SR UNIVERSITY

School of Computer Science and Artificial Intelligence

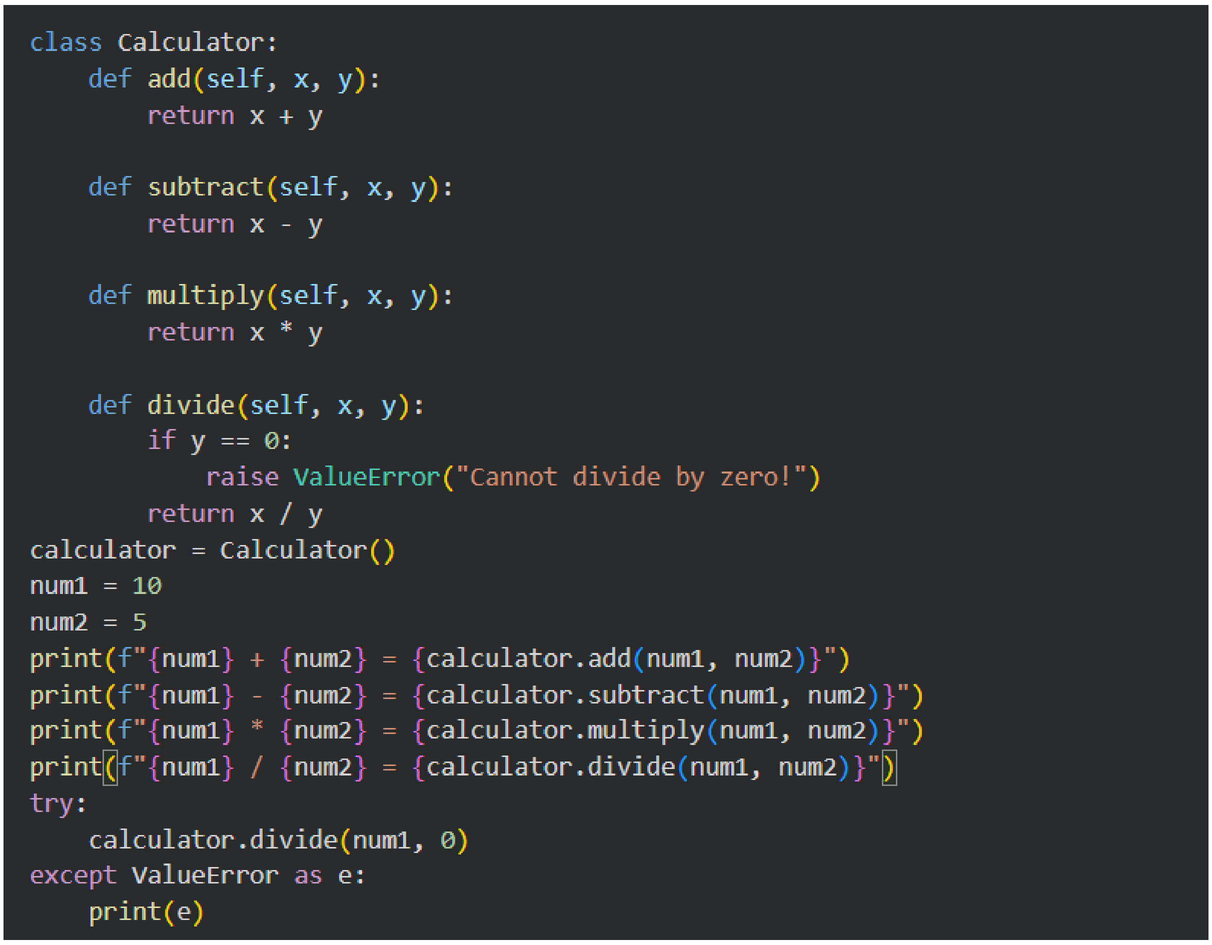
Lab Assignment 3.2

Program	**: B. Tech (CSE)**
Course Title	**: AI Assisted Coding**
Course Code	**: 23CS002PC304**
Semester	**: III**
Academic Session	**: 2025-2026**
Name of Student	**: Sruthi**
Enrollment No.	**: 2403A51L10**
Batch No.	**: 51**
Date	**: 13-01-2026**

Task 1

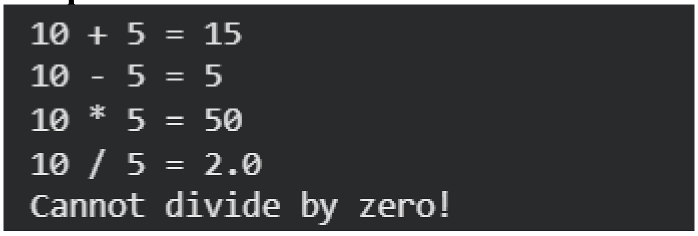
Prompt 1: Design a calculator program.

Code:



```python
class Calculator:
    def add(self, x, y):
        return x + y

    def subtract(self, x, y):
        return x - y

    def multiply(self, x, y):
        return x * y

    def divide(self, x, y):
        if y == 0:
            raise ValueError("Cannot divide by zero!")
        return x / y
calculator = Calculator()
num1 = 10
num2 = 5
print(f"{num1} + {num2} = {calculator.add(num1, num2)}")
print(f"{num1} - {num2} = {calculator.subtract(num1, num2)}")
print(f"{num1} * {num2} = {calculator.multiply(num1, num2)}")
print(f"{num1} / {num2} = {calculator.divide(num1, num2)}")
try:
    calculator.divide(num1, 0)
except ValueError as e:
    print(e)
```

Output:



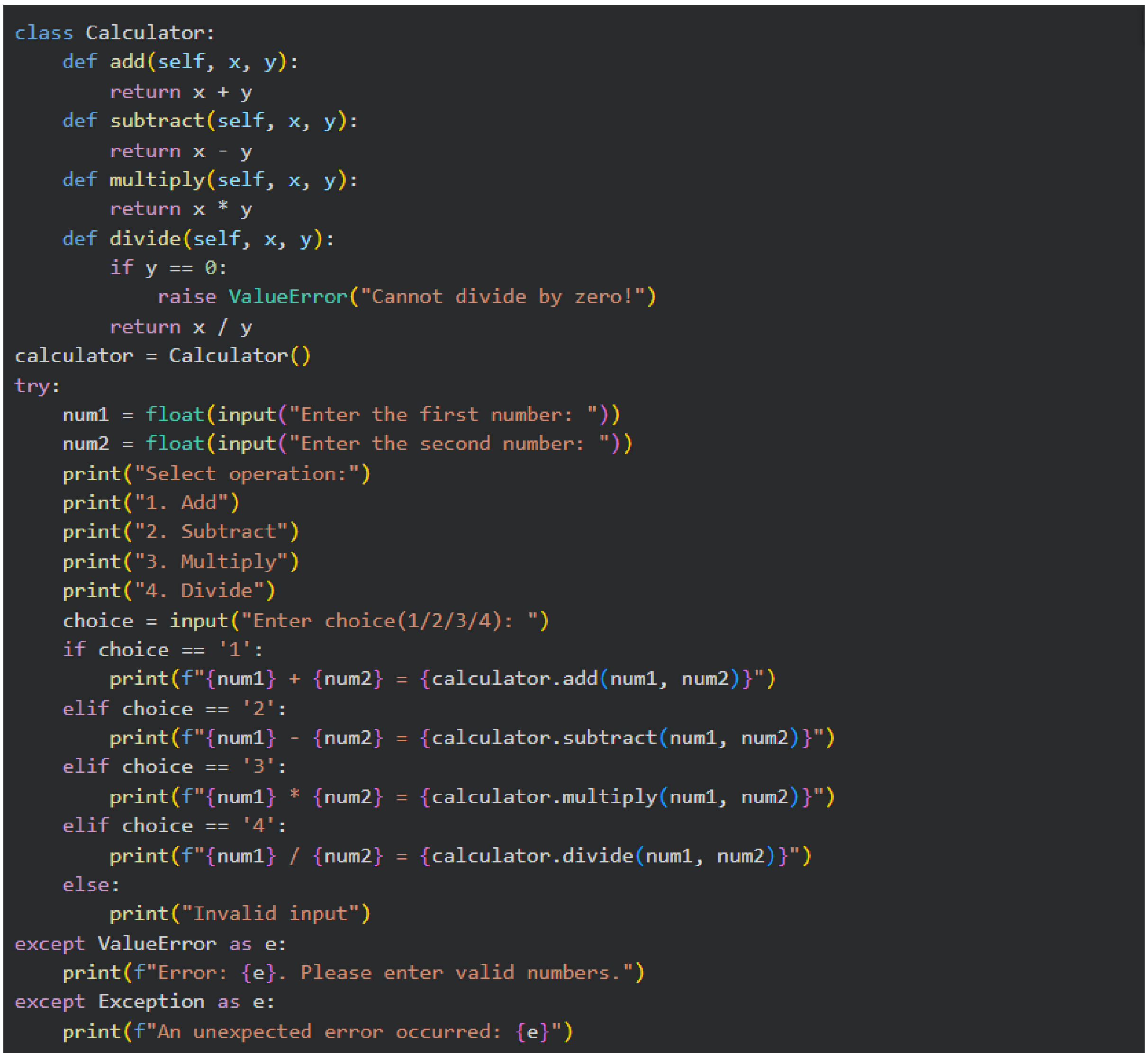
```
10 + 5 = 15
10 - 5 = 5
10 * 5 = 50
10 / 5 = 2.0
Cannot divide by zero!
```

Explanation:

Even though the initial prompt was vague, the AI assumed default calculator requirements and generated a complete object-oriented implementation. This highlights that some AI models infer common patterns even with minimal instruction.

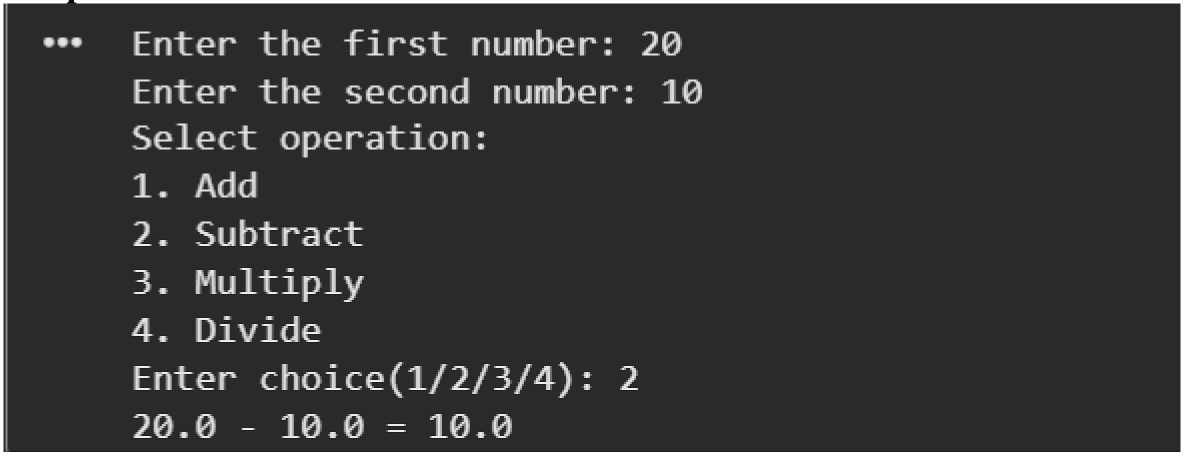
Prompt 2: Add user input from the command line so the user can enter two numbers and choose an operation to perform.

Code:



```python
class Calculator:
    def add(self, x, y):
        return x + y
    def subtract(self, x, y):
        return x - y
    def multiply(self, x, y):
        return x * y
    def divide(self, x, y):
        if y == 0:
            raise ValueError("Cannot divide by zero!")
        return x / y
calculator = Calculator()
try:
    num1 = float(input("Enter the first number: "))
    num2 = float(input("Enter the second number: "))
    print("Select operation:")
    print("1. Add")
    print("2. Subtract")
    print("3. Multiply")
    print("4. Divide")
    choice = input("Enter choice(1/2/3/4): ")
    if choice == '1':
        print(f"{num1} + {num2} = {calculator.add(num1, num2)}")
    elif choice == '2':
        print(f"{num1} - {num2} = {calculator.subtract(num1, num2)}")
    elif choice == '3':
        print(f"{num1} * {num2} = {calculator.multiply(num1, num2)}")
    elif choice == '4':
        print(f"{num1} / {num2} = {calculator.divide(num1, num2)}")
    else:
        print("Invalid input")
except ValueError as e:
    print(f"Error: {e}. Please enter valid numbers.")
except Exception as e:
    print(f"An unexpected error occurred: {e}")
```

Output:



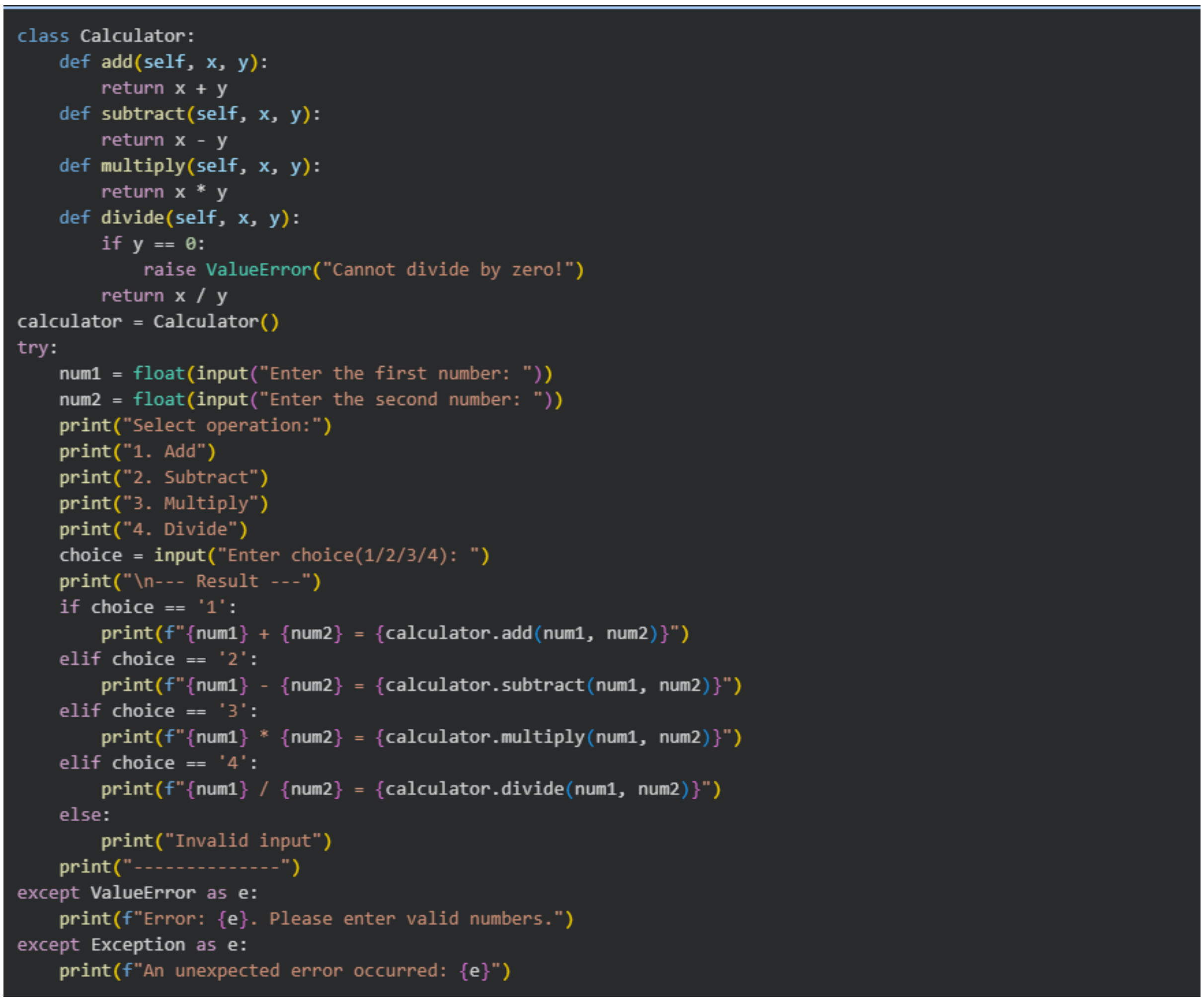
```
Enter the first number: 20
Enter the second number: 10
Select operation:
1. Add
2. Subtract
3. Multiply
4. Divide
Enter choice(1/2/3/4): 2
20.0 - 10.0 = 10.0
```

Explanation:

This program implements a class-based calculator using object-oriented programming. It supports basic arithmetic operations, handles user input through a menu-driven interface, and includes exception handling to manage invalid inputs and division by zero.

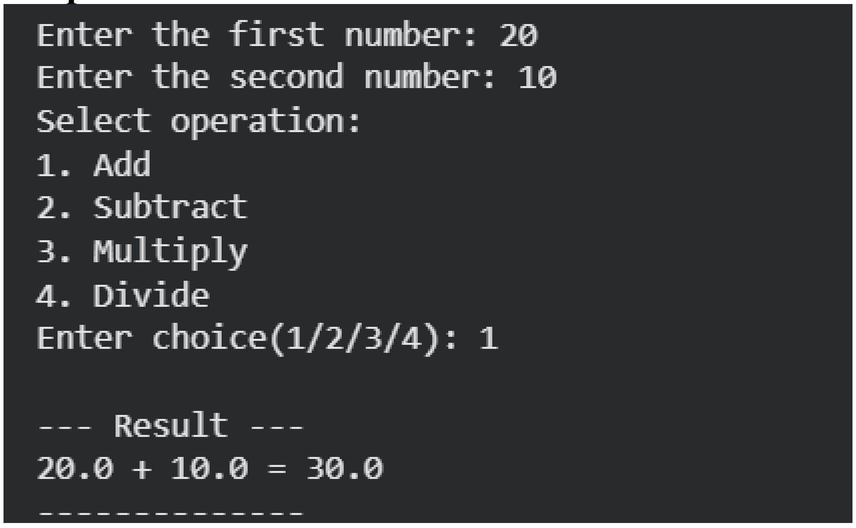
Prompt 3: Display results in a clean and user-friendly format

Code:



```python
class Calculator:
    def add(self, x, y):
        return x + y
    def subtract(self, x, y):
        return x - y
    def multiply(self, x, y):
        return x * y
    def divide(self, x, y):
        if y == 0:
            raise ValueError("Cannot divide by zero!")
        return x / y
calculator = Calculator()
try:
    num1 = float(input("Enter the first number: "))
    num2 = float(input("Enter the second number: "))
    print("Select operation:")
    print("1. Add")
    print("2. Subtract")
    print("3. Multiply")
    print("4. Divide")
    choice = input("Enter choice(1/2/3/4): ")
    print("\n--- Result ---")
    if choice == '1':
        print(f"{num1} + {num2} = {calculator.add(num1, num2)}")
    elif choice == '2':
        print(f"{num1} - {num2} = {calculator.subtract(num1, num2)}")
    elif choice == '3':
        print(f"{num1} * {num2} = {calculator.multiply(num1, num2)}")
    elif choice == '4':
        print(f"{num1} / {num2} = {calculator.divide(num1, num2)}")
    else:
        print("Invalid input")
    print("--------------")
except ValueError as e:
    print(f"Error: {e}. Please enter valid numbers.")
except Exception as e:
    print(f"An unexpected error occurred: {e}")
```

Output:



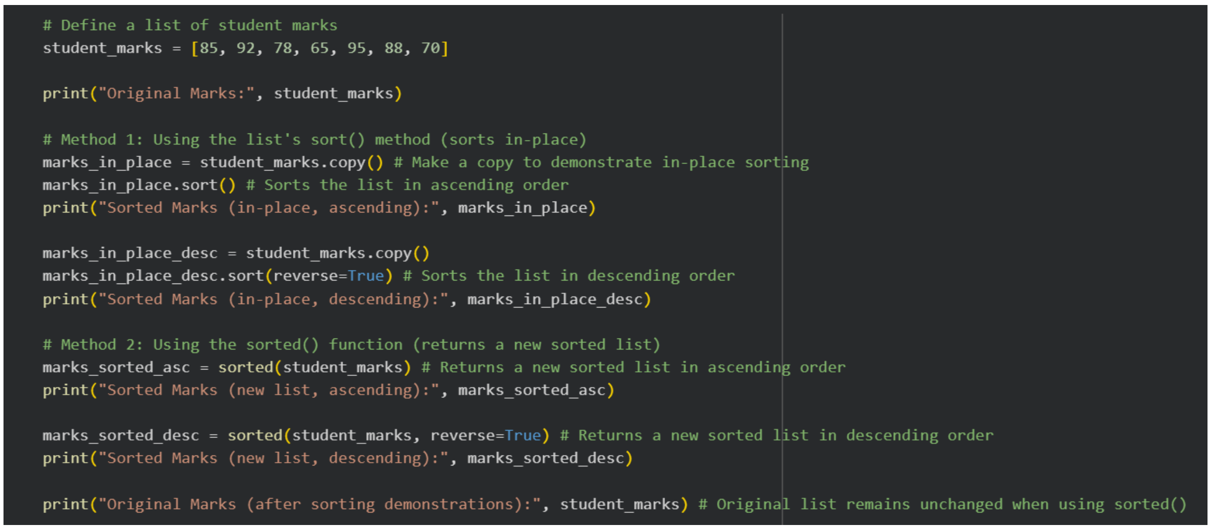
```
Enter the first number: 20
Enter the second number: 10
Select operation:
1. Add
2. Subtract
3. Multiply
4. Divide
Enter choice(1/2/3/4): 1

--- Result ---
20.0 + 10.0 = 30.0
--------------
```

Explanation:
The updated program improves output formatting and user interaction while preserving the original object-oriented structure and error-handling logic. These changes enhance readability, usability, and overall program clarity without affecting core functionality.

Task 2:

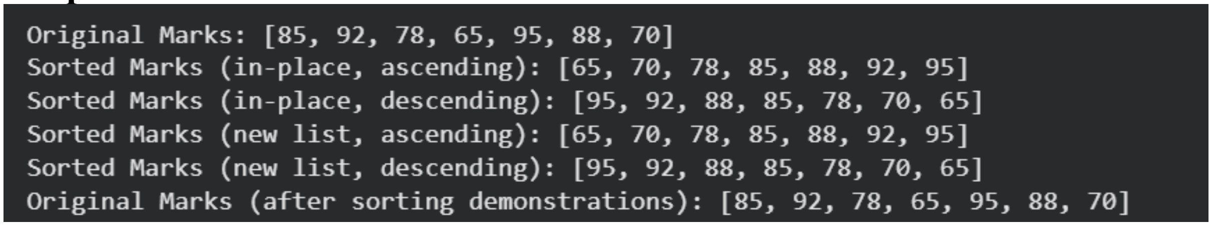
Prompt: Write a program to sort student marks.

Code:



```python
# Define a list of student marks
student_marks = [85, 92, 78, 65, 95, 88, 70]

print("Original Marks:", student_marks)

# Method 1: Using the list's sort() method (sorts in-place)
marks_in_place = student_marks.copy() # Make a copy to demonstrate in-place sorting
marks_in_place.sort() # Sorts the list in ascending order
print("Sorted Marks (in-place, ascending):", marks_in_place)

marks_in_place_desc = student_marks.copy()
marks_in_place_desc.sort(reverse=True) # Sorts the list in descending order
print("Sorted Marks (in-place, descending):", marks_in_place_desc)

# Method 2: Using the sorted() function (returns a new sorted list)
marks_sorted_asc = sorted(student_marks) # Returns a new sorted list in ascending order
print("Sorted Marks (new list, ascending):", marks_sorted_asc)

marks_sorted_desc = sorted(student_marks, reverse=True) # Returns a new sorted list in descending order
print("Sorted Marks (new list, descending):", marks_sorted_desc)

print("Original Marks (after sorting demonstrations):", student_marks) # Original list remains unchanged when using sorted()
```

Output:

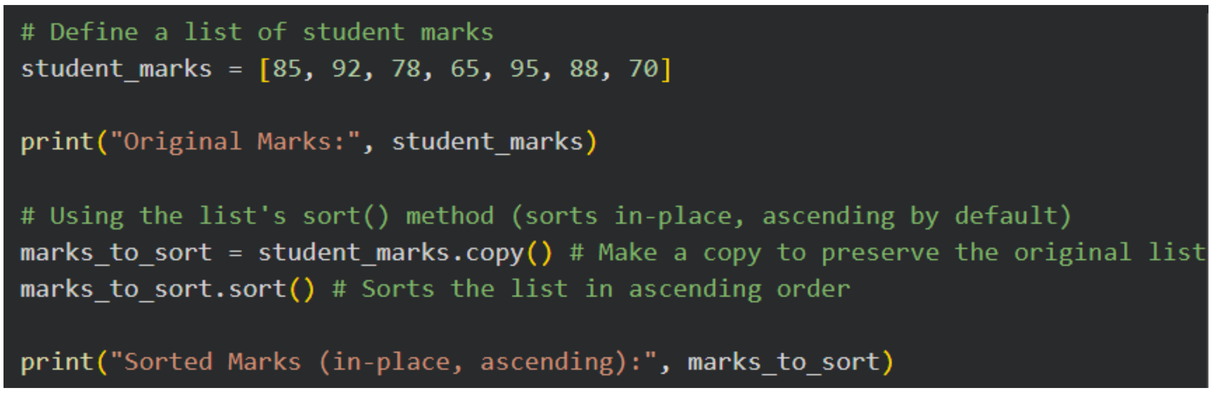


```
Original Marks: [85, 92, 78, 65, 95, 88, 70]
Sorted Marks (in-place, ascending): [65, 70, 78, 85, 88, 92, 95]
Sorted Marks (in-place, descending): [95, 92, 88, 85, 78, 70, 65]
Sorted Marks (new list, ascending): [65, 70, 78, 85, 88, 92, 95]
Sorted Marks (new list, descending): [95, 92, 88, 85, 78, 70, 65]
Original Marks (after sorting demonstrations): [85, 92, 78, 65, 95, 88, 70]
```

Explanation:
This program shows how to sort a list of student marks in Python. It uses the sort () method to sort the list in place, meaning the original list is modified. It also uses the sorted () function, which creates and returns a new sorted list without changing the original one. Both ascending and descending order sorting are demonstrated.

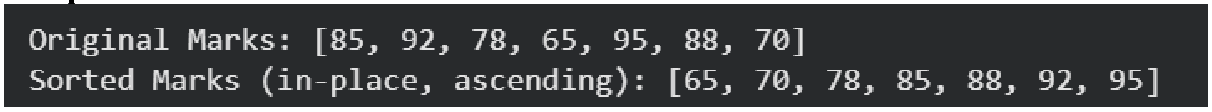
Prompt 2: Write a Python program to sort a list of student marks in ascending order, where marks are integers and using only sort function

Code:



```python
# Define a list of student marks
student_marks = [85, 92, 78, 65, 95, 88, 70]

print("Original Marks:", student_marks)

# Using the list's sort() method (sorts in-place, ascending by default)
marks_to_sort = student_marks.copy() # Make a copy to preserve the original list
marks_to_sort.sort() # Sorts the list in ascending order

print("Sorted Marks (in-place, ascending):", marks_to_sort)
```

Output:



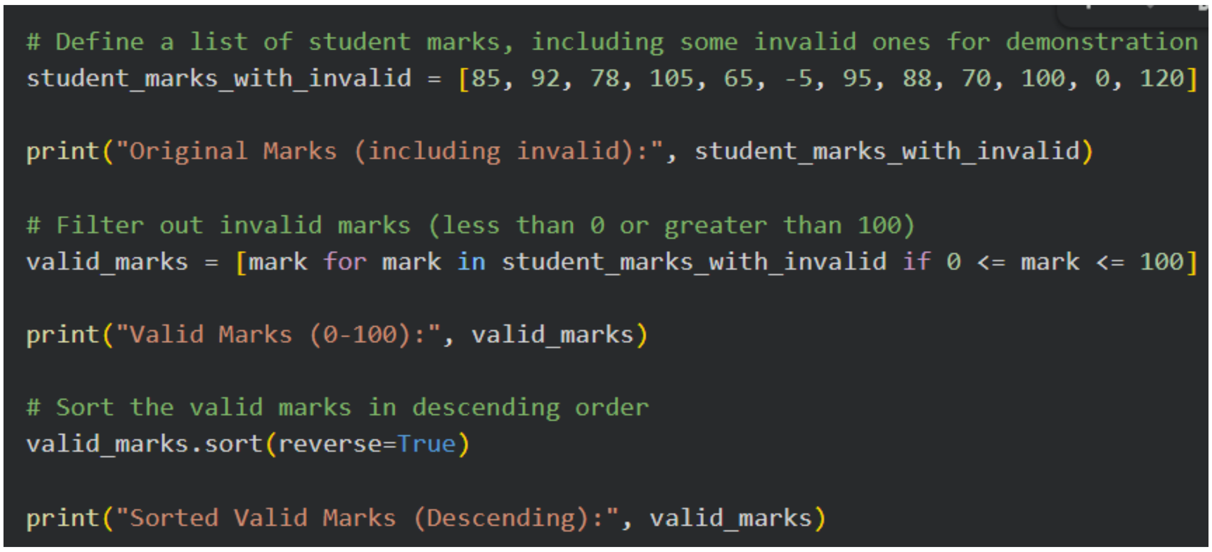
```
Original Marks: [85, 92, 78, 65, 95, 88, 70]
Sorted Marks (in-place, ascending): [65, 70, 78, 85, 88, 92, 95]
```

Explanation:

The previous code provides a comprehensive comparison of different sorting techniques, while the current code focuses only on in-place ascending sorting using the sort () method. This makes the current version easier to understand but less flexible than the previous one.

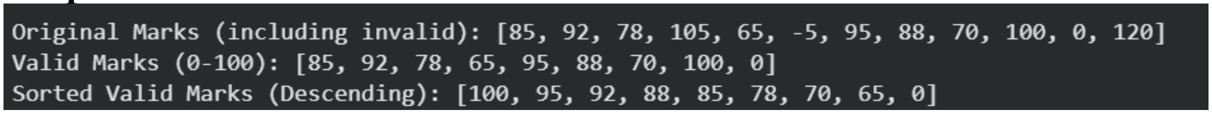
Prompt 3: Write a Python program that takes a list of student marks (integers between 0 and 100) and sorts them in descending order. The program should ignore invalid marks (less than 0 or greater than 100) and display the sorted list.

Code:



```python
# Define a list of student marks, including some invalid ones for demonstration
student_marks_with_invalid = [85, 92, 78, 105, 65, -5, 95, 88, 70, 100, 0, 120]

print("Original Marks (including invalid):", student_marks_with_invalid)

# Filter out invalid marks (less than 0 or greater than 100)
valid_marks = [mark for mark in student_marks_with_invalid if 0 <= mark <= 100]

print("Valid Marks (0-100):", valid_marks)

# Sort the valid marks in descending order
valid_marks.sort(reverse=True)

print("Sorted Valid Marks (Descending):", valid_marks)
```

Output:



```
Original Marks (including invalid): [85, 92, 78, 105, 65, -5, 95, 88, 70, 100, 0, 120]
Valid Marks (0-100): [85, 92, 78, 65, 95, 88, 70, 100, 0]
Sorted Valid Marks (Descending): [100, 95, 92, 88, 85, 78, 70, 65, 0]
```

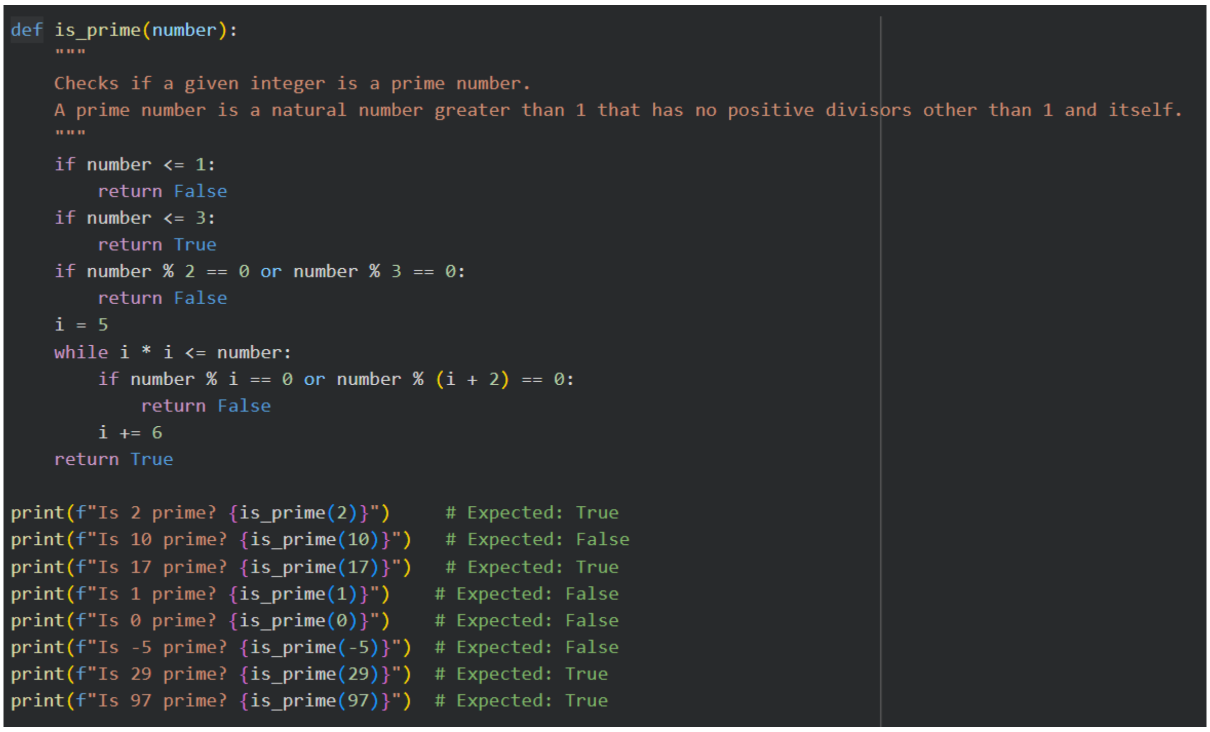
Explanation:

The updated code improves the previous code by validating the input data before sorting. It removes invalid marks and applies sorting only to valid values, making the program more reliable and aligned with real-world constraints.

Task 3:

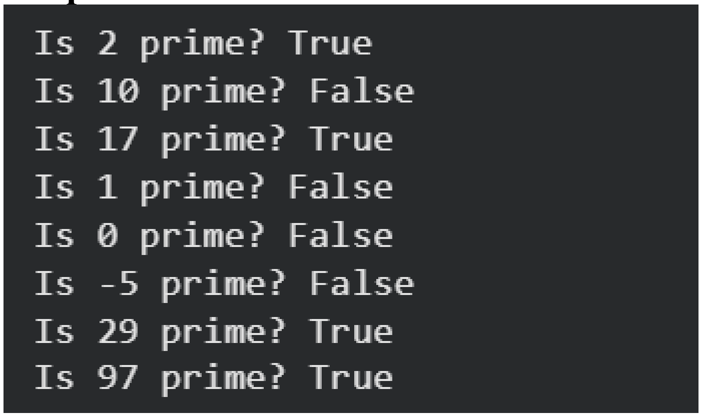
Prompt 1: Write a function that checks whether a number is prime.

Code:



```python
def is_prime(number):
    """
    Checks if a given integer is a prime number.
    A prime number is a natural number greater than 1 that has no positive divisors other than 1 and itself.
    """
    if number <= 1:
        return False
    if number <= 3:
        return True
    if number % 2 == 0 or number % 3 == 0:
        return False
    i = 5
    while i * i <= number:
        if number % i == 0 or number % (i + 2) == 0:
            return False
        i += 6
    return True

print(f"Is 2 prime? {is_prime(2)}")     # Expected: True
print(f"Is 10 prime? {is_prime(10)}")   # Expected: False
print(f"Is 17 prime? {is_prime(17)}")   # Expected: True
print(f"Is 1 prime? {is_prime(1)}")    # Expected: False
print(f"Is 0 prime? {is_prime(0)}")    # Expected: False
print(f"Is -5 prime? {is_prime(-5)}")  # Expected: False
print(f"Is 29 prime? {is_prime(29)}")  # Expected: True
print(f"Is 97 prime? {is_prime(97)}")  # Expected: True
```

Output:



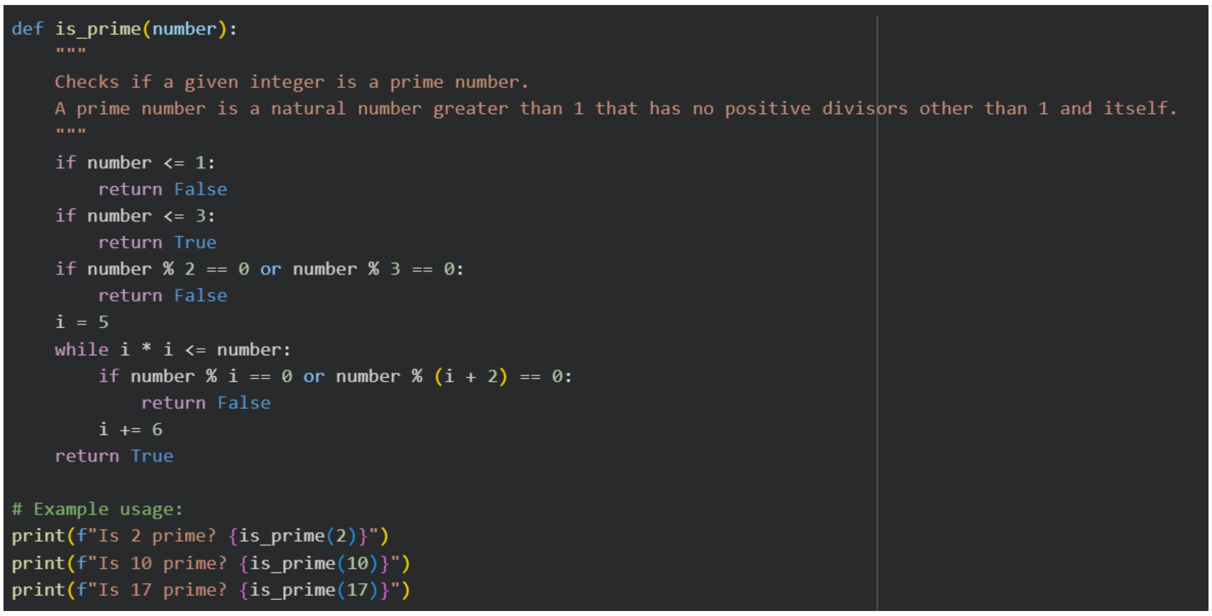
```
Is 2 prime? True
Is 10 prime? False
Is 17 prime? True
Is 1 prime? False
Is 0 prime? False
Is -5 prime? False
Is 29 prime? True
Is 97 prime? True
```

Explanation:

The function produces correct results because it includes proper logical checks and efficient iteration. This highlights how implementation quality can compensate for ambiguity, but also shows why few-shot prompting is important—it clearly defines expected behavior and improves consistency and correctness.

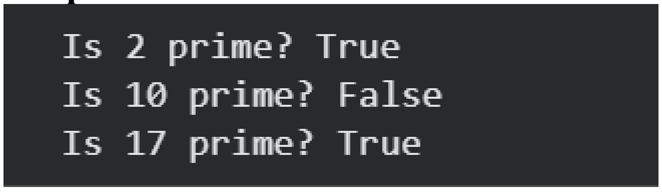
Prompt 2: Write a Python function is_prime(n) that checks whether a number is prime. A prime number is a natural number greater than 1 that has exactly two positive divisors: 1 and itself.

Code:



```python
def is_prime(number):
    """
    Checks if a given integer is a prime number.
    A prime number is a natural number greater than 1 that has no positive divisors other than 1 and itself.
    """
    if number <= 1:
        return False
    if number <= 3:
        return True
    if number % 2 == 0 or number % 3 == 0:
        return False
    i = 5
    while i * i <= number:
        if number % i == 0 or number % (i + 2) == 0:
            return False
        i += 6
    return True

# Example usage:
print(f"Is 2 prime? {is_prime(2)}")
print(f"Is 10 prime? {is_prime(10)}")
print(f"Is 17 prime? {is_prime(17)}")
```

Output:



```
Is 2 prime? True
Is 10 prime? False
Is 17 prime? True
```

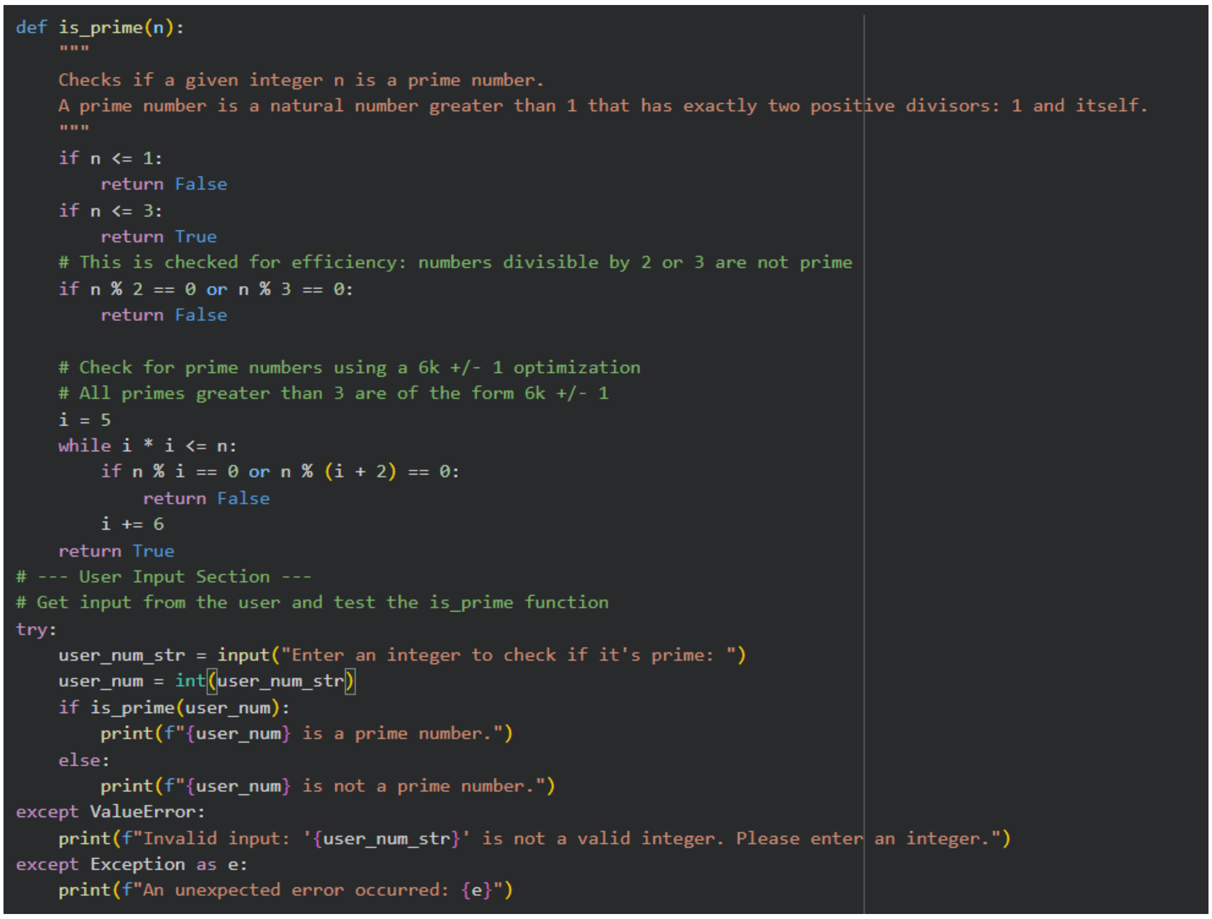
Explanation:

The previous code thoroughly tested the is_prime function with normal and edge cases, while the current code demonstrates only basic prime number validation. The earlier version provides stronger proof of correctness, whereas the current version is simpler and more concise for demonstration purposes.

Prompt 3:

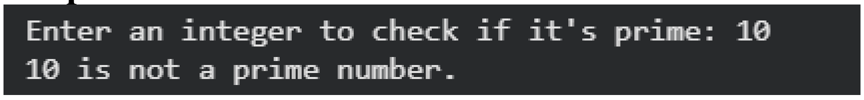
Write a Python function is_prime(n) that checks whether a number is prime. And the input is taken from the user

Code:



```python
def is_prime(n):
    """
    Checks if a given integer n is a prime number.
    A prime number is a natural number greater than 1 that has exactly two positive divisors: 1 and itself.
    """
    if n <= 1:
        return False
    if n <= 3:
        return True
    # This is checked for efficiency: numbers divisible by 2 or 3 are not prime
    if n % 2 == 0 or n % 3 == 0:
        return False

    # Check for prime numbers using a 6k +/- 1 optimization
    # All primes greater than 3 are of the form 6k +/- 1
    i = 5
    while i * i <= n:
        if n % i == 0 or n % (i + 2) == 0:
            return False
        i += 6
    return True
# --- User Input Section ---
# Get input from the user and test the is_prime function
try:
    user_num_str = input("Enter an integer to check if it's prime: ")
    user_num = int(user_num_str)
    if is_prime(user_num):
        print(f"{user_num} is a prime number.")
    else:
        print(f"{user_num} is not a prime number.")
except ValueError:
    print(f"Invalid input: '{user_num_str}' is not a valid integer. Please enter an integer.")
except Exception as e:
    print(f"An unexpected error occurred: {e}")
```

Output:



```
Enter an integer to check if it's prime: 10
10 is not a prime number.
```

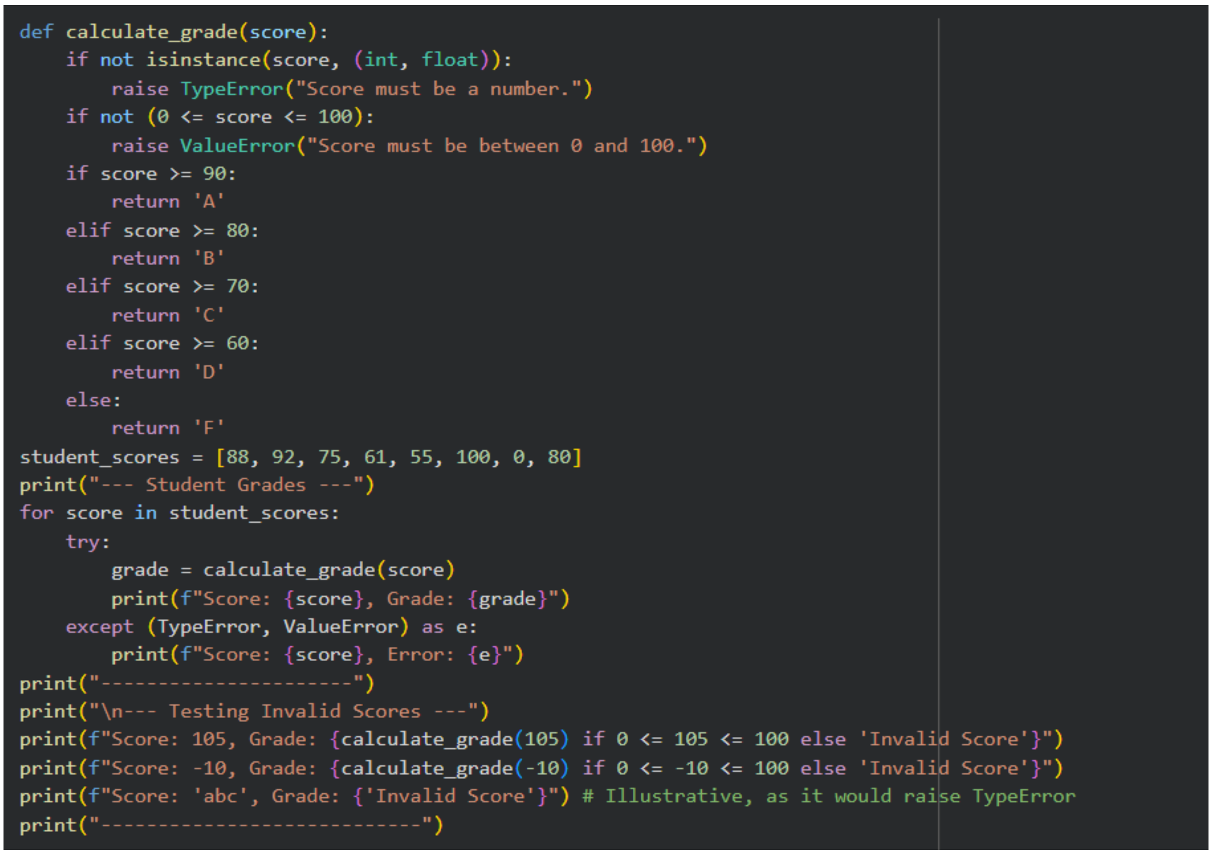
Explanation:

The main difference is that the previous code demonstrated the function with preset test cases, while the new code accepts user input, validates it, and provides interactive feedback. The new version is more robust, flexible, and suitable for real-world use, while the earlier version is simpler and ideal for demonstration or testing.

Task 4:

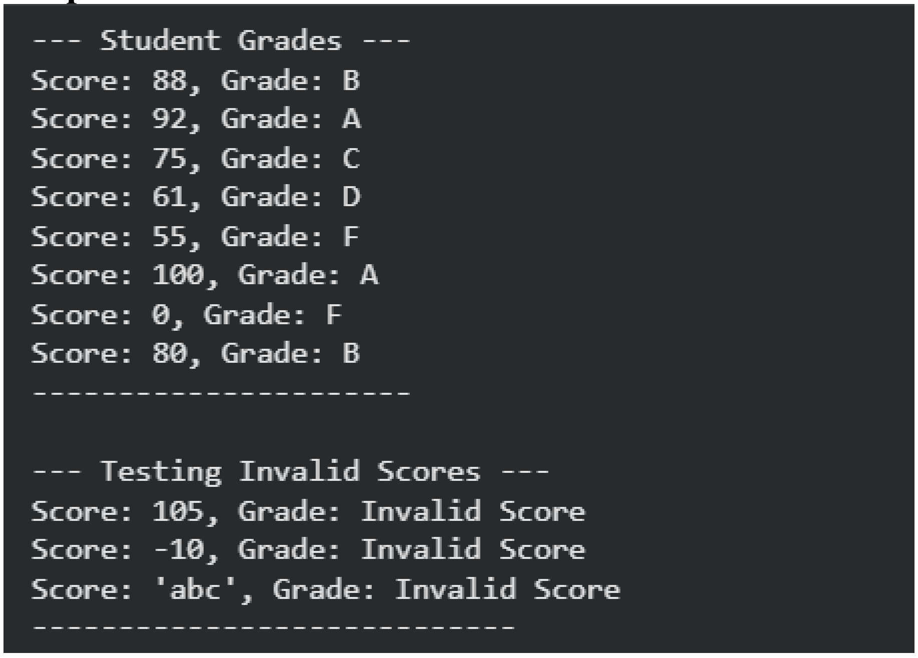
Prompt: Create a program to calculate student grades.

Code:



```python
def calculate_grade(score):
    if not isinstance(score, (int, float)):
        raise TypeError("Score must be a number.")
    if not (0 <= score <= 100):
        raise ValueError("Score must be between 0 and 100.")
    if score >= 90:
        return 'A'
    elif score >= 80:
        return 'B'
    elif score >= 70:
        return 'C'
    elif score >= 60:
        return 'D'
    else:
        return 'F'
student_scores = [88, 92, 75, 61, 55, 100, 0, 80]
print("--- Student Grades ---")
for score in student_scores:
    try:
        grade = calculate_grade(score)
        print(f"Score: {score}, Grade: {grade}")
    except (TypeError, ValueError) as e:
        print(f"Score: {score}, Error: {e}")
print("----------------------")
print("\n--- Testing Invalid Scores ---")
print(f"Score: 105, Grade: {calculate_grade(105) if 0 <= 105 <= 100 else 'Invalid Score'}")
print(f"Score: -10, Grade: {calculate_grade(-10) if 0 <= -10 <= 100 else 'Invalid Score'}")
print(f"Score: 'abc', Grade: {'Invalid Score'}") # Illustrative, as it would raise TypeError
print("----------------------------")
```

Output:



```
--- Student Grades ---
Score: 88, Grade: B
Score: 92, Grade: A
Score: 75, Grade: C
Score: 61, Grade: D
Score: 55, Grade: F
Score: 100, Grade: A
Score: 0, Grade: F
Score: 80, Grade: B
----------------------


--- Testing Invalid Scores ---
Score: 105, Grade: Invalid Score
Score: -10, Grade: Invalid Score
Score: 'abc', Grade: Invalid Score
----------------------------
```

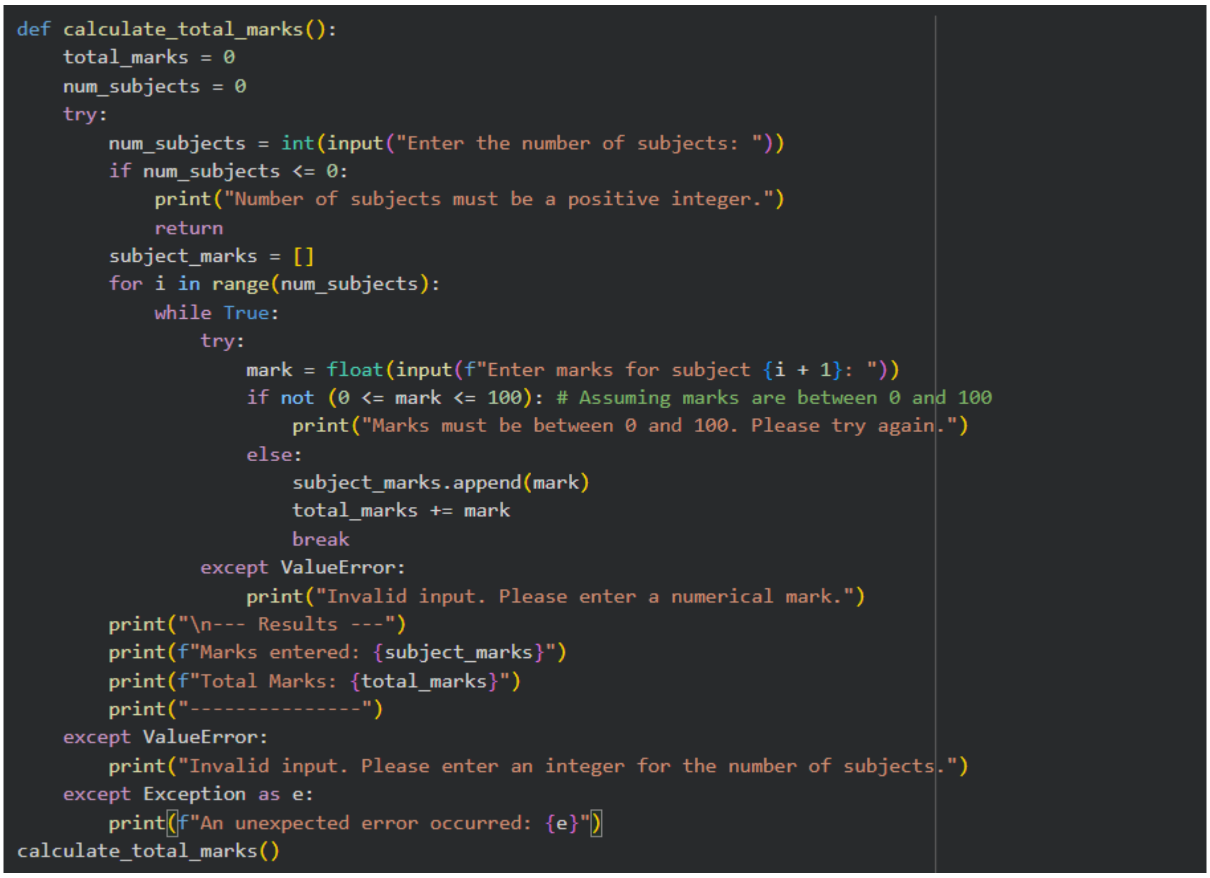
Explanation:

The program calculates student grades using the calculate_grade function, which validates inputs (must be numbers between 0 and 100) and assigns grades based on score ranges. It iterates through a list of scores, displays grades for valid inputs, and handles invalid scores gracefully using exception handling.

Prompt 2:

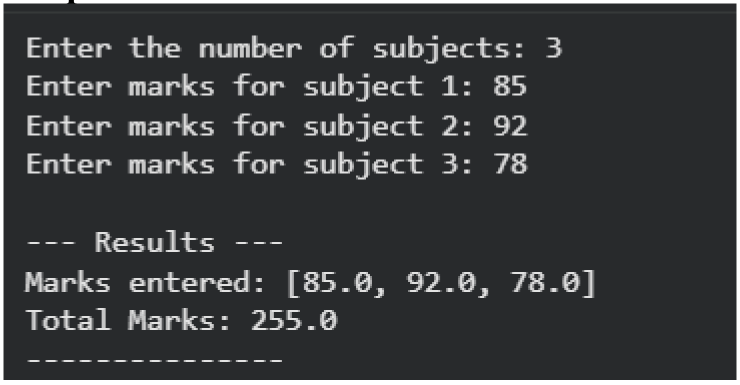
Create a program that asks the user for marks of a student in multiple subjects and calculates the total marks.

Code:



```python
def calculate_total_marks():
    total_marks = 0
    num_subjects = 0
    try:
        num_subjects = int(input("Enter the number of subjects: "))
        if num_subjects <= 0:
            print("Number of subjects must be a positive integer.")
            return
        subject_marks = []
        for i in range(num_subjects):
            while True:
                try:
                    mark = float(input(f"Enter marks for subject {i + 1}: "))
                    if not (0 <= mark <= 100): # Assuming marks are between 0 and 100
                        print("Marks must be between 0 and 100. Please try again.")
                    else:
                        subject_marks.append(mark)
                        total_marks += mark
                        break
                except ValueError:
                    print("Invalid input. Please enter a numerical mark.")
        print("\n--- Results ---")
        print(f"Marks entered: {subject_marks}")
        print(f"Total Marks: {total_marks}")
        print("---------------")
    except ValueError:
        print("Invalid input. Please enter an integer for the number of subjects.")
    except Exception as e:
        print(f"An unexpected error occurred: {e}")
calculate_total_marks()
```

Output:

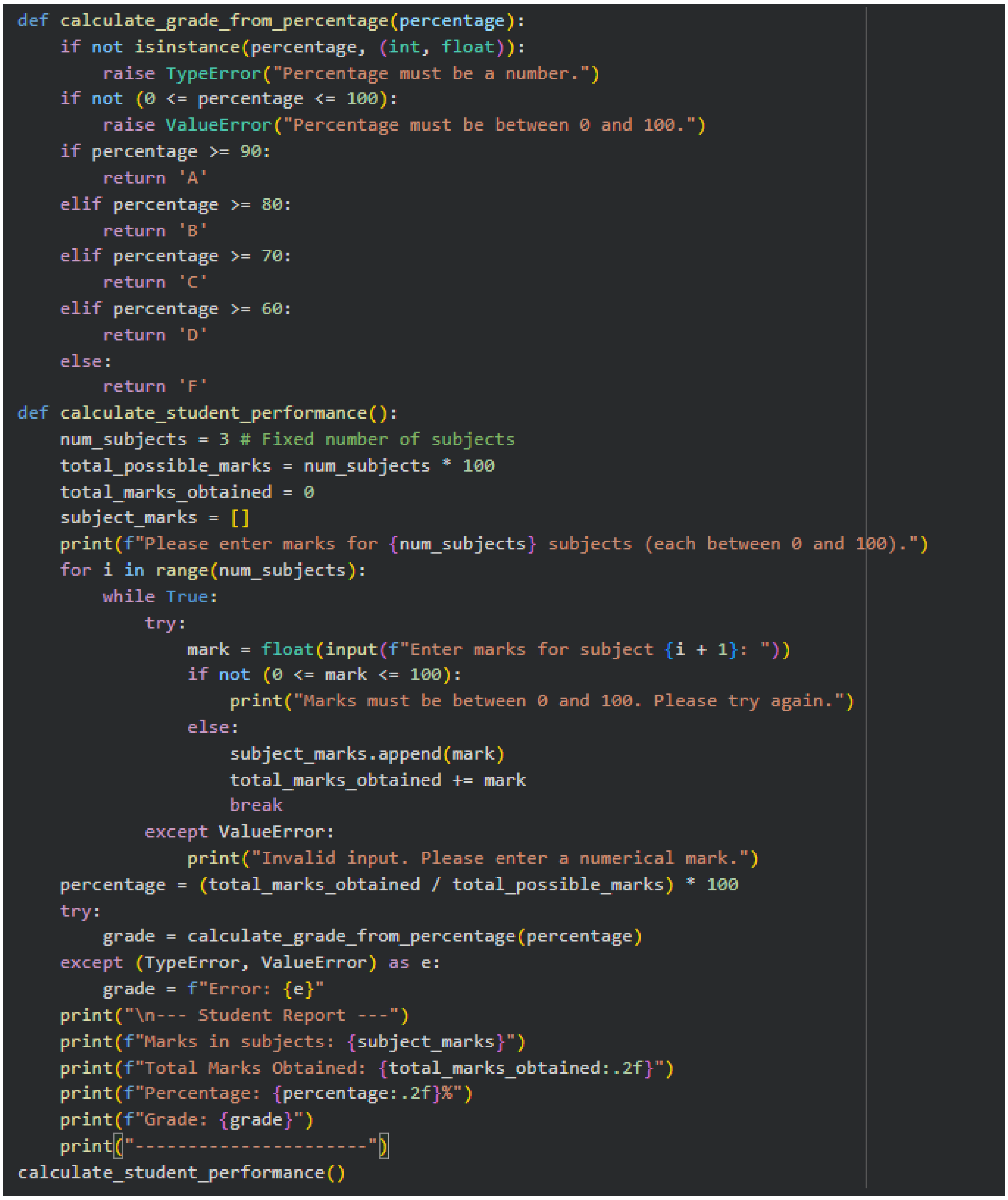


```
Enter the number of subjects: 3
Enter marks for subject 1: 85
Enter marks for subject 2: 92
Enter marks for subject 3: 78

--- Results ---
Marks entered: [85.0, 92.0, 78.0]
Total Marks: 255.0
---------------
```

Explanation:

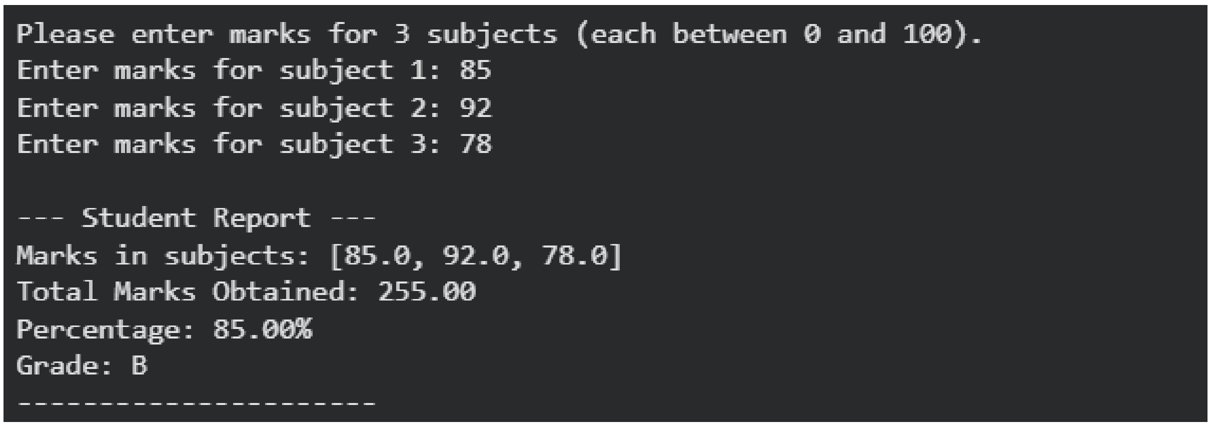
The previous code calculates grades for a fixed list of scores, while the new code calculates total marks for a user-defined number of subjects. The new code is more interactive, includes input validation for both the number of subjects and marks, and displays both individual marks and the total, making it suitable for a student grading system UI.

Prompt 3:
Create a Python program that asks the user to enter marks for 3 subjects. The program should calculate: Total marks Percentage Grade based on percentage
Examples:
Input: [85, 92, 78] → Output: Total = 255, Percentage = 85%, Grade = A
Input: [65, 70, 60] → Output: Total = 195, Percentage = 65%, Grade = B
Input: [45, 50, 40] → Output: Total = 135, Percentage = 45%, Grade = C
Code:



```python
def calculate_grade_from_percentage(percentage):
    if not isinstance(percentage, (int, float)):
        raise TypeError("Percentage must be a number.")
    if not (0 <= percentage <= 100):
        raise ValueError("Percentage must be between 0 and 100.")
    if percentage >= 90:
        return 'A'
    elif percentage >= 80:
        return 'B'
    elif percentage >= 70:
        return 'C'
    elif percentage >= 60:
        return 'D'
    else:
        return 'F'
def calculate_student_performance():
    num_subjects = 3 # Fixed number of subjects
    total_possible_marks = num_subjects * 100
    total_marks_obtained = 0
    subject_marks = []
    print(f"Please enter marks for {num_subjects} subjects (each between 0 and 100).")
    for i in range(num_subjects):
        while True:
            try:
                mark = float(input(f"Enter marks for subject {i + 1}: "))
                if not (0 <= mark <= 100):
                    print("Marks must be between 0 and 100. Please try again.")
                else:
                    subject_marks.append(mark)
                    total_marks_obtained += mark
                    break
            except ValueError:
                print("Invalid input. Please enter a numerical mark.")
    percentage = (total_marks_obtained / total_possible_marks) * 100
    try:
        grade = calculate_grade_from_percentage(percentage)
    except (TypeError, ValueError) as e:
        grade = f"Error: {e}"
    print("\n--- Student Report ---")
    print(f"Marks in subjects: {subject_marks}")
    print(f"Total Marks Obtained: {total_marks_obtained:.2f}")
    print(f"Percentage: {percentage:.2f}%")
    print(f"Grade: {grade}")
    print("----------------------")
calculate_student_performance()
```

Output:



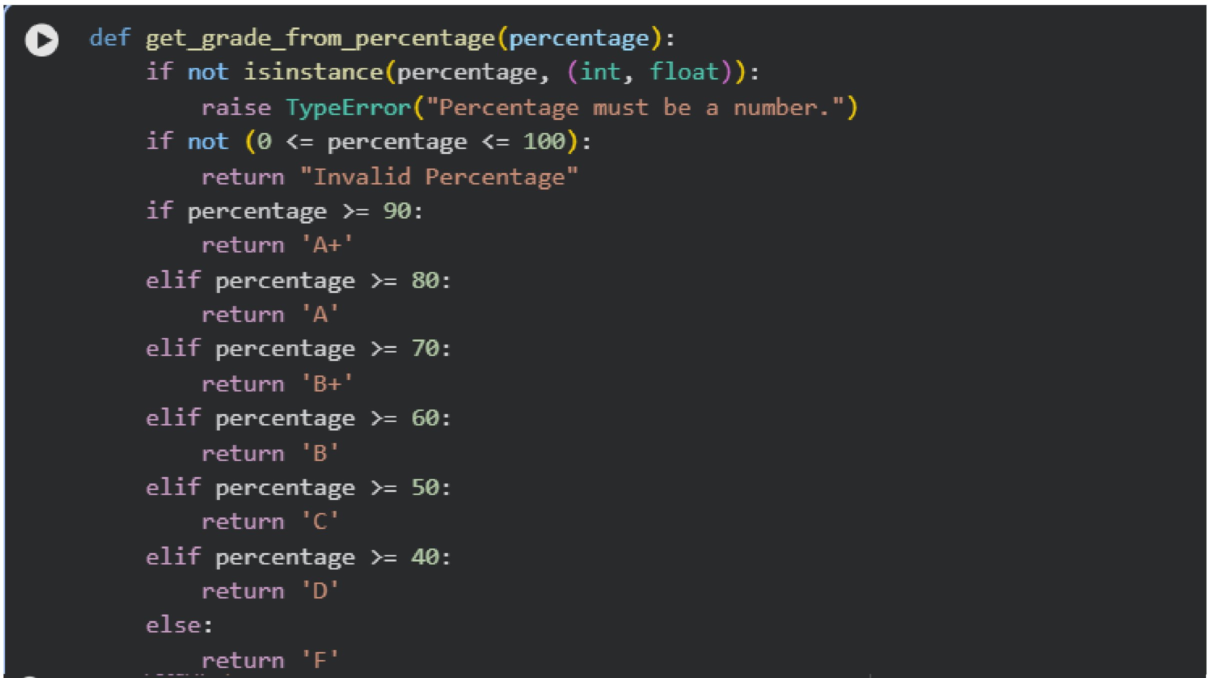
```
Please enter marks for 3 subjects (each between 0 and 100).
Enter marks for subject 1: 85
Enter marks for subject 2: 92
Enter marks for subject 3: 78

--- Student Report ---
Marks in subjects: [85.0, 92.0, 78.0]
Total Marks Obtained: 255.00
Percentage: 85.00%
Grade: B
----------------------
```

Explanation:
This program provides a well-structured student grading system that calculates total marks, percentage, and grade based on user input. It validates inputs, handles errors gracefully, and displays a clear and informative report.

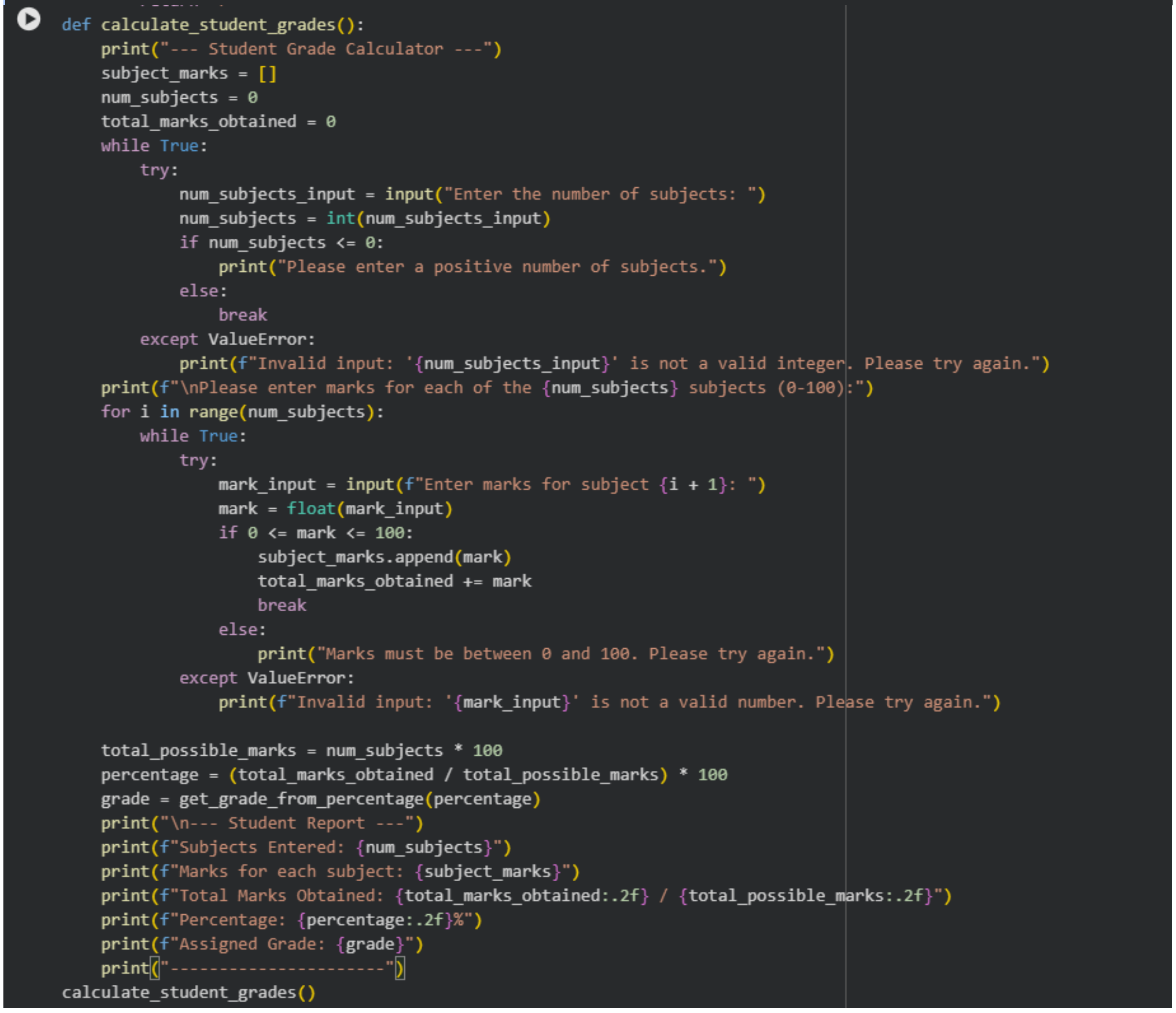
Prompt 4:

- Design a Python program with a user-friendly interface to calculate student grades.
- The program should: Ask the user to input marks for multiple subjects. Calculate the total marks.
- Calculate the percentage.
- Assign a grade based on the percentage:
  - 90–100 → A+
  - 80–89 → A
  - 70–79 → B+
  - 60–69 → B
  - 50–59 → C
  - 40–49 → D
  - Below 40 → F
- Display all results clearly with labels.

Code:

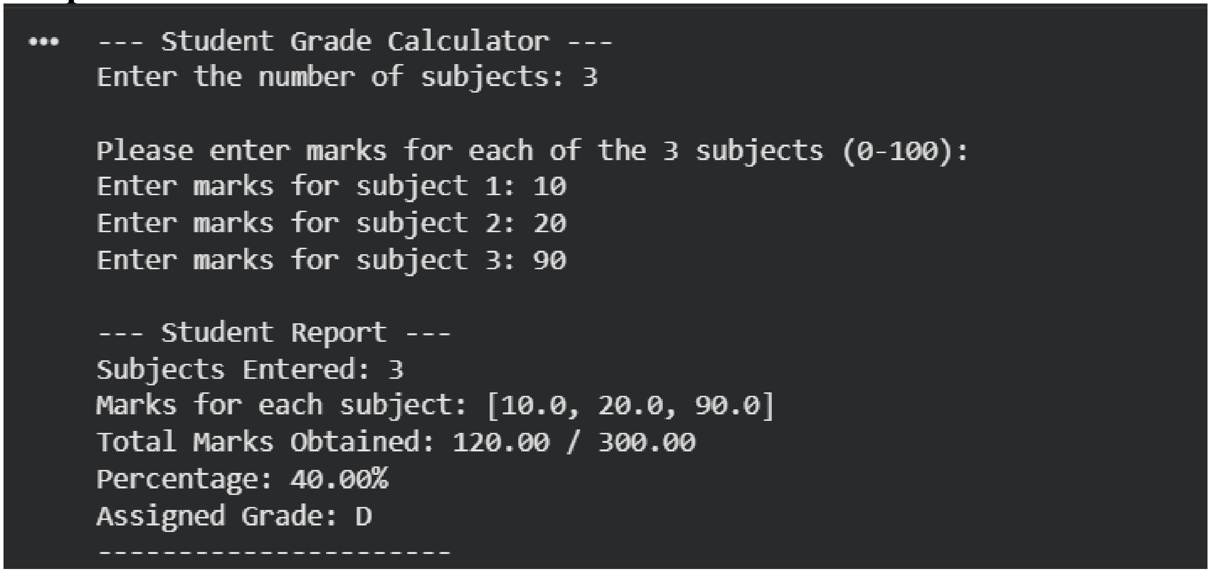


```python
def get_grade_from_percentage(percentage):
    if not isinstance(percentage, (int, float)):
        raise TypeError("Percentage must be a number.")
    if not (0 <= percentage <= 100):
        return "Invalid Percentage"
    if percentage >= 90:
        return 'A+'
    elif percentage >= 80:
        return 'A'
    elif percentage >= 70:
        return 'B+'
    elif percentage >= 60:
        return 'B'
    elif percentage >= 50:
        return 'C'
    elif percentage >= 40:
        return 'D'
    else:
        return 'F'
```



```python
def calculate_student_grades():
    print("--- Student Grade Calculator ---")
    subject_marks = []
    num_subjects = 0
    total_marks_obtained = 0
    while True:
        try:
            num_subjects_input = input("Enter the number of subjects: ")
            num_subjects = int(num_subjects_input)
            if num_subjects <= 0:
                print("Please enter a positive number of subjects.")
            else:
                break
        except ValueError:
            print(f"Invalid input: '{num_subjects_input}' is not a valid integer. Please try again.")
    print(f"\nPlease enter marks for each of the {num_subjects} subjects (0-100):")
    for i in range(num_subjects):
        while True:
            try:
                mark_input = input(f"Enter marks for subject {i + 1}: ")
                mark = float(mark_input)
                if 0 <= mark <= 100:
                    subject_marks.append(mark)
                    total_marks_obtained += mark
                    break
                else:
                    print("Marks must be between 0 and 100. Please try again.")
            except ValueError:
                print(f"Invalid input: '{mark_input}' is not a valid number. Please try again.")

    total_possible_marks = num_subjects * 100
    percentage = (total_marks_obtained / total_possible_marks) * 100
    grade = get_grade_from_percentage(percentage)
    print("\n--- Student Report ---")
    print(f"Subjects Entered: {num_subjects}")
    print(f"Marks for each subject: {subject_marks}")
    print(f"Total Marks Obtained: {total_marks_obtained:.2f} / {total_possible_marks:.2f}")
    print(f"Percentage: {percentage:.2f}%")
    print(f"Assigned Grade: {grade}")
    print("----------------------")
calculate_student_grades()
```

Output:



```
--- Student Grade Calculator ---
Enter the number of subjects: 3

Please enter marks for each of the 3 subjects (0-100):
Enter marks for subject 1: 10
Enter marks for subject 2: 20
Enter marks for subject 3: 90

--- Student Report ---
Subjects Entered: 3
Marks for each subject: [10.0, 20.0, 90.0]
Total Marks Obtained: 120.00 / 300.00
Percentage: 40.00%
Assigned Grade: D
----------------------
```

Explanation:
It dynamically accepts the number of subjects, validates all inputs, calculates total marks and percentage, assigns grades with a detailed scale, and displays a structured report.

Task 5:
Prompt refinement directly improves code quality and accuracy.
Moving from vague to few-shot and fully refined prompts ensures:

- Correct handling of edge cases and invalid inputs
- Better user experience and UI clarity
- Consistent and reliable outputs for a wide range of inputs