ASSIGNMENT-10.5

Hall Ticket:2303A51846

Task Description #1 – Variable Naming Issues

Task: Use AI to improve unclear variable names.

Sample Input Code:

def f(a, b):

return a + b

print(f(10, 20))

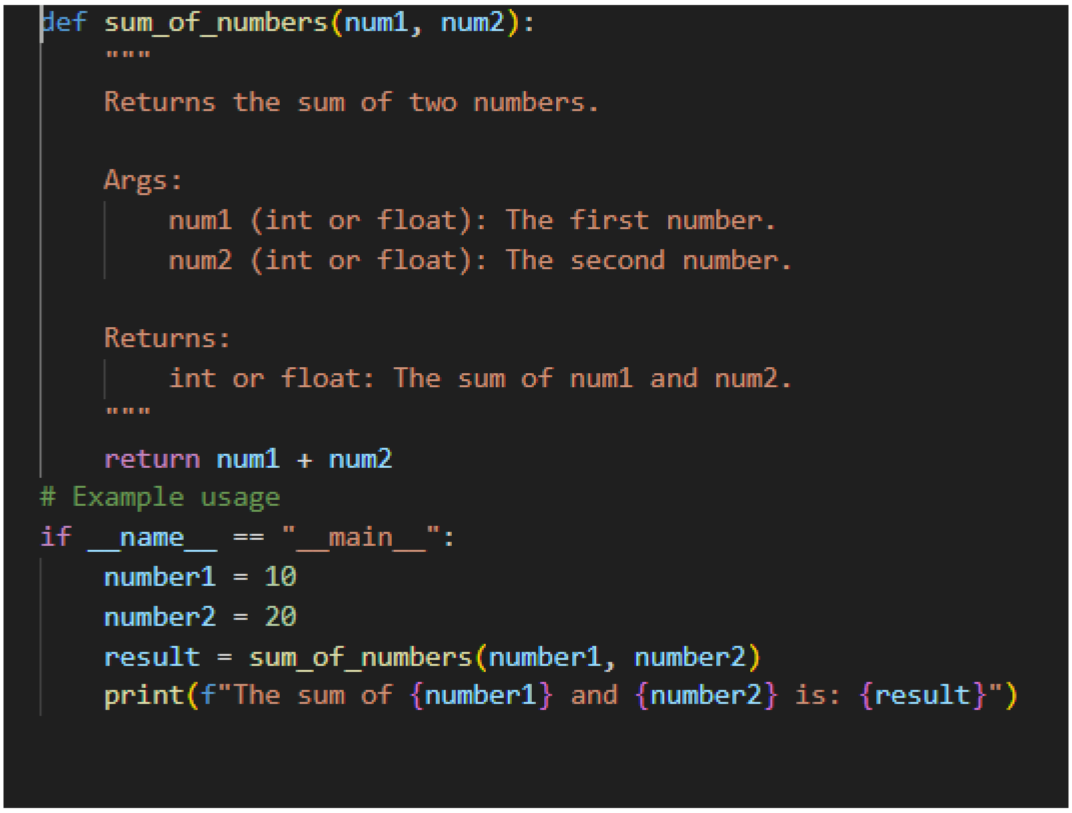
Expected Output:

• Code rewritten with meaningful function and variable names.

Prompt:

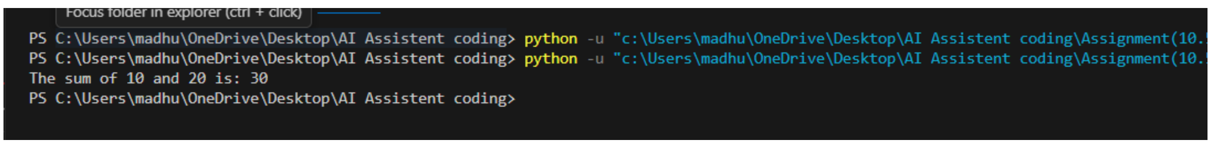
#code rewrite with meaningful function and variable names:

Code:



```python
def sum_of_numbers(num1, num2):
    """
    Returns the sum of two numbers.

    Args:
        num1 (int or float): The first number.
        num2 (int or float): The second number.

    Returns:
        int or float: The sum of num1 and num2.
    """
    return num1 + num2
# Example usage
if __name__ == "__main__":
    number1 = 10
    number2 = 20
    result = sum_of_numbers(number1, number2)
    print(f"The sum of {number1} and {number2} is: {result}")
```

Output:



```
PS C:\Users\madhu\OneDrive\Desktop\AI Assistent coding> python -u "c:\Users\madhu\OneDrive\Desktop\AI Assistent coding\Assignment(10.
PS C:\Users\madhu\OneDrive\Desktop\AI Assistent coding> python -u "c:\Users\madhu\OneDrive\Desktop\AI Assistent coding\Assignment(10.
The sum of 10 and 20 is: 30
PS C:\Users\madhu\OneDrive\Desktop\AI Assistent coding>
```

Observation:

1.The original code defines a function 'f' that takes two parameters 'a' and 'b', and returns their sum. It then calls the function with the arguments 10 and 20, and prints the result. The rewritten code provides a more descriptive function name 'sum_of_numbers' and uses meaningful variable names 'num1' and 'num2'. Additionally, it includes a docstring to explain the purpose of the function, its parameters, and its return value. An example usage is also provided to demonstrate how to use the function.

2.The rewritten code is more readable and maintainable, making it easier for other developers to understand the purpose of the function and how to use it.

3.The original code is concise but lacks clarity, while the rewritten code prioritizes clarity and maintainability over brevity.

4.In summary, the rewritten code enhances the readability and usability of the original code by using meaningful names, adding comments, and providing a clear docstring.

5.The original code is a simple function that adds two numbers, while the rewritten code provides a more comprehensive and user-friendly implementation of the same functionality.

Task Description #2 – Missing Error Handling

Task: Use AI to add proper error handling.

Sample Input Code:

def divide(a, b):

return a / b

print(divide(10, 0))

Expected Output:

• Code with exception handling and clear error messages

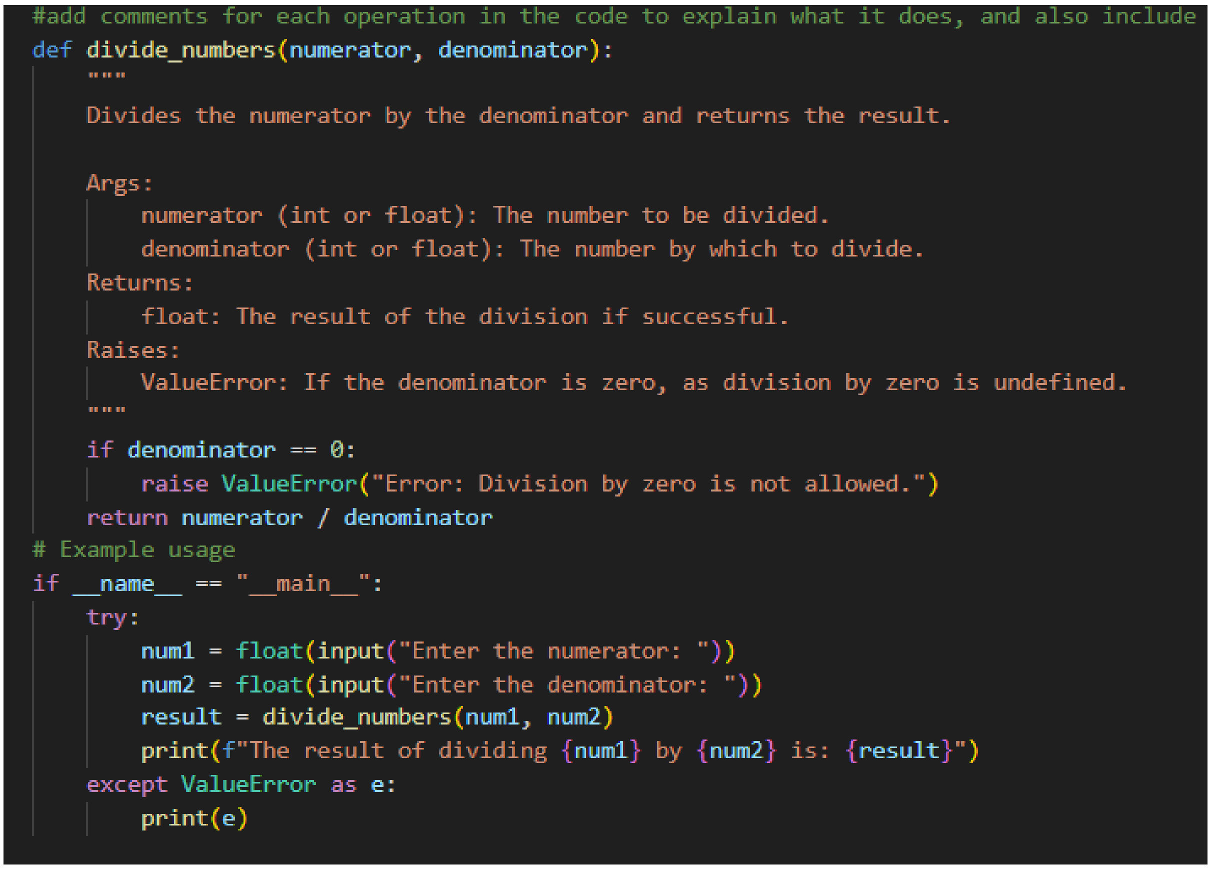
Prompt:

# code rewrite with error handling for division by zero, it should take input from the user and display the result or an error message:

#add in line comments to explain the code,explain the code in detail in the docstring, and use meaningful function and variable names:

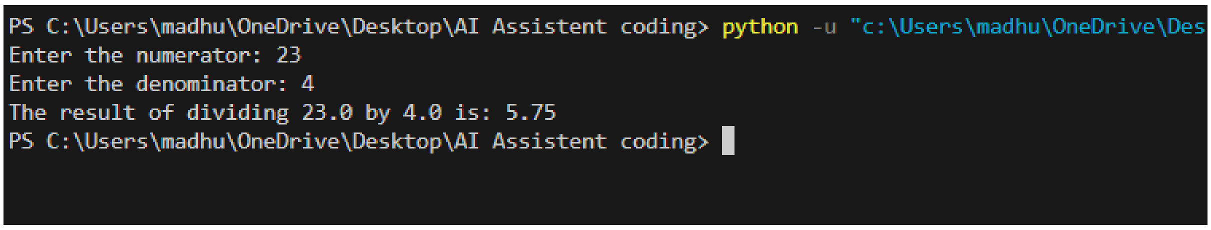
#add comments for each operation in the code to explain what it does, and also include an example usage of the function to demonstrate how it works**.**

Code:



```python
#add comments for each operation in the code to explain what it does, and also include
def divide_numbers(numerator, denominator):
    """
    Divides the numerator by the denominator and returns the result.

    Args:
        numerator (int or float): The number to be divided.
        denominator (int or float): The number by which to divide.
    Returns:
        float: The result of the division if successful.
    Raises:
        ValueError: If the denominator is zero, as division by zero is undefined.
    """
    if denominator == 0:
        raise ValueError("Error: Division by zero is not allowed.")
    return numerator / denominator
# Example usage
if __name__ == "__main__":
    try:
        num1 = float(input("Enter the numerator: "))
        num2 = float(input("Enter the denominator: "))
        result = divide_numbers(num1, num2)
        print(f"The result of dividing {num1} by {num2} is: {result}")
    except ValueError as e:
        print(e)
```

Output:



```
PS C:\Users\madhu\OneDrive\Desktop\AI Assistent coding> python -u "c:\Users\madhu\OneDrive\Des
Enter the numerator: 23
Enter the denominator: 4
The result of dividing 23.0 by 4.0 is: 5.75
PS C:\Users\madhu\OneDrive\Desktop\AI Assistent coding>
```

Observation:

1.The original code defines a function 'divide' that takes two parameters 'a' and 'b', and returns the result of dividing 'a' by 'b'. It then calls the function with the arguments 10 and 0, which would raise a division by zero error. The rewritten code provides a more descriptive function name 'divide_numbers' and uses meaningful variable names 'numerator' and 'denominator'. Additionally, it includes a docstring to explain the purpose of the function, its parameters, and its return value. The rewritten code also includes error handling to catch division by zero errors and provides an example usage to demonstrate how to use the function. This makes the code more robust and user-friendly compared to the original code.

2.The original code is concise but lacks error handling and clarity, while the rewritten code prioritizes clarity, maintainability, and robustness over brevity. In summary, the rewritten code enhances the readability and usability of the original code by using meaningful names, adding comments, providing a clear docstring, and implementing error handling for division by zero.

3.The original code is a simple function that performs division, while the rewritten code provides a more comprehensive and user-friendly implementation of the same functionality, including error handling and user input.

4.The rewritten code is more suitable for real-world applications where user input and error handling are important considerations, while the original code is more of a basic example of a division function without any safeguards or user interaction.

5.In conclusion, the rewritten code is a significant improvement over the original code in terms of readability, maintainability, and functionality, making it more suitable for practical use.

Task Description #3: Student Marks Processing System

The following program calculates total, average, and grade of a

student, but it has poor readability, style issues, and no error

handling.

```
marks=[78,85,90,66,88]
t=0
for i in marks:
t=t+i
a=t/len(marks)
if a>=90:
print("A")
elif a>=75:
print("B")
elif a>=60:
print("C")
else:
print("F")
```

Task:

• Use AI to refactor the code to follow PEP 8 standards.

• Add meaningful variable names, functions, and comments.

• Add basic input validation and documentation.

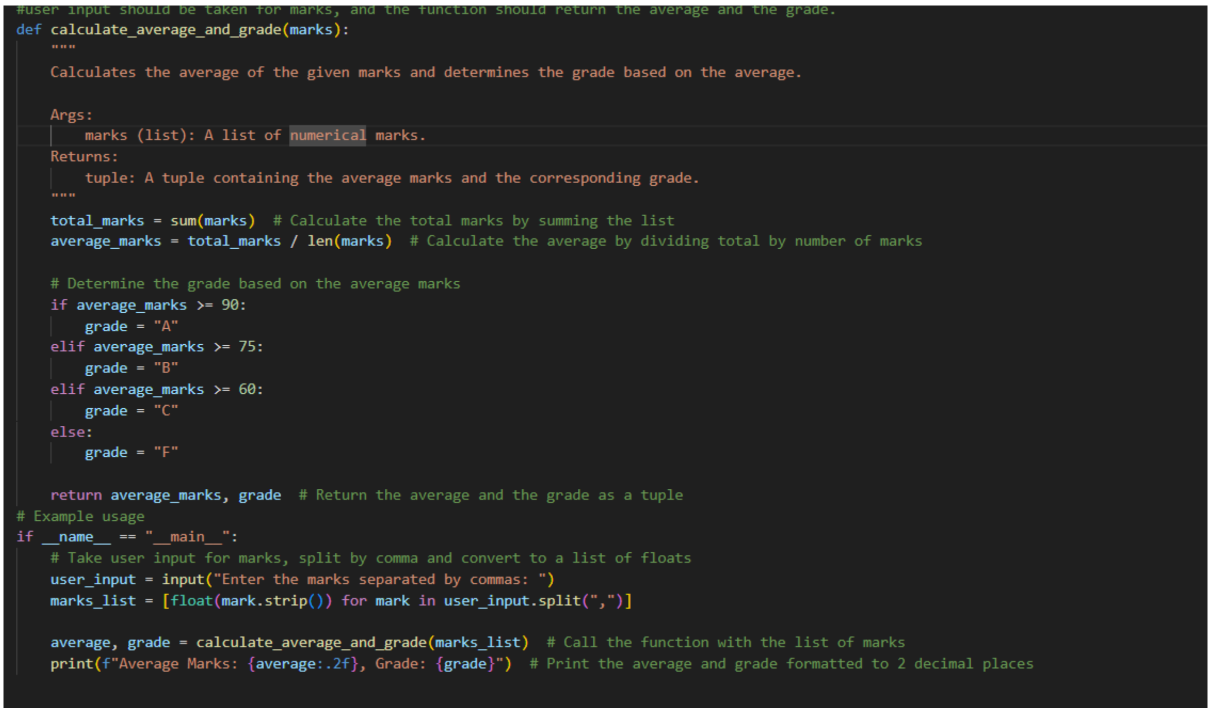
Prompt:

#rewrite the code with meaningful variable names, add comments to explain the code, and include a docstring to describe the function's purpose and parameters

#in code each line should be explained with comments, and the function should be defined to calculate the average marks and determine the grade based on the average.

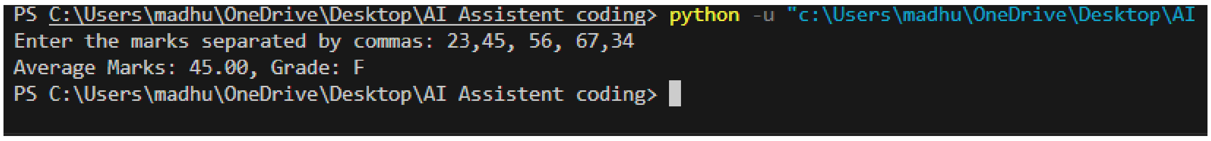
#user input should be taken for marks, and the function should return the average and the grade.

Code:



```python
#user input should be taken for marks, and the function should return the average and the grade.
def calculate_average_and_grade(marks):
    """
    Calculates the average of the given marks and determines the grade based on the average.

    Args:
        marks (list): A list of numerical marks.
    Returns:
        tuple: A tuple containing the average marks and the corresponding grade.
    """
    total_marks = sum(marks)  # Calculate the total marks by summing the list
    average_marks = total_marks / len(marks)  # Calculate the average by dividing total by number of marks

    # Determine the grade based on the average marks
    if average_marks >= 90:
        grade = "A"
    elif average_marks >= 75:
        grade = "B"
    elif average_marks >= 60:
        grade = "C"
    else:
        grade = "F"

    return average_marks, grade  # Return the average and the grade as a tuple
# Example usage
if __name__ == "__main__":
    # Take user input for marks, split by comma and convert to a list of floats
    user_input = input("Enter the marks separated by commas: ")
    marks_list = [float(mark.strip()) for mark in user_input.split(",")]

    average, grade = calculate_average_and_grade(marks_list)  # Call the function with the list of marks
    print(f"Average Marks: {average:.2f}, Grade: {grade}")  # Print the average and grade formatted to 2 decimal places
```

Output:



```
PS C:\Users\madhu\OneDrive\Desktop\AI Assistent coding> python -u "c:\Users\madhu\OneDrive\Desktop\AI
Enter the marks separated by commas: 23,45, 56, 67,34
Average Marks: 45.00, Grade: F
PS C:\Users\madhu\OneDrive\Desktop\AI Assistent coding> █
```

Observations:

1.The original code calculates the average of a list of marks and determines the grade based on the average. The rewritten code provides a more descriptive function name 'calculate_average_and_grade' and uses meaningful variable names such as 'total_marks' and 'average_marks'. Additionally, it includes a docstring to explain the purpose of the function, its parameters, and its return value. The rewritten code also takes user input for marks, processes it into a list of floats, and returns both the average and the grade as a tuple. This makes the code more robust and user-friendly compared to the original code.

2.The original code is concise but lacks clarity and user interaction, while the rewritten code prioritizes clarity, maintainability, and user interaction over brevity. In summary, the rewritten code enhances the readability and usability of the original code by using meaningful names, adding comments, providing a clear docstring, and implementing user input for marks. The original code is a simple implementation that calculates average and grade, while the rewritten code provides a more comprehensive and user-friendly implementation of the same functionality, including user input and formatted output.

3.In conclusion, the rewritten code is a significant improvement over the original code in terms of readability, maintainability, and functionality, making it more suitable for practical use. The original code is a basic implementation of average and grade calculation, while the rewritten code provides a more robust and user-friendly version that includes user input and clear documentation. The rewritten code is more suitable for real-world applications where user interaction and clarity are important considerations, while the original code is more of a basic example without any user interaction or clear documentation.

Task Description #4: Use AI to add docstrings and inline comments

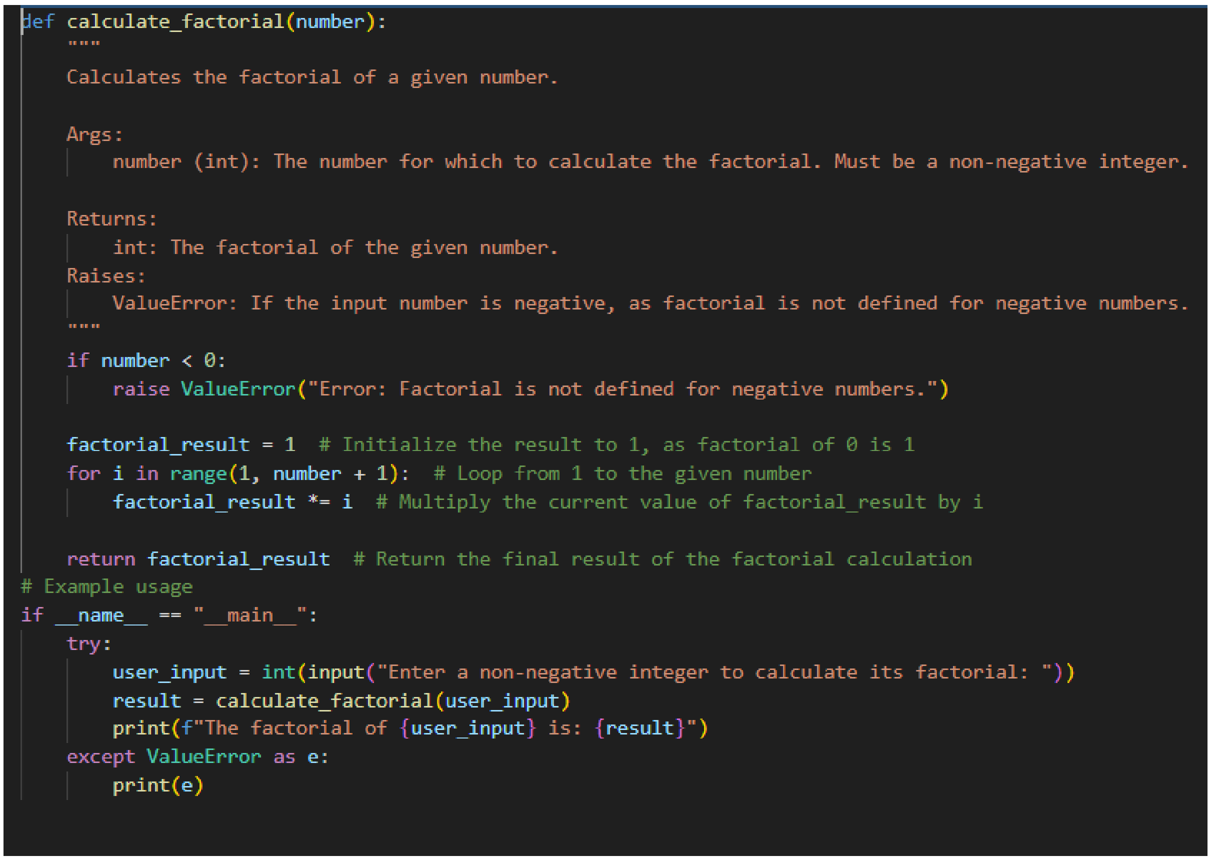
to the following function.

```
def factorial(n):

result = 1

for i in range(1,n+1):

result *= i

return result
```

Prompt:

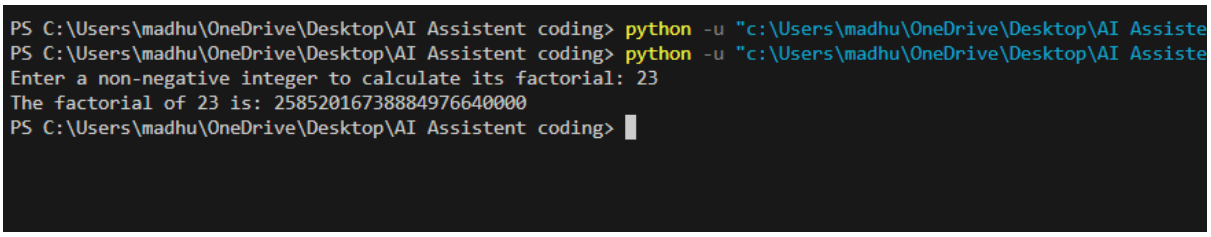
#rewrite the code with meaningful variable names, add comments to explain the code, and include a docstring to describe the function's purpose and parameters,and inline comments to explain each step of the code:

Code:



```python
def calculate_factorial(number):
    """
    Calculates the factorial of a given number.

    Args:
        number (int): The number for which to calculate the factorial. Must be a non-negative integer.

    Returns:
        int: The factorial of the given number.
    Raises:
        ValueError: If the input number is negative, as factorial is not defined for negative numbers.
    """
    if number < 0:
        raise ValueError("Error: Factorial is not defined for negative numbers.")

    factorial_result = 1  # Initialize the result to 1, as factorial of 0 is 1
    for i in range(1, number + 1):  # Loop from 1 to the given number
        factorial_result *= i  # Multiply the current value of factorial_result by i

    return factorial_result  # Return the final result of the factorial calculation
# Example usage
if __name__ == "__main__":
    try:
        user_input = int(input("Enter a non-negative integer to calculate its factorial: "))
        result = calculate_factorial(user_input)
        print(f"The factorial of {user_input} is: {result}")
    except ValueError as e:
        print(e)
```

Output:



```
PS C:\Users\madhu\OneDrive\Desktop\AI Assistent coding> python -u "c:\Users\madhu\OneDrive\Desktop\AI Assiste
PS C:\Users\madhu\OneDrive\Desktop\AI Assistent coding> python -u "c:\Users\madhu\OneDrive\Desktop\AI Assiste
Enter a non-negative integer to calculate its factorial: 23
The factorial of 23 is: 25852016738884976640000
PS C:\Users\madhu\OneDrive\Desktop\AI Assistent coding> 
```

Observations:

1.The original code defines a function 'factorial' that takes a single parameter 'n' and calculates the factorial of that number using a for loop. The rewritten code provides a more descriptive function name 'calculate_factorial' and uses meaningful variable names such as 'number' and 'factorial_result'. Additionally, it includes a docstring to explain the purpose of the function, its parameters, and its return value. The rewritten code also includes error handling to catch negative input values, which are not valid for factorial calculations. An example usage is provided to demonstrate how to use the function. This makes the code more robust and user-friendly compared to the original code.

2.The original code is concise but lacks error handling and clarity, while the rewritten code prioritizes clarity, maintainability, and robustness over brevity. In summary, the rewritten code enhances the readability and usability of the original code by using meaningful names, adding comments, providing a clear docstring, and implementing error handling for invalid input. The original code is a simple function that calculates factorial, while the rewritten code provides a more comprehensive and user-friendly implementation of the same functionality, including error handling and user input.

3.The original code is a basic implementation of a factorial function, while the rewritten code provides a more robust and user-friendly version that includes error handling and clear documentation.

4.The rewritten code is more suitable for real-world applications where user input and error handling are important considerations, while the original code is more of a basic example of a factorial function without any safeguards or user interaction.

5.In conclusion, the rewritten code is a significant improvement over the original code in terms of readability, maintainability, and functionality, making it more suitable for practical use.

Task Description #5: Password Validation System (Enhanced)

The following Python program validates a password using only a

minimum length check, which is insufficient for real-world

security requirements.

```
pwd = input("Enter password: ")

if len(pwd) >= 8:

print("Strong")

else:

print("Weak")
```

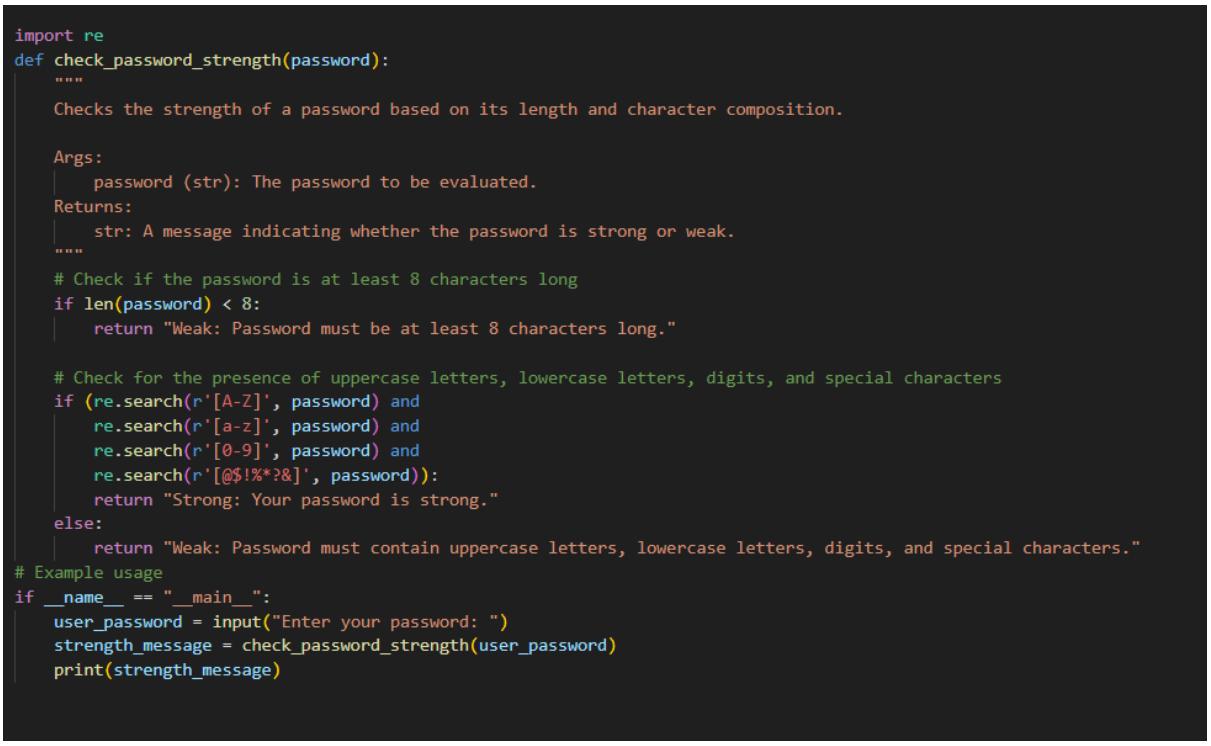
Prompt:

# rewrite the code with meaningful variable names, add comments to explain the code, and include a docstring to describe the function's purpose and parameters:

# This function checks the strength of a password based on its length. A password is considered strong if it has 8 or more characters, and weak otherwise.

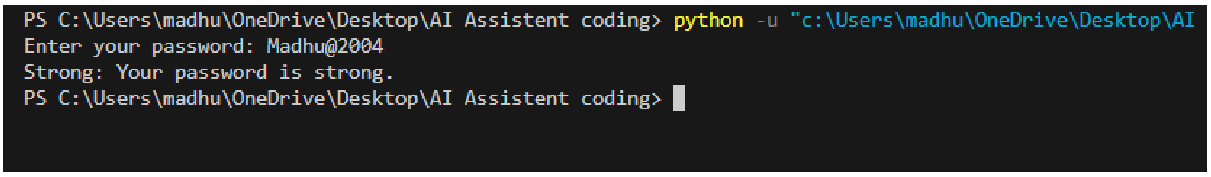
#password should contains atleast one uppercase letter, one lowercase letter, one digit and one special character and ensure security of the password by checking for common patterns or using a password strength library.

Code:



```python
import re
def check_password_strength(password):
    """
    Checks the strength of a password based on its length and character composition.

    Args:
        password (str): The password to be evaluated.
    Returns:
        str: A message indicating whether the password is strong or weak.
    """
    # Check if the password is at least 8 characters long
    if len(password) < 8:
        return "Weak: Password must be at least 8 characters long."

    # Check for the presence of uppercase letters, lowercase letters, digits, and special characters
    if (re.search(r'[A-Z]', password) and
        re.search(r'[a-z]', password) and
        re.search(r'[0-9]', password) and
        re.search(r'[@$!%*?&]', password)):
        return "Strong: Your password is strong."
    else:
        return "Weak: Password must contain uppercase letters, lowercase letters, digits, and special characters."
# Example usage
if __name__ == "__main__":
    user_password = input("Enter your password: ")
    strength_message = check_password_strength(user_password)
    print(strength_message)
```

Output:



```
PS C:\Users\madhu\OneDrive\Desktop\AI Assistent coding> python -u "c:\Users\madhu\OneDrive\Desktop\AI
Enter your password: Madhu@2004
Strong: Your password is strong.
PS C:\Users\madhu\OneDrive\Desktop\AI Assistent coding> 
```

Observation:

# The original code checks the strength of a password based solely on its length, categorizing it as strong if it has 8 or more characters and weak otherwise. The rewritten code provides a more comprehensive evaluation of password strength by checking not only the length but also the presence of uppercase letters, lowercase letters, digits, and special characters. The function 'check_password_strength' takes a password as input and returns a message indicating whether the password is strong or weak, along with specific reasons for weakness if applicable. This makes the rewritten code more robust and effective in assessing password strength compared to the original code.

# The original code is concise but lacks depth in evaluating password strength, while the rewritten code prioritizes security and thoroughness over brevity. In summary, the rewritten code enhances the functionality of the original code by providing a more detailed assessment of password strength, making it more suitable for real-world applications where password security is a concern. The original code is a basic implementation that checks only the length of the password, while the rewritten code provides a more comprehensive and user-friendly implementation that includes checks for character composition and specific feedback on password weaknesses.

# In conclusion, the rewritten code is a significant improvement over the original code in terms of functionality and security, making it more suitable for practical use in scenarios where password strength is important. The original code is a simple length check, while the

rewritten code offers a more robust and user-friendly approach to evaluating password strength.