ASSIGNMENT – 10.5

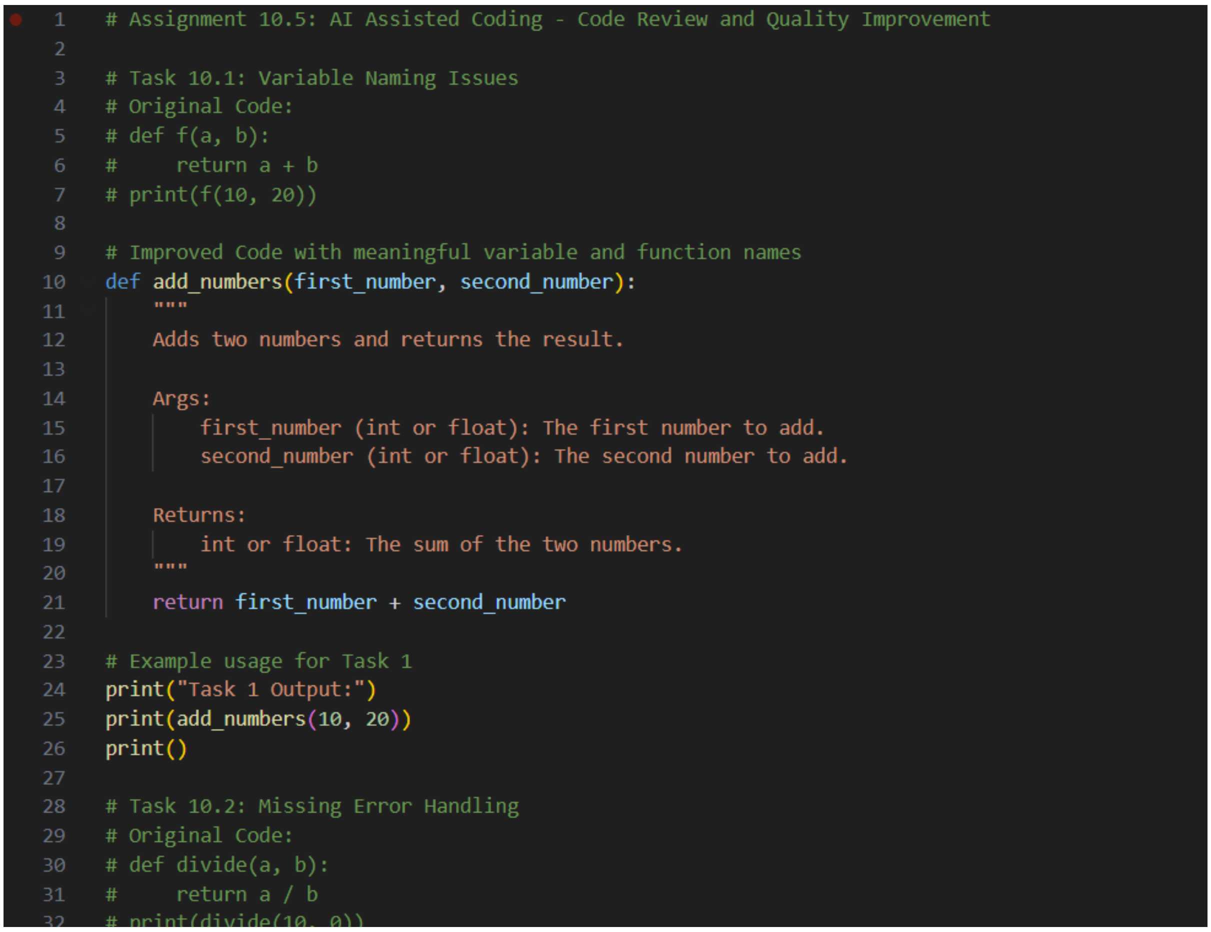
Name: B.Tejaswi

Roll Number: 2303A51184

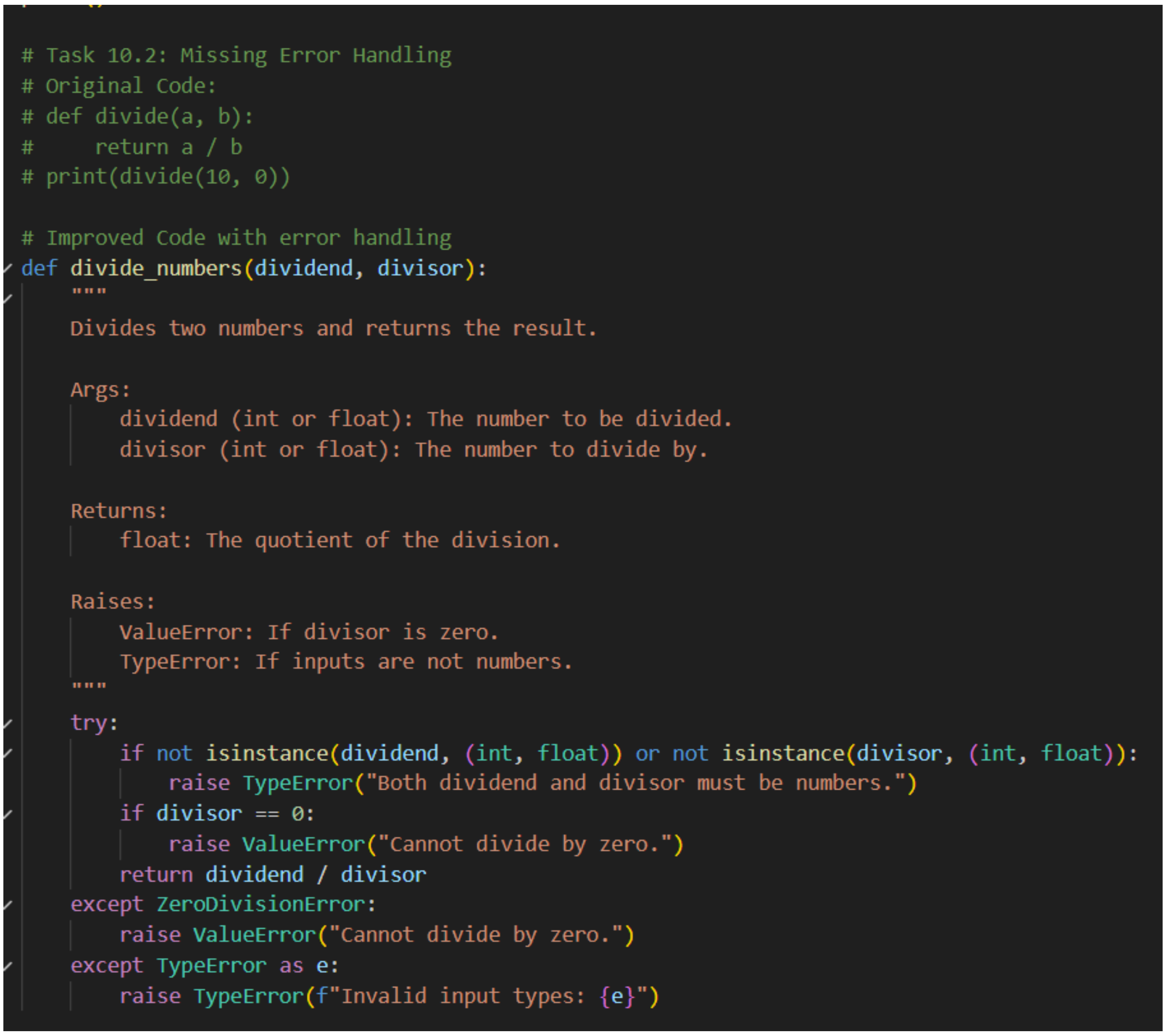
Batch - 03

AI Assisted Coding

Task 1:-

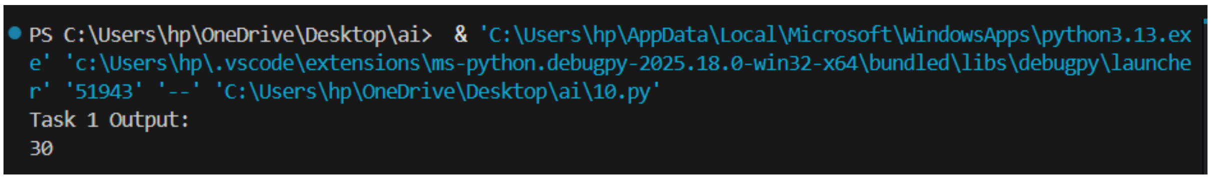


```python
# Assignment 10.5: AI Assisted Coding - Code Review and Quality Improvement

# Task 10.1: Variable Naming Issues
# Original Code:
# def f(a, b):
#     return a + b
# print(f(10, 20))

# Improved Code with meaningful variable and function names
def add_numbers(first_number, second_number):
    """
    Adds two numbers and returns the result.

    Args:
        first_number (int or float): The first number to add.
        second_number (int or float): The second number to add.

    Returns:
        int or float: The sum of the two numbers.
    """
    return first_number + second_number

# Example usage for Task 1
print("Task 1 Output:")
print(add_numbers(10, 20))
print()

# Task 10.2: Missing Error Handling
# Original Code:
# def divide(a, b):
#     return a / b
# print(divide(10, 0))
```



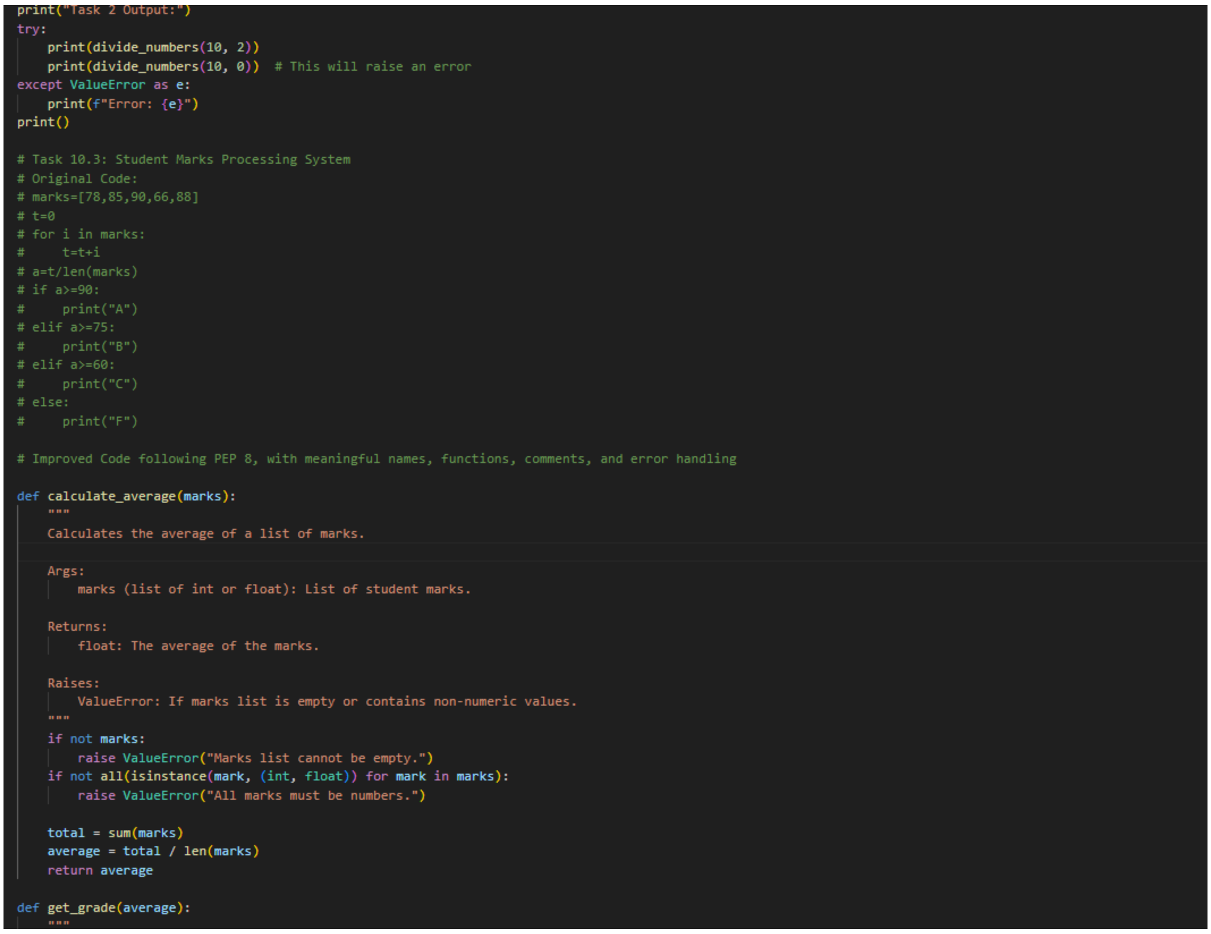
```python
# Task 10.2: Missing Error Handling
# Original Code:
# def divide(a, b):
#     return a / b
# print(divide(10, 0))

# Improved Code with error handling
def divide_numbers(dividend, divisor):
    """
    Divides two numbers and returns the result.

    Args:
        dividend (int or float): The number to be divided.
        divisor (int or float): The number to divide by.

    Returns:
        float: The quotient of the division.

    Raises:
        ValueError: If divisor is zero.
        TypeError: If inputs are not numbers.
    """
    try:
        if not isinstance(dividend, (int, float)) or not isinstance(divisor, (int, float)):
            raise TypeError("Both dividend and divisor must be numbers.")
        if divisor == 0:
            raise ValueError("Cannot divide by zero.")
        return dividend / divisor
    except ZeroDivisionError:
        raise ValueError("Cannot divide by zero.")
    except TypeError as e:
        raise TypeError(f"Invalid input types: {e}")
```

Output:-

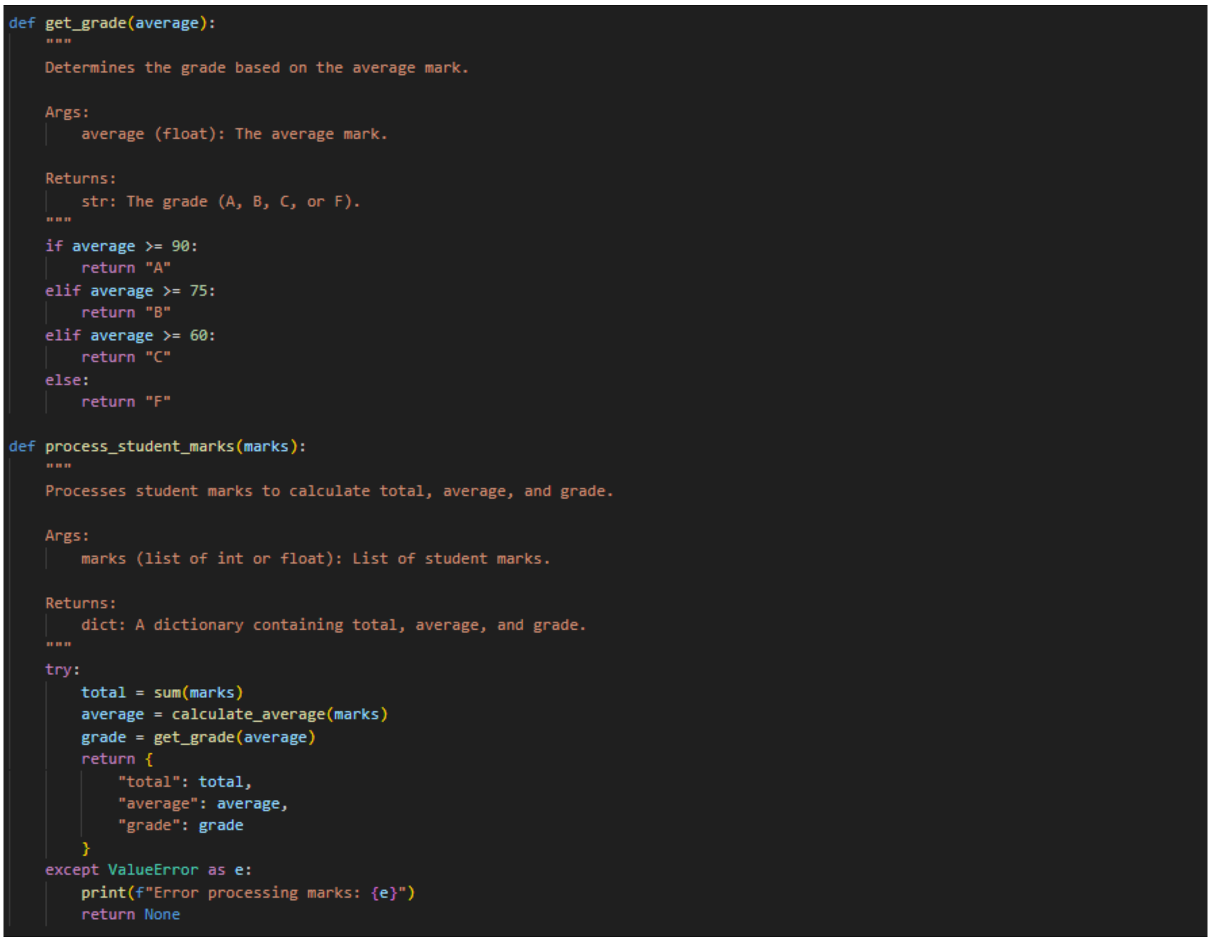


```
PS C:\Users\hp\OneDrive\Desktop\ai>  & 'C:\Users\hp\AppData\Local\Microsoft\WindowsApps\python3.13.ex
e' 'c:\Users\hp\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launche
r' '51943' '--' 'C:\Users\hp\OneDrive\Desktop\ai\10.py'
Task 1 Output:
30
```

Task 2:

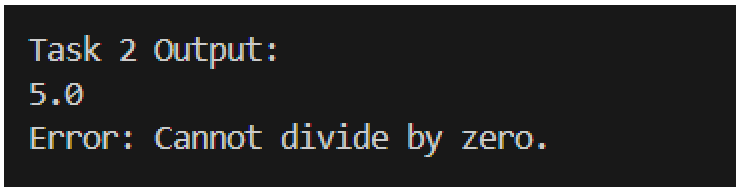


```python
print("Task 2 Output:")
try:
    print(divide_numbers(10, 2))
    print(divide_numbers(10, 0))  # This will raise an error
except ValueError as e:
    print(f"Error: {e}")
print()

# Task 10.3: Student Marks Processing System
# Original Code:
# 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")

# Improved Code following PEP 8, with meaningful names, functions, comments, and error handling

def calculate_average(marks):
    """
    Calculates the average of a list of marks.

    Args:
        marks (list of int or float): List of student marks.

    Returns:
        float: The average of the marks.

    Raises:
        ValueError: If marks list is empty or contains non-numeric values.
    """
    if not marks:
        raise ValueError("Marks list cannot be empty.")
    if not all(isinstance(mark, (int, float)) for mark in marks):
        raise ValueError("All marks must be numbers.")

    total = sum(marks)
    average = total / len(marks)
    return average

def get_grade(average):
    """
```



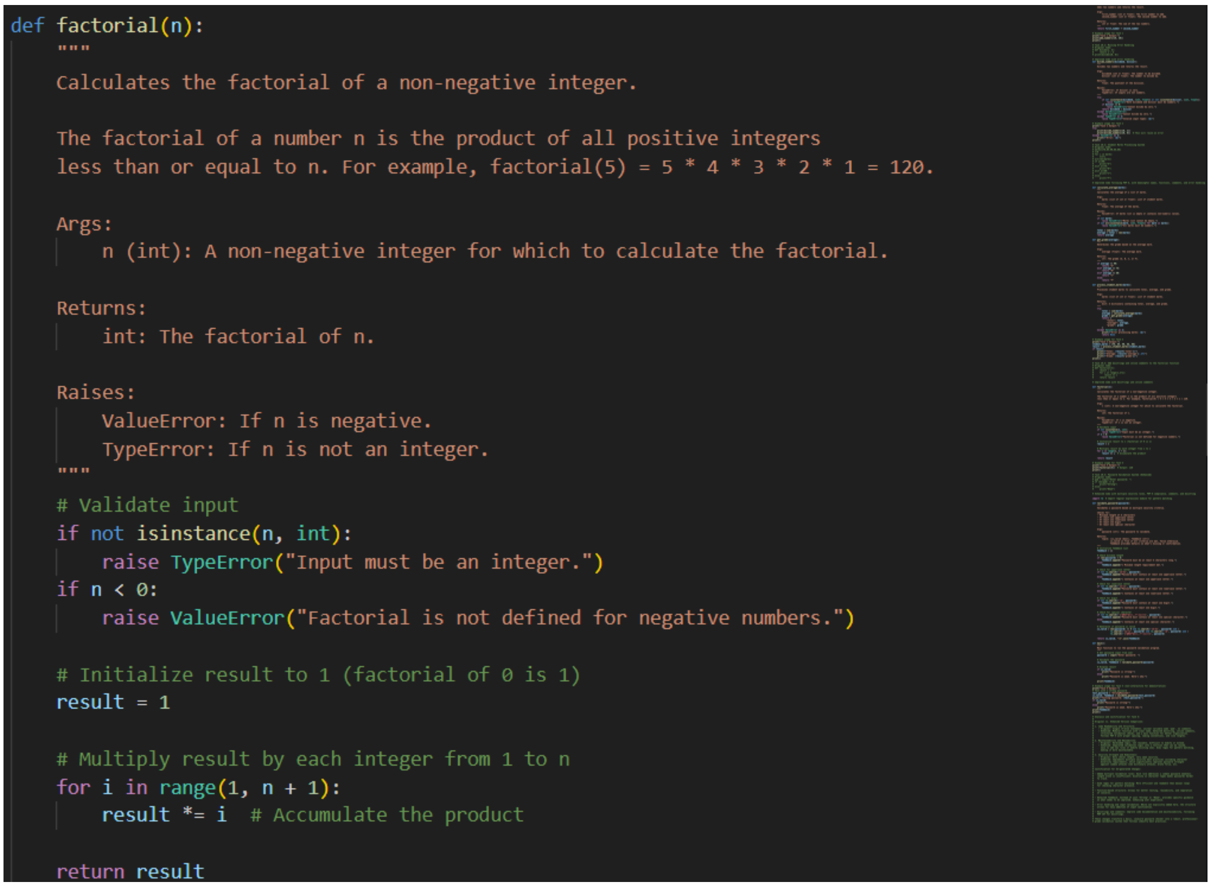
```python
def get_grade(average):
    """
    Determines the grade based on the average mark.

    Args:
        average (float): The average mark.

    Returns:
        str: The grade (A, B, C, or F).
    """
    if average >= 90:
        return "A"
    elif average >= 75:
        return "B"
    elif average >= 60:
        return "C"
    else:
        return "F"

def process_student_marks(marks):
    """
    Processes student marks to calculate total, average, and grade.

    Args:
        marks (list of int or float): List of student marks.

    Returns:
        dict: A dictionary containing total, average, and grade.
    """
    try:
        total = sum(marks)
        average = calculate_average(marks)
        grade = get_grade(average)
        return {
            "total": total,
            "average": average,
            "grade": grade
        }
    except ValueError as e:
        print(f"Error processing marks: {e}")
        return None
```

Output:-



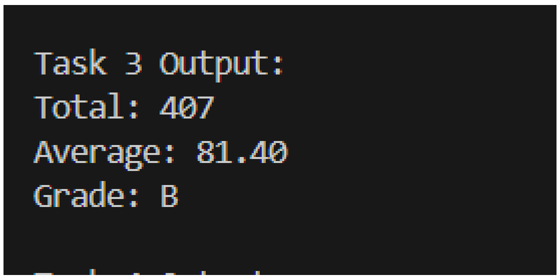
```
Task 2 Output:
5.0
Error: Cannot divide by zero.
```

Task 3:-



```python
def factorial(n):
    """
    Calculates the factorial of a non-negative integer.

    The factorial of a number n is the product of all positive integers
    less than or equal to n. For example, factorial(5) = 5 * 4 * 3 * 2 * 1 = 120.

    Args:
        n (int): A non-negative integer for which to calculate the factorial.

    Returns:
        int: The factorial of n.

    Raises:
        ValueError: If n is negative.
        TypeError: If n is not an integer.
    """
    # Validate input
    if not isinstance(n, int):
        raise TypeError("Input must be an integer.")
    if n < 0:
        raise ValueError("Factorial is not defined for negative numbers.")

    # Initialize result to 1 (factorial of 0 is 1)
    result = 1

    # Multiply result by each integer from 1 to n
    for i in range(1, n + 1):
        result *= i  # Accumulate the product

    return result
```

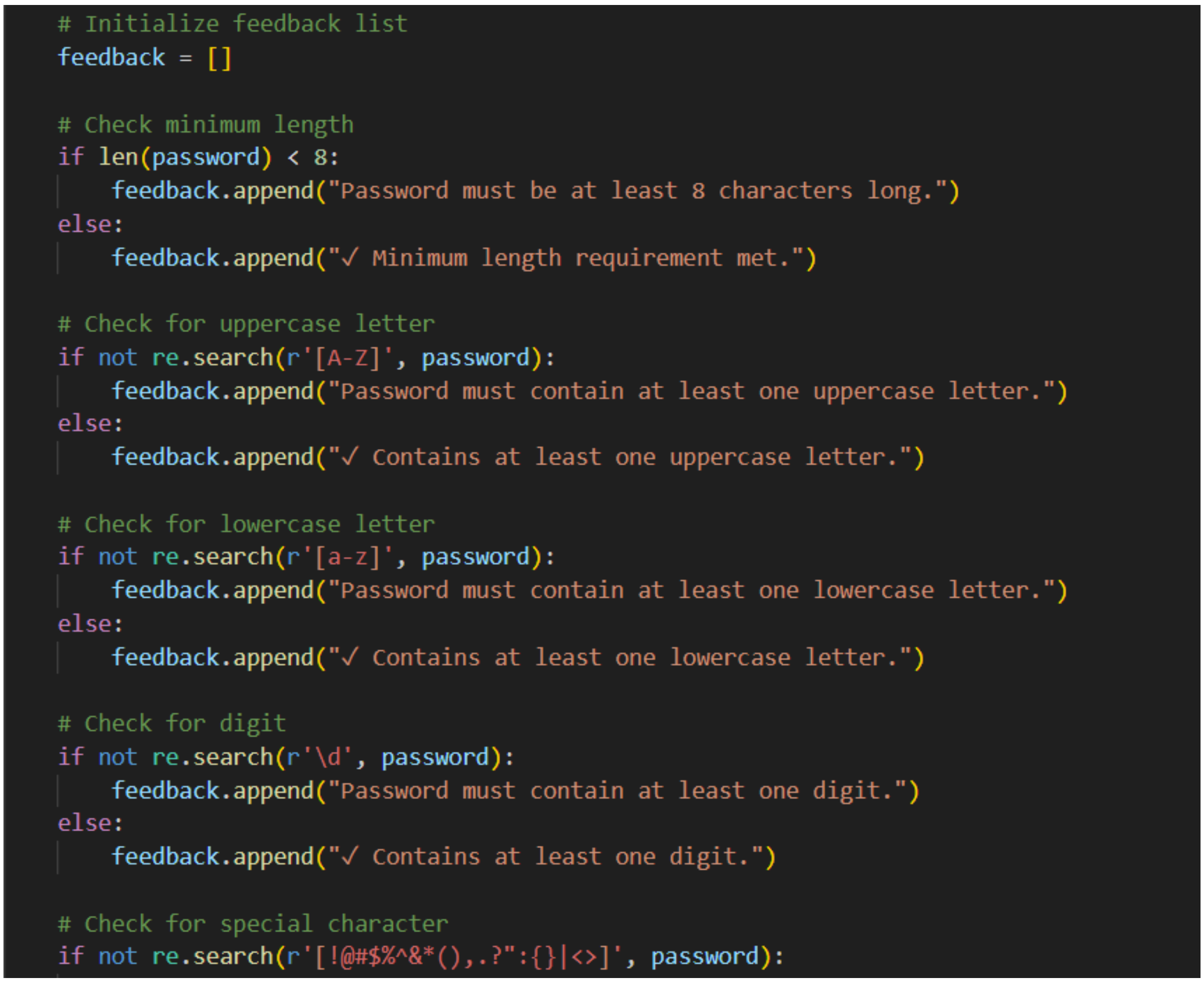
Output:



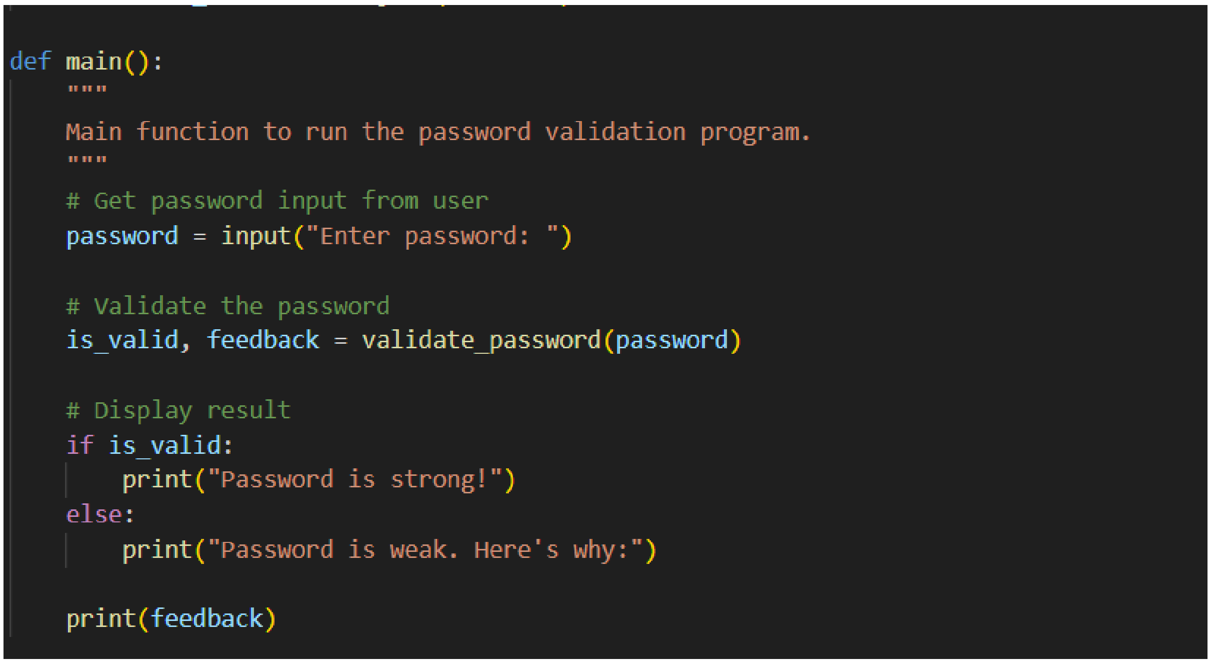
```
Task 3 Output:
Total: 407
Average: 81.40
Grade: B
```

Task 4:-

Input:-

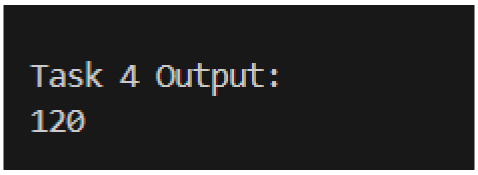


```python
    # Initialize feedback list
    feedback = []

    # Check minimum length
    if len(password) < 8:
        feedback.append("Password must be at least 8 characters long.")
    else:
        feedback.append("✓ Minimum length requirement met.")

    # Check for uppercase letter
    if not re.search(r'[A-Z]', password):
        feedback.append("Password must contain at least one uppercase letter.")
    else:
        feedback.append("✓ Contains at least one uppercase letter.")

    # Check for lowercase letter
    if not re.search(r'[a-z]', password):
        feedback.append("Password must contain at least one lowercase letter.")
    else:
        feedback.append("✓ Contains at least one lowercase letter.")

    # Check for digit
    if not re.search(r'\d', password):
        feedback.append("Password must contain at least one digit.")
    else:
        feedback.append("✓ Contains at least one digit.")

    # Check for special character
    if not re.search(r'[!@#$%^&*(),.?":{}|<>]', password):
```



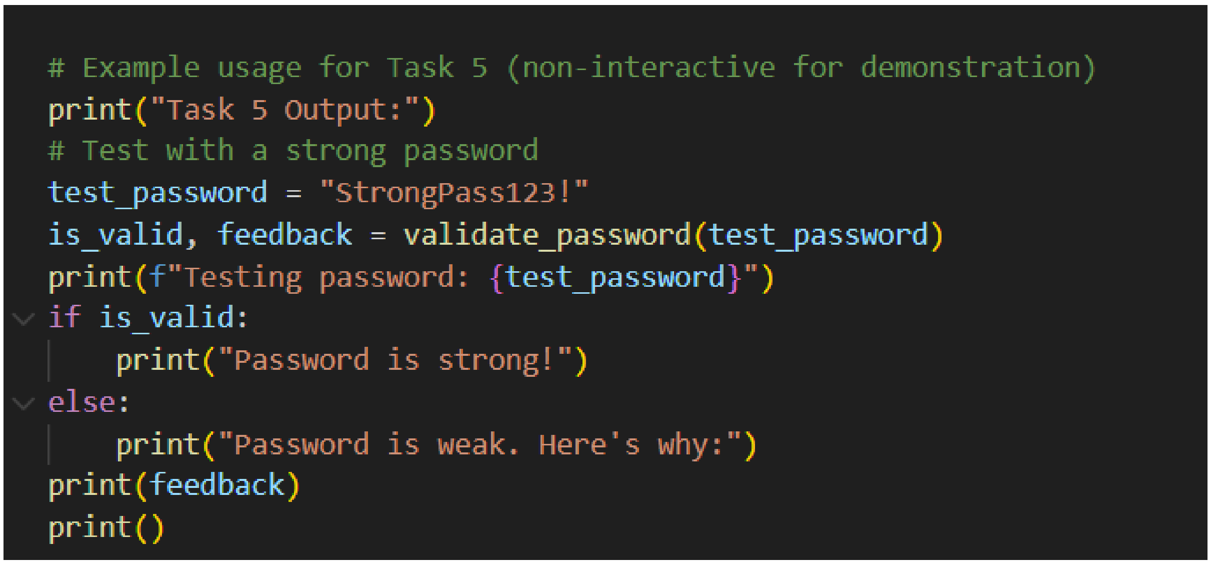
```python
def main():
    """
    Main function to run the password validation program.
    """
    # Get password input from user
    password = input("Enter password: ")

    # Validate the password
    is_valid, feedback = validate_password(password)

    # Display result
    if is_valid:
        print("Password is strong!")
    else:
        print("Password is weak. Here's why:")

    print(feedback)
```

Output:-



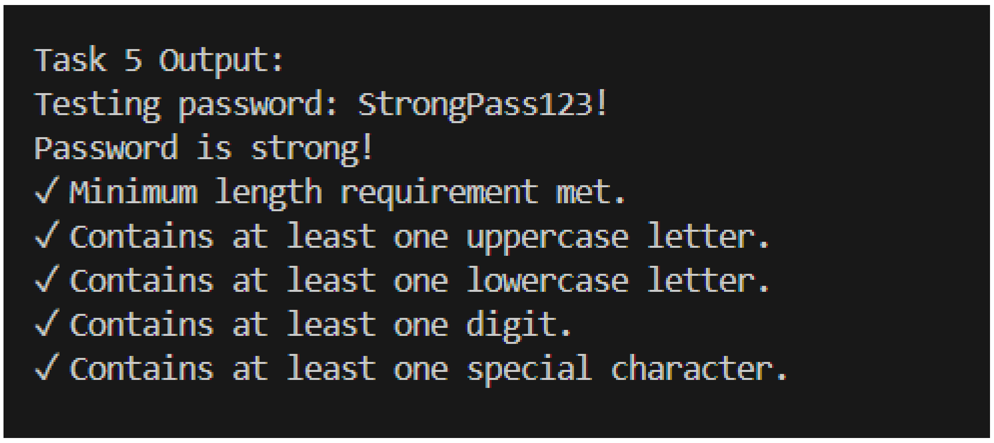
```
Task 4 Output:
120
```

Task 5:



```python
# Example usage for Task 5 (non-interactive for demonstration)
print("Task 5 Output:")
# Test with a strong password
test_password = "StrongPass123!"
is_valid, feedback = validate_password(test_password)
print(f"Testing password: {test_password}")
if is_valid:
    print("Password is strong!")
else:
    print("Password is weak. Here's why:")
print(feedback)
print()
```

Output:-



```
Task 5 Output:
Testing password: StrongPass123!
Password is strong!
✓Minimum length requirement met.
✓Contains at least one uppercase letter.
✓Contains at least one lowercase letter.
✓Contains at least one digit.
✓Contains at least one special character.
```