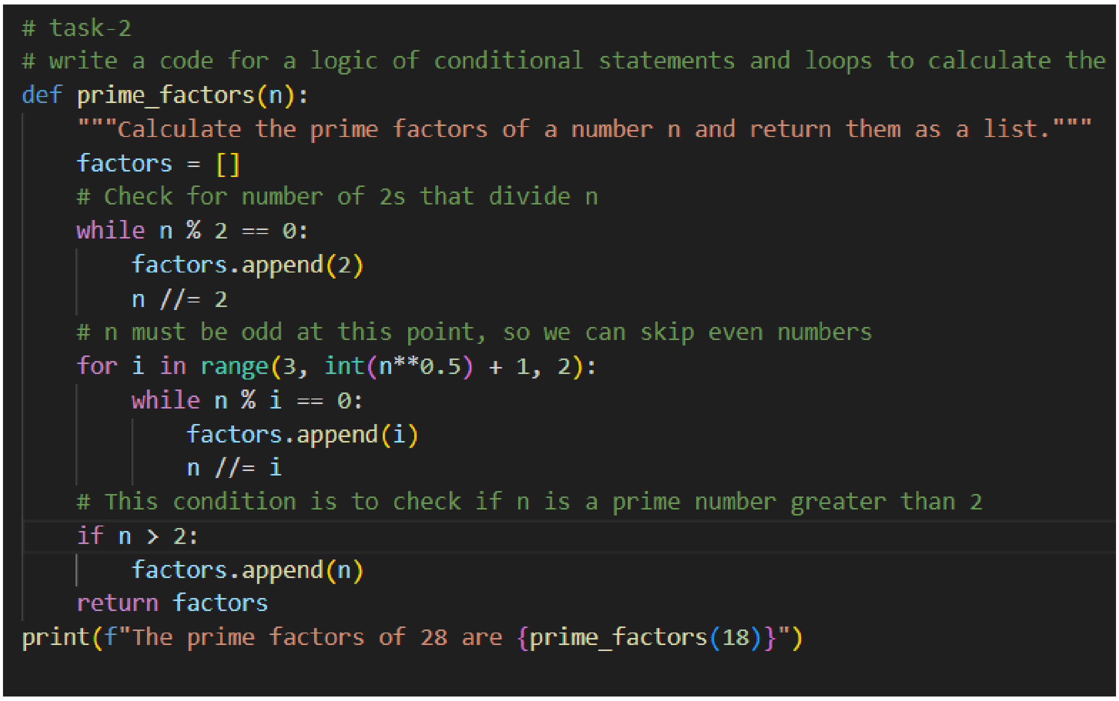
**Task Description -2 (Documentation – Logical Explanation for Conditions and Loops)**
**Task:**
**Use AI to document the logic behind conditional statements and loops in a Python program.**
**Instructions:**

- **Provide a Python program without comments.**
- **Instruct AI to explain only decision-making logic and loop behavior.**
- **Skip basic syntax explanations.**

**Expected Output -2:**
**Python code with clear explanations describing the logic of conditions and loops.**

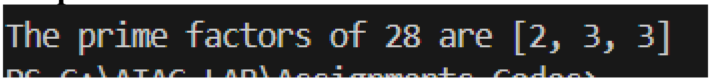
**Prompt:-**
write a code for a logic of conditional statements and loops to calculate the prime factors of a number and write a function docstring concise summary and only decision making logic and loops behaviour, skip basic syntax explanation.

**Code:-**



```
# task-2
# write a code for a logic of conditional statements and loops to calculate the
def prime_factors(n):
    """Calculate the prime factors of a number n and return them as a list."""
    factors = []
    # Check for number of 2s that divide n
    while n % 2 == 0:
        factors.append(2)
        n //= 2
    # n must be odd at this point, so we can skip even numbers
    for i in range(3, int(n**0.5) + 1, 2):
        while n % i == 0:
            factors.append(i)
            n //= i
    # This condition is to check if n is a prime number greater than 2
    if n > 2:
        factors.append(n)
    return factors
print(f"The prime factors of 28 are {prime_factors(18)}")
```

**Output:-**



```
The prime factors of 28 are [2, 3, 3]
```

**Justification:-**
The function `prime_factors` uses a combination of loops and conditional statements to determine the prime factors of a given number. It first checks for the number of times 2 divides the input number, then iterates through odd numbers starting from 3 up to the square root of the input number to check for other prime factors. Finally, it checks if the remaining number is greater than 2, which would indicate that it is a prime factor itself. This approach efficiently identifies all prime factors while minimizing unnecessary iterations.

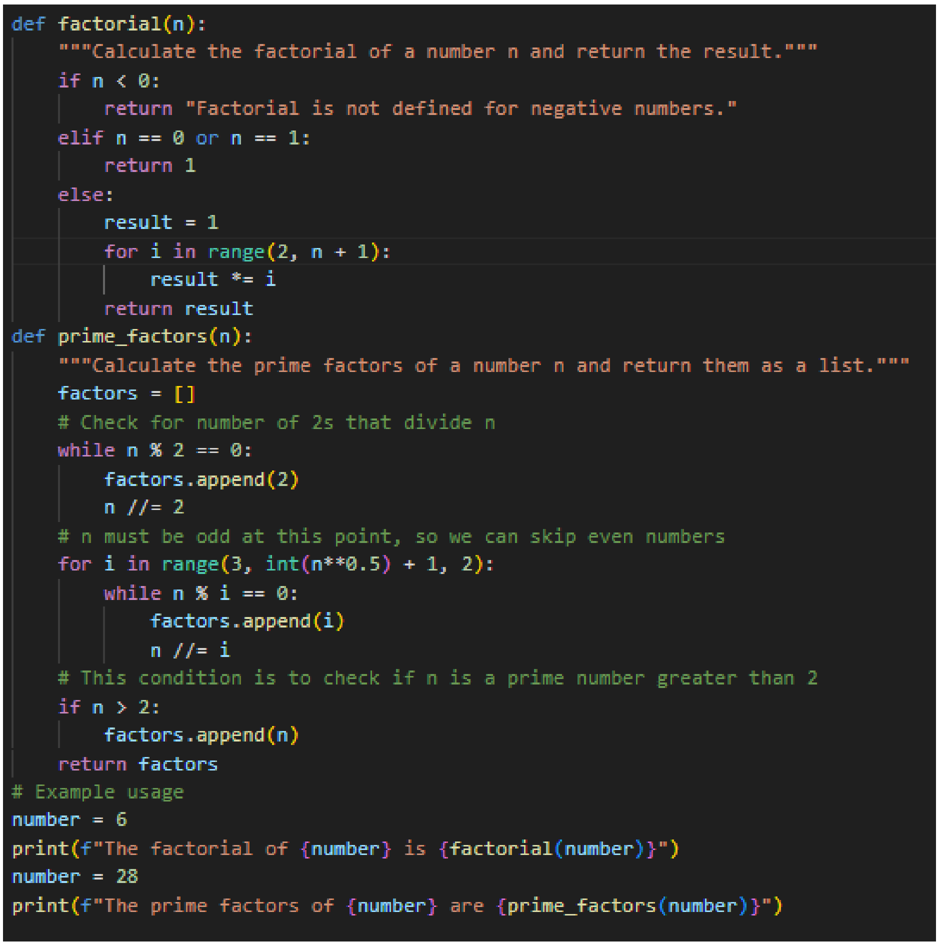
**Task Description -3 (Documentation – File-Level Overview)**
**Task:**
**Use AI to generate a high-level overview describing the functionality of an entire Python file.**
**Instructions:**
- **Provide the complete Python file to AI.**
- **Ask AI to write a brief overview summarizing the file's purpose and functionality.**
- **Place the overview at the top of the file.**

**Expected Output -3:**
**A Python file with a clear and concise file-level overview at the beginning.**

**Prompt:-**
generate a complete python file below clear, concise file-level overview and insert it at the very top as a module docstring.

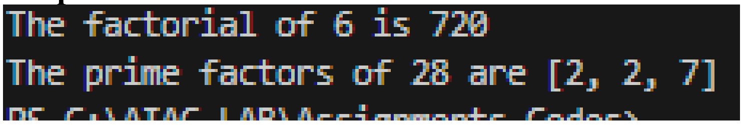
Comment :-
This program contains functions to calculate the factorial of a number and to find the prime factors of a number. The factorial function uses an iterative approach to compute the factorial, while the prime factors function uses a combination of loops and conditional statements to determine the prime factors of a given number. Both functions include error handling for invalid inputs and are designed for easy reuse in larger projects.
The program also includes example usage of both functions, demonstrating how to call them and display their results.

**Code:-**



```python
def factorial(n):
    """Calculate the factorial of a number n and return the result."""
    if n < 0:
        return "Factorial is not defined for negative numbers."
    elif n == 0 or n == 1:
        return 1
    else:
        result = 1
        for i in range(2, n + 1):
            result *= i
        return result
def prime_factors(n):
    """Calculate the prime factors of a number n and return them as a list."""
    factors = []
    # Check for number of 2s that divide n
    while n % 2 == 0:
        factors.append(2)
        n //= 2
    # n must be odd at this point, so we can skip even numbers
    for i in range(3, int(n**0.5) + 1, 2):
        while n % i == 0:
            factors.append(i)
            n //= i
    # This condition is to check if n is a prime number greater than 2
    if n > 2:
        factors.append(n)
    return factors
# Example usage
number = 6
print(f"The factorial of {number} is {factorial(number)}")
number = 28
print(f"The prime factors of {number} are {prime_factors(number)}")
```

**Output:-**



```
The factorial of 6 is 720
The prime factors of 28 are [2, 2, 7]
```

**Justification:-**

The module docstring provides a clear and concise overview of the program's functionality, describing the purpose of each function and how they operate. It also mentions the inclusion of example usage, which helps users understand how to utilize the functions effectively. This level of documentation enhances readability and makes it easier for other developers to understand and reuse the code in their projects.

**Task Description -4 (Documentation – Refine Existing Documentation)**

**Task:**

**Use AI to improve clarity and consistency of existing documentation in Python code.**

**Instructions:**

- **Ask AI to rewrite the documentation to improve clarity and consistency.**
- **Ensure technical meaning remains unchanged.**

**Expected Output -4:**

**Python code with refined and improved documentation that is clear and consistent.**

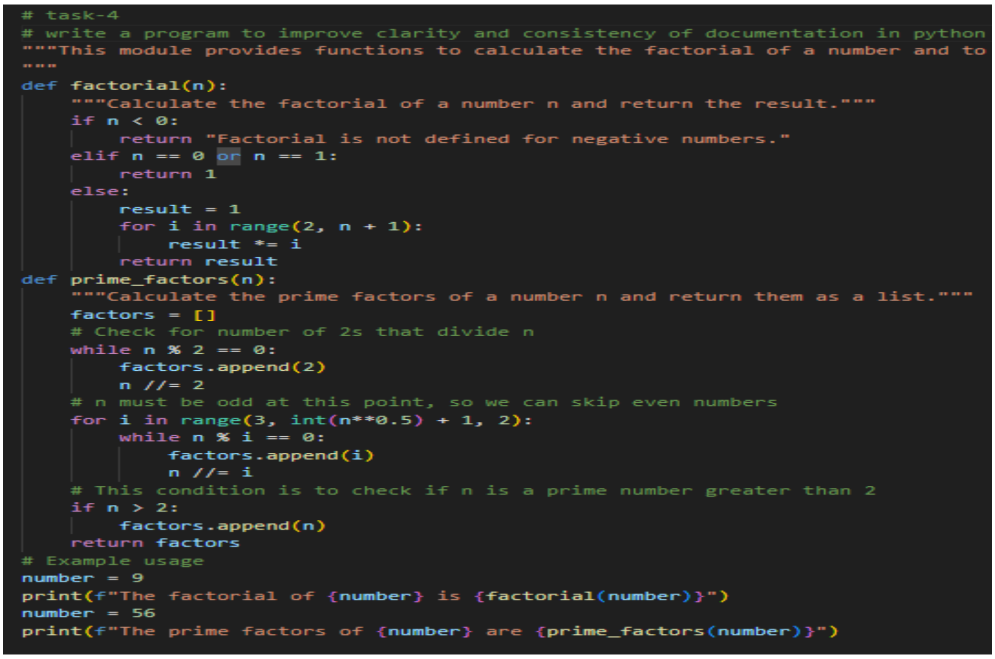
**Prompt:-**

write a program to improve clarity and consistency of documentation in python code, containing basic or unclear comments.

Comment:-

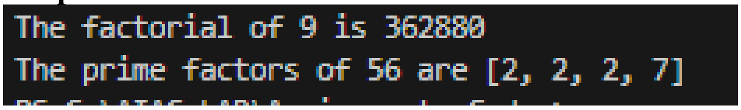
This module provides functions to calculate the factorial of a number and to determine the prime factors of a number. The factorial function computes the product of all positive integers up to a given number, while the prime factors function identifies all prime numbers that divide the input number without leaving a remainder. Both functions include error handling for invalid inputs and are designed for easy reuse in various applications.

**Code:-**



```python
# task-4
# write a program to improve clarity and consistency of documentation in python
"""This module provides functions to calculate the factorial of a number and to
"""
def factorial(n):
    """Calculate the factorial of a number n and return the result."""
    if n < 0:
        return "Factorial is not defined for negative numbers."
    elif n == 0 or n == 1:
        return 1
    else:
        result = 1
        for i in range(2, n + 1):
            result *= i
        return result
def prime_factors(n):
    """Calculate the prime factors of a number n and return them as a list."""
    factors = []
    # Check for number of 2s that divide n
    while n % 2 == 0:
        factors.append(2)
        n //= 2
    # n must be odd at this point, so we can skip even numbers
    for i in range(3, int(n**0.5) + 1, 2):
        while n % i == 0:
            factors.append(i)
            n //= i
    # This condition is to check if n is a prime number greater than 2
    if n > 2:
        factors.append(n)
    return factors
# Example usage
number = 9
print(f"The factorial of {number} is {factorial(number)}")
number = 56
print(f"The prime factors of {number} are {prime_factors(number)}")
```

**Output:-**



```
The factorial of 9 is 362880
The prime factors of 56 are [2, 2, 2, 7]
```

**Justification:-**

The module docstring provides a clear and concise overview of the functionality offered by the code, explaining what each function does and how they can be used. The function docstrings are also clear and informative, detailing the purpose of each function, the expected input, and the output. This level of documentation enhances readability and makes it easier for other developers to understand and utilize the code effectively.

**Task Description -5 (Documentation – Prompt Detail Impact Study)**

**Task:**

**Study the impact of prompt detail on AI-generated documentation quality.**

**Instructions:**

**Create two prompts: one brief and one detailed.**

- **Use both prompts to document the same Python function.**
- **Compare the generated outputs.**

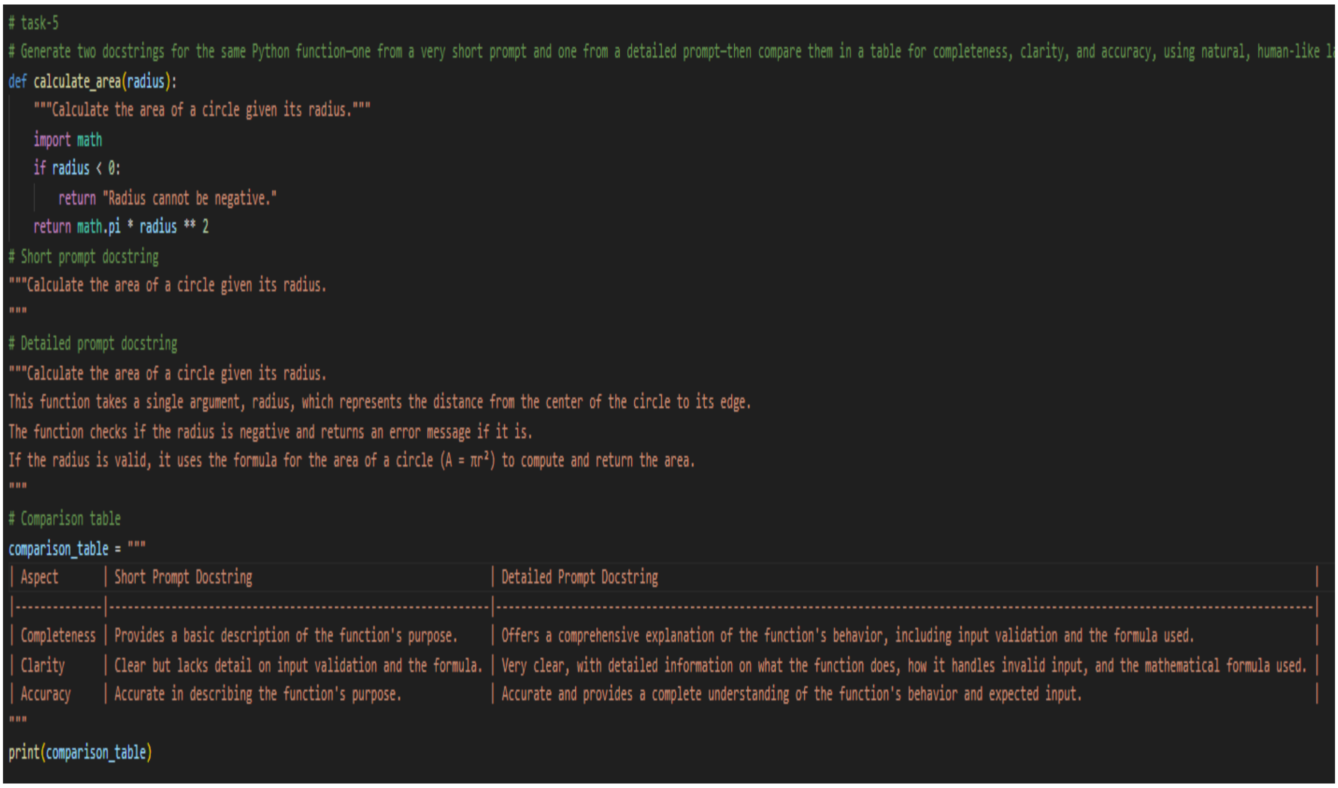
**Expected Output -5:**

**A comparison table highlighting differences in completeness, clarity, and accuracy of documentation.**

**Prompt:-**

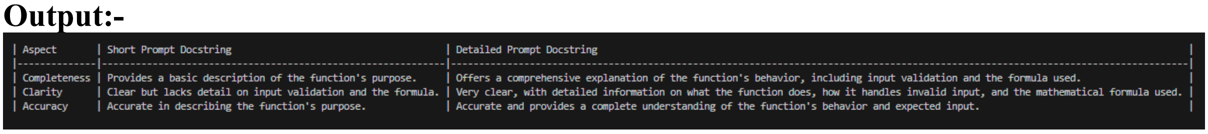
Generate two docstrings for the same Python function—one from a very short prompt and one from a detailed prompt—then compare them in a table for completeness, clarity, and accuracy, using natural, human-like language throughout.

**Code:-**



```python
# task-5
# Generate two docstrings for the same Python function-one from a very short prompt and one from a detailed prompt-then compare them in a table for completeness, clarity, and accuracy, using natural, human-like l
def calculate_area(radius):
    """Calculate the area of a circle given its radius."""
    import math
    if radius < 0:
        return "Radius cannot be negative."
    return math.pi * radius ** 2
# Short prompt docstring
"""Calculate the area of a circle given its radius.
"""
# Detailed prompt docstring
"""Calculate the area of a circle given its radius.
This function takes a single argument, radius, which represents the distance from the center of the circle to its edge.
The function checks if the radius is negative and returns an error message if it is.
If the radius is valid, it uses the formula for the area of a circle (A = πr²) to compute and return the area.
"""
# Comparison table
comparison_table = """
| Aspect      | Short Prompt Docstring                                   | Detailed Prompt Docstring                                                                                         |
|-------------|----------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------|
| Completeness | Provides a basic description of the function's purpose.  | Offers a comprehensive explanation of the function's behavior, including input validation and the formula used.  |
| Clarity     | Clear but lacks detail on input validation and the formula. | Very clear, with detailed information on what the function does, how it handles invalid input, and the mathematical formula used. |
| Accuracy    | Accurate in describing the function's purpose.           | Accurate and provides a complete understanding of the function's behavior and expected input.                     |
"""
print(comparison_table)
```

**Output:-**



```
| Aspect       | Short Prompt Docstring                                      | Detailed Prompt Docstring                                                                                                         |
|--------------|-------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------|
| Completeness | Provides a basic description of the function's purpose.     | Offers a comprehensive explanation of the function's behavior, including input validation and the formula used.                  |
| Clarity      | Clear but lacks detail on input validation and the formula. | Very clear, with detailed information on what the function does, how it handles invalid input, and the mathematical formula used. |
| Accuracy     | Accurate in describing the function's purpose.              | Accurate and provides a complete understanding of the function's behavior and expected input.                                    |
```

**Justification:-**

The detailed prompt docstring is more complete, providing not only the purpose of the function but also details about input validation and the mathematical formula used. This makes it clearer and more informative for users who may want to understand how the function works or modify it in the future. The short prompt docstring, while accurate, lacks this depth of information, which can lead to confusion or misuse of the function.