2303A51193

Batch : 04

Lab 3: Prompt Engineering – Improving Prompts and Context Management

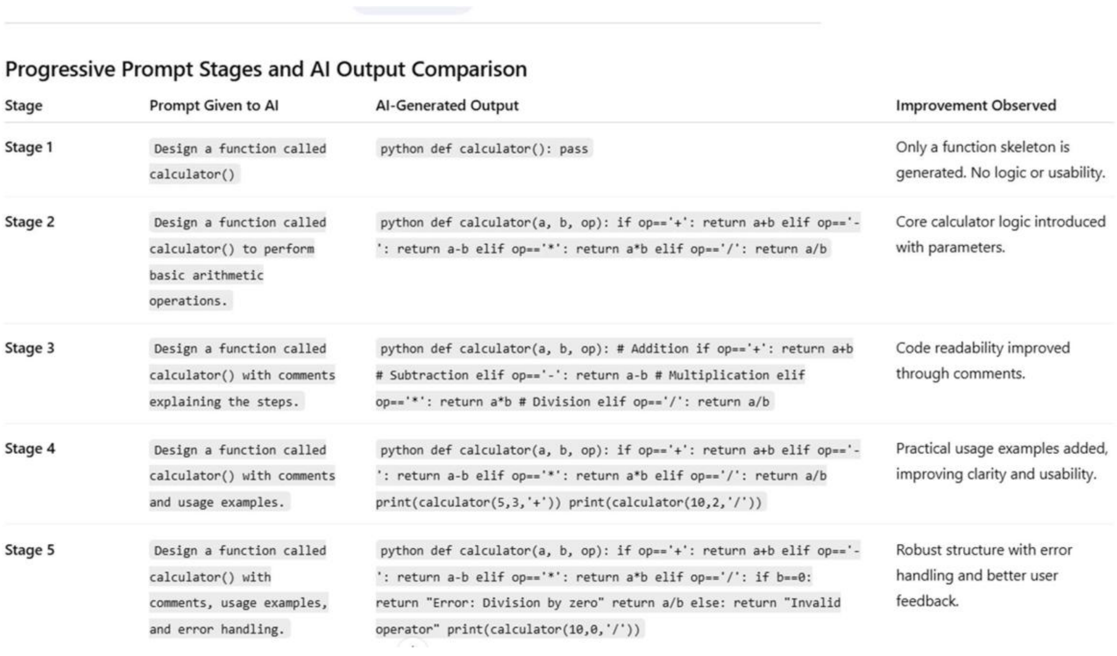
Lab Objectives:

Task Description-1

- Progressive Prompting for Calculator Design: Ask the AI to design a simple calculator program by initially providing only the function name. Gradually enhance the prompt by adding comments and usage examples.

Expected Output-1

- Comparison showing improvement in AI-generated calculator logic and structure.



Progressive Prompt Stages and AI Output Comparison

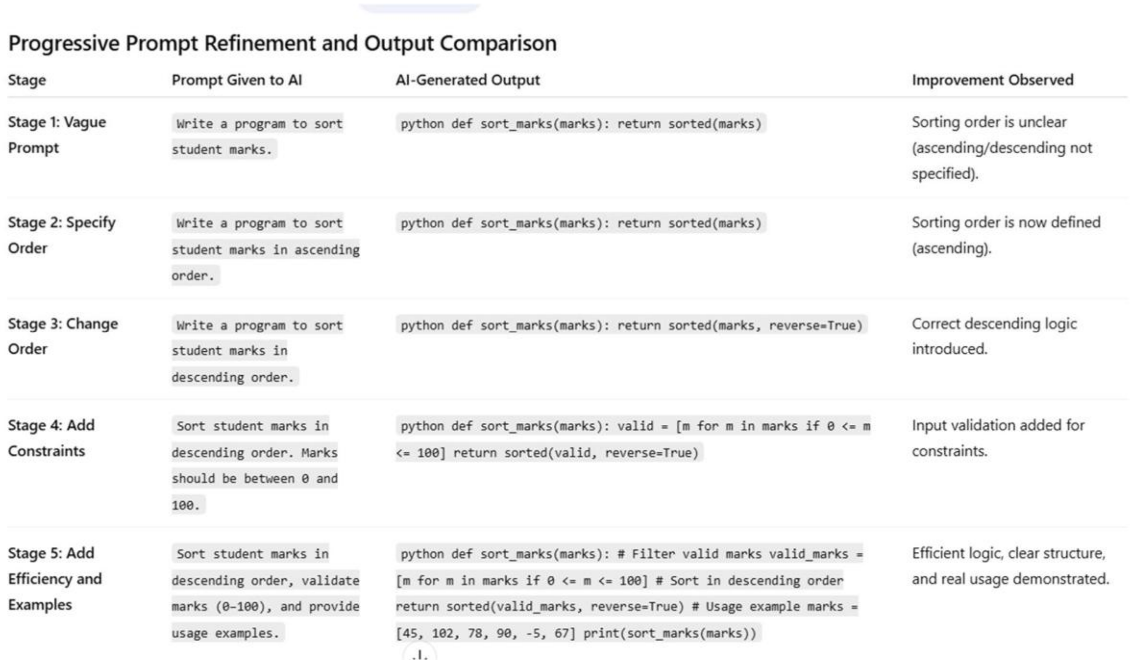
Stage	Prompt Given to AI	AI-Generated Output	Improvement Observed
Stage 1	`Design a function called calculator()`	`python def calculator(): pass`	Only a function skeleton is generated. No logic or usability.
Stage 2	`Design a function called calculator() to perform basic arithmetic operations.`	`python def calculator(a, b, op): if op=='+': return a+b elif op=='-': return a-b elif op=='*': return a*b elif op=='/': return a/b`	Core calculator logic introduced with parameters.
Stage 3	`Design a function called calculator() with comments explaining the steps.`	`python def calculator(a, b, op): # Addition if op=='+': return a+b # Subtraction elif op=='-': return a-b # Multiplication elif op=='*': return a*b # Division elif op=='/': return a/b`	Code readability improved through comments.
Stage 4	`Design a function called calculator() with comments and usage examples.`	`python def calculator(a, b, op): if op=='+': return a+b elif op=='-': return a-b elif op=='*': return a*b elif op=='/': return a/b print(calculator(5,3,'+')) print(calculator(10,2,'/'))`	Practical usage examples added, improving clarity and usability.
Stage 5	`Design a function called calculator() with comments, usage examples, and error handling.`	`python def calculator(a, b, op): if op=='+': return a+b elif op=='-': return a-b elif op=='*': return a*b elif op=='/': if b==0: return "Error: Division by zero" return a/b else: return "Invalid operator" print(calculator(10,0,'/'))`	Robust structure with error handling and better user feedback.

Task Description-2

- Refining Prompts for Sorting Logic: Start with a vague prompt for sorting student marks, then refine it to clearly specify sorting order and constraints.

Expected Output-2

- AI-generated sorting function evolves from ambiguous logic to an accurate and efficient implementation.



Progressive Prompt Refinement and Output Comparison

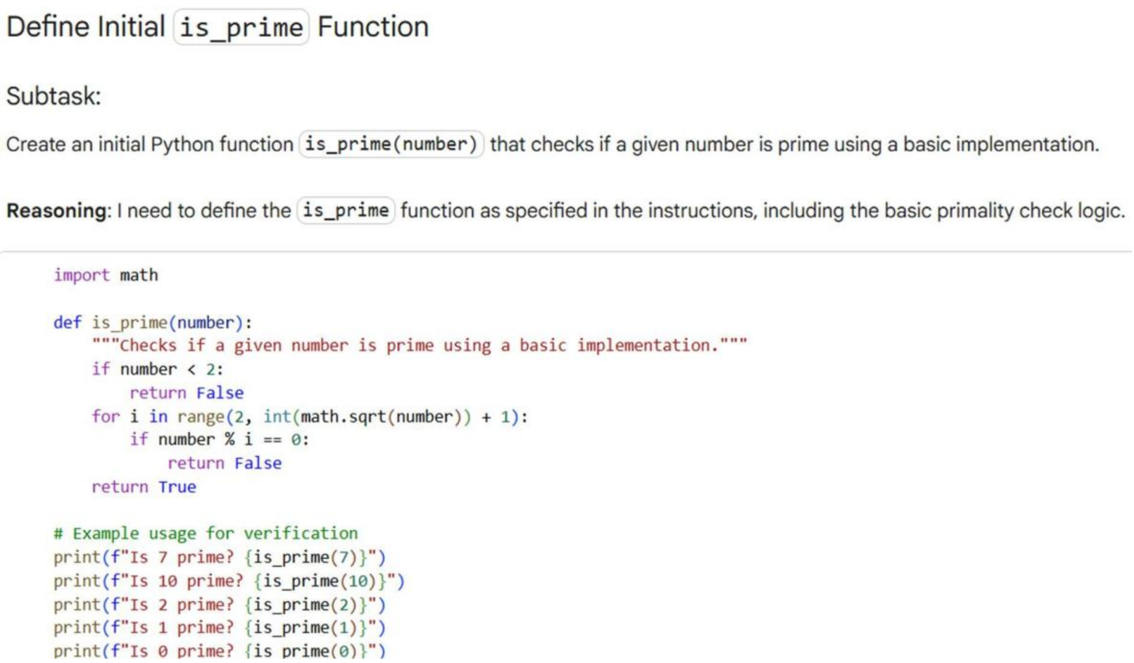
Stage	Prompt Given to AI	AI-Generated Output	Improvement Observed
Stage 1: Vague Prompt	`Write a program to sort student marks.`	`python def sort_marks(marks): return sorted(marks)`	Sorting order is unclear (ascending/descending not specified).
Stage 2: Specify Order	`Write a program to sort student marks in ascending order.`	`python def sort_marks(marks): return sorted(marks)`	Sorting order is now defined (ascending).
Stage 3: Change Order	`Write a program to sort student marks in descending order.`	`python def sort_marks(marks): return sorted(marks, reverse=True)`	Correct descending logic introduced.
Stage 4: Add Constraints	`Sort student marks in descending order. Marks should be between 0 and 100.`	`python def sort_marks(marks): valid = [m for m in marks if 0 <= m <= 100] return sorted(valid, reverse=True)`	Input validation added for constraints.
Stage 5: Add Efficiency and Examples	`Sort student marks in descending order, validate marks (0-100), and provide usage examples.`	`python def sort_marks(marks): # Filter valid marks valid_marks = [m for m in marks if 0 <= m <= 100] # Sort in descending order return sorted(valid_marks, reverse=True) # Usage example marks = [45, 102, 78, 90, -5, 67] print(sort_marks(marks))`	Efficient logic, clear structure, and real usage demonstrated.

Task Description-3

- Few-Shot Prompting for Prime Number Validation: Provide multiple input-output examples for a function that checks whether a number is prime. Observe how few-shot prompting improves correctness.

Expected Output-3

- Improved prime-checking function with better edge-case handling.



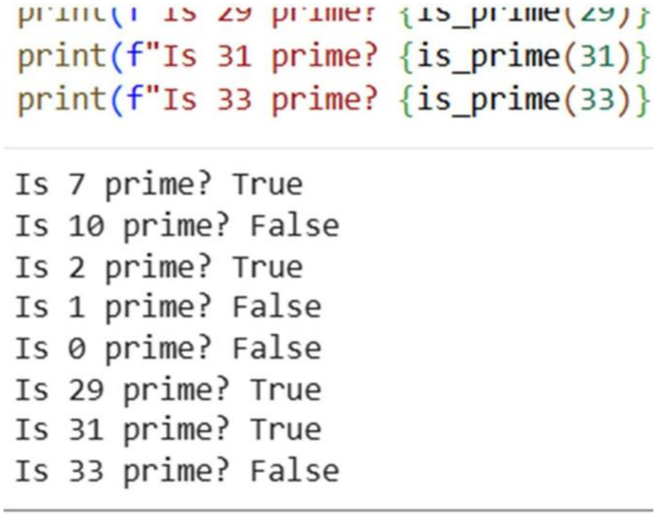
Define Initial `is_prime` Function

Subtask:

Create an initial Python function `is_prime(number)` that checks if a given number is prime using a basic implementation.

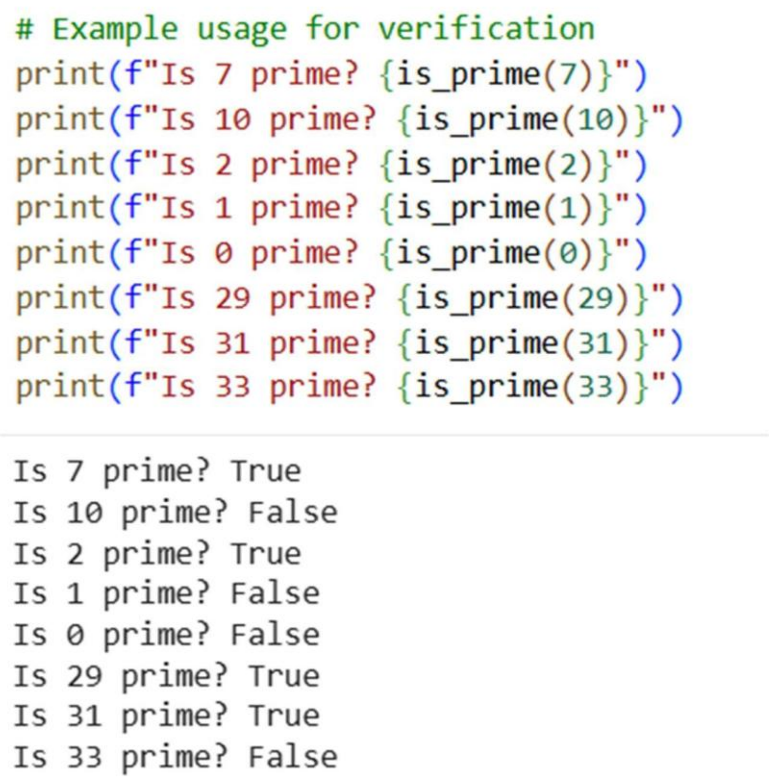
Reasoning: I need to define the `is_prime` function as specified in the instructions, including the basic primality check logic.

```
import math

def is_prime(number):
    """Checks if a given number is prime using a basic implementation."""
    if number < 2:
        return False
    for i in range(2, int(math.sqrt(number)) + 1):
        if number % i == 0:
            return False
    return True

# Example usage for verification
print(f"Is 7 prime? {is_prime(7)}")
print(f"Is 10 prime? {is_prime(10)}")
print(f"Is 2 prime? {is_prime(2)}")
print(f"Is 1 prime? {is_prime(1)}")
print(f"Is 0 prime? {is_prime(0)}")
```



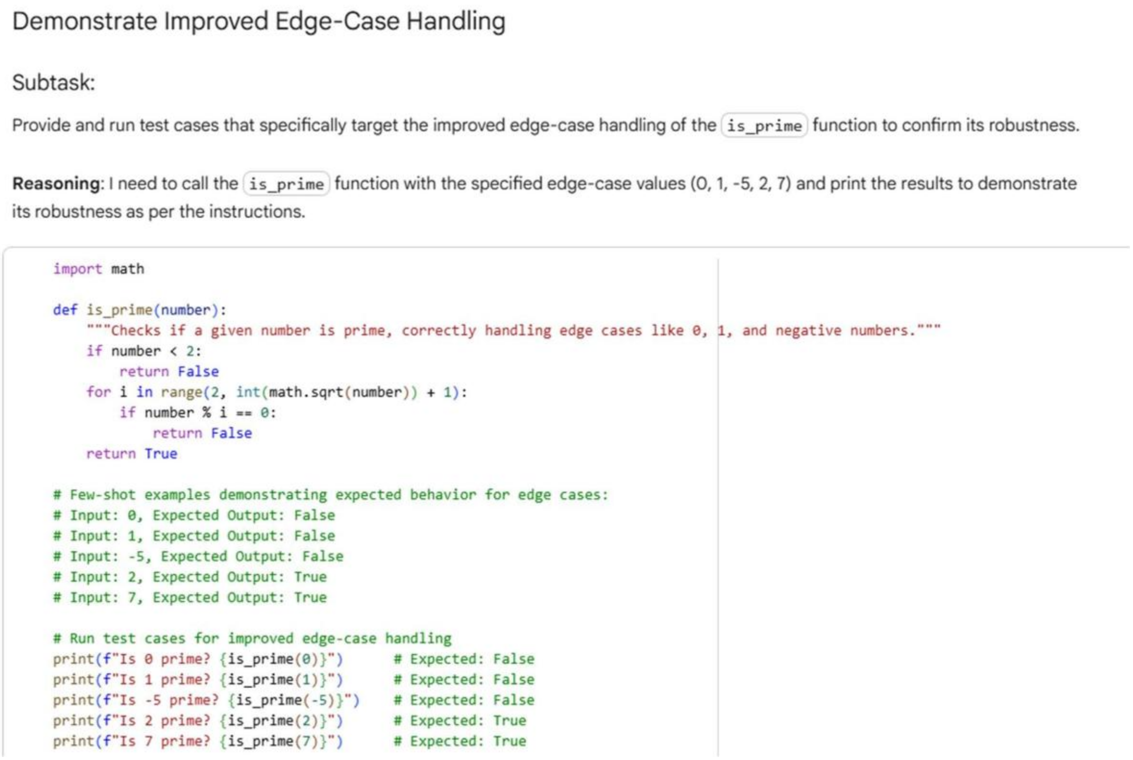
```
print(f"Is 29 prime? {is_prime(29)}
print(f"Is 31 prime? {is_prime(31)}
print(f"Is 33 prime? {is_prime(33)}
```

```
Is 7 prime? True
Is 10 prime? False
Is 2 prime? True
Is 1 prime? False
Is 0 prime? False
Is 29 prime? True
Is 31 prime? True
Is 33 prime? False
```



```
# Example usage for verification
print(f"Is 7 prime? {is_prime(7)}")
print(f"Is 10 prime? {is_prime(10)}")
print(f"Is 2 prime? {is_prime(2)}")
print(f"Is 1 prime? {is_prime(1)}")
print(f"Is 0 prime? {is_prime(0)}")
print(f"Is 29 prime? {is_prime(29)}")
print(f"Is 31 prime? {is_prime(31)}")
print(f"Is 33 prime? {is_prime(33)}")
```

```
Is 7 prime? True
Is 10 prime? False
Is 2 prime? True
Is 1 prime? False
Is 0 prime? False
Is 29 prime? True
Is 31 prime? True
Is 33 prime? False
```



Demonstrate Improved Edge-Case Handling

Subtask:

Provide and run test cases that specifically target the improved edge-case handling of the `is_prime` function to confirm its robustness.

Reasoning: I need to call the `is_prime` function with the specified edge-case values (0, 1, -5, 2, 7) and print the results to demonstrate its robustness as per the instructions.

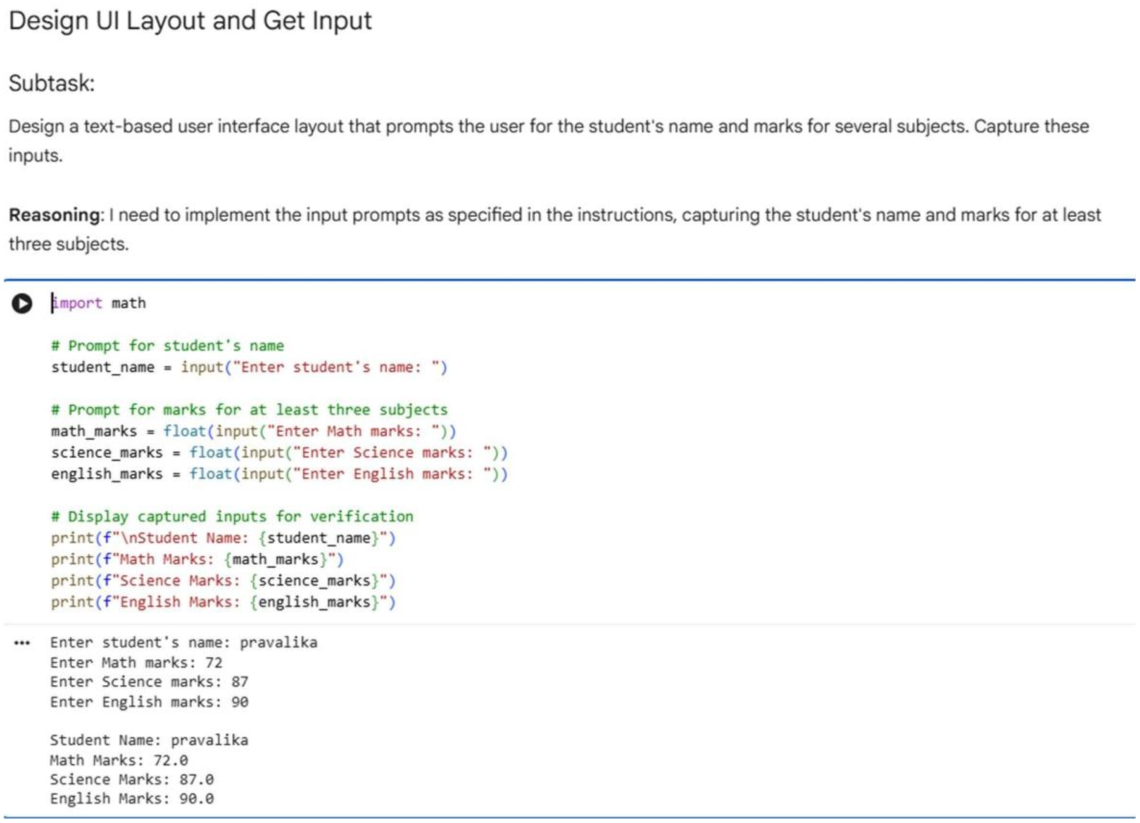
```
import math

def is_prime(number):
    """Checks if a given number is prime, correctly handling edge cases like 0, 1, and negative numbers."""
    if number < 2:
        return False
    for i in range(2, int(math.sqrt(number)) + 1):
        if number % i == 0:
            return False
    return True

# Few-shot examples demonstrating expected behavior for edge cases:
# Input: 0, Expected Output: False
# Input: 1, Expected Output: False
# Input: -5, Expected Output: False
# Input: 2, Expected Output: True
# Input: 7, Expected Output: True

# Run test cases for improved edge-case handling
print(f"Is 0 prime? {is_prime(0)}")      # Expected: False
print(f"Is 1 prime? {is_prime(1)}")      # Expected: False
print(f"Is -5 prime? {is_prime(-5)}")    # Expected: False
print(f"Is 2 prime? {is_prime(2)}")      # Expected: True
print(f"Is 7 prime? {is_prime(7)}")      # Expected: True
```

Task Description-4

- Prompt-Guided UI Design for Student Grading System: Create a user interface for a student grading system that calculates total marks, percentage, and grade based on user input.

Expected Output-4

- Well-structured UI code with accurate calculations and clear output display.



Design UI Layout and Get Input

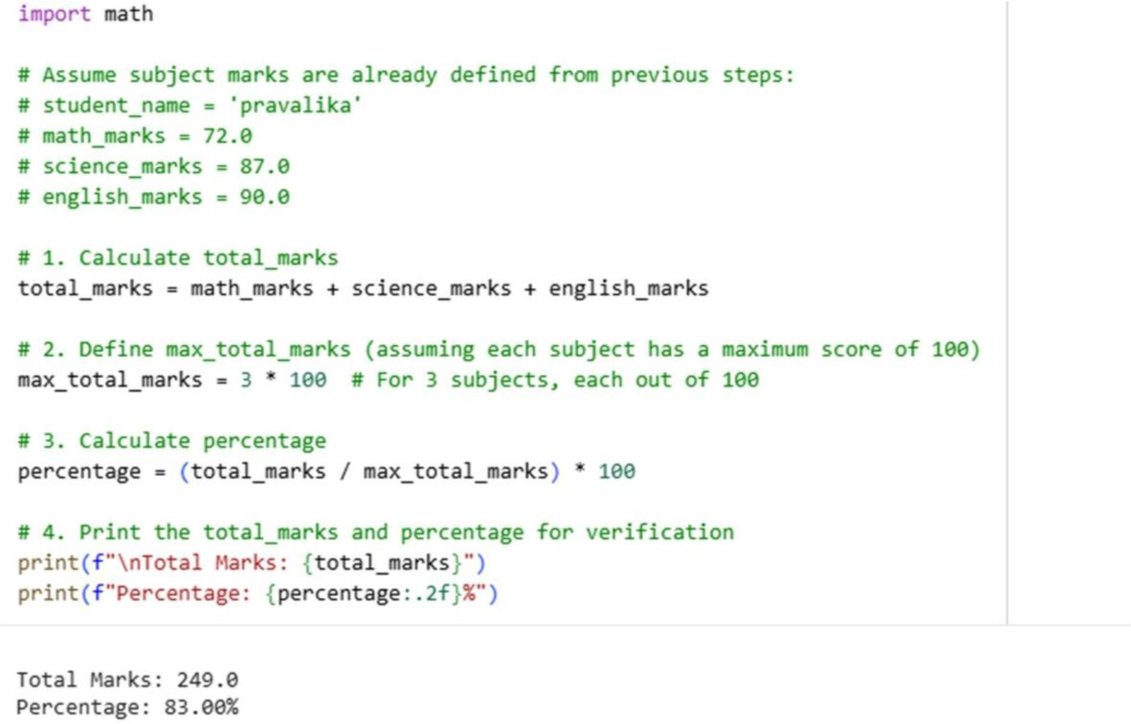
Subtask:

Design a text-based user interface layout that prompts the user for the student's name and marks for several subjects. Capture these inputs.

Reasoning: I need to implement the input prompts as specified in the instructions, capturing the student's name and marks for at least three subjects.

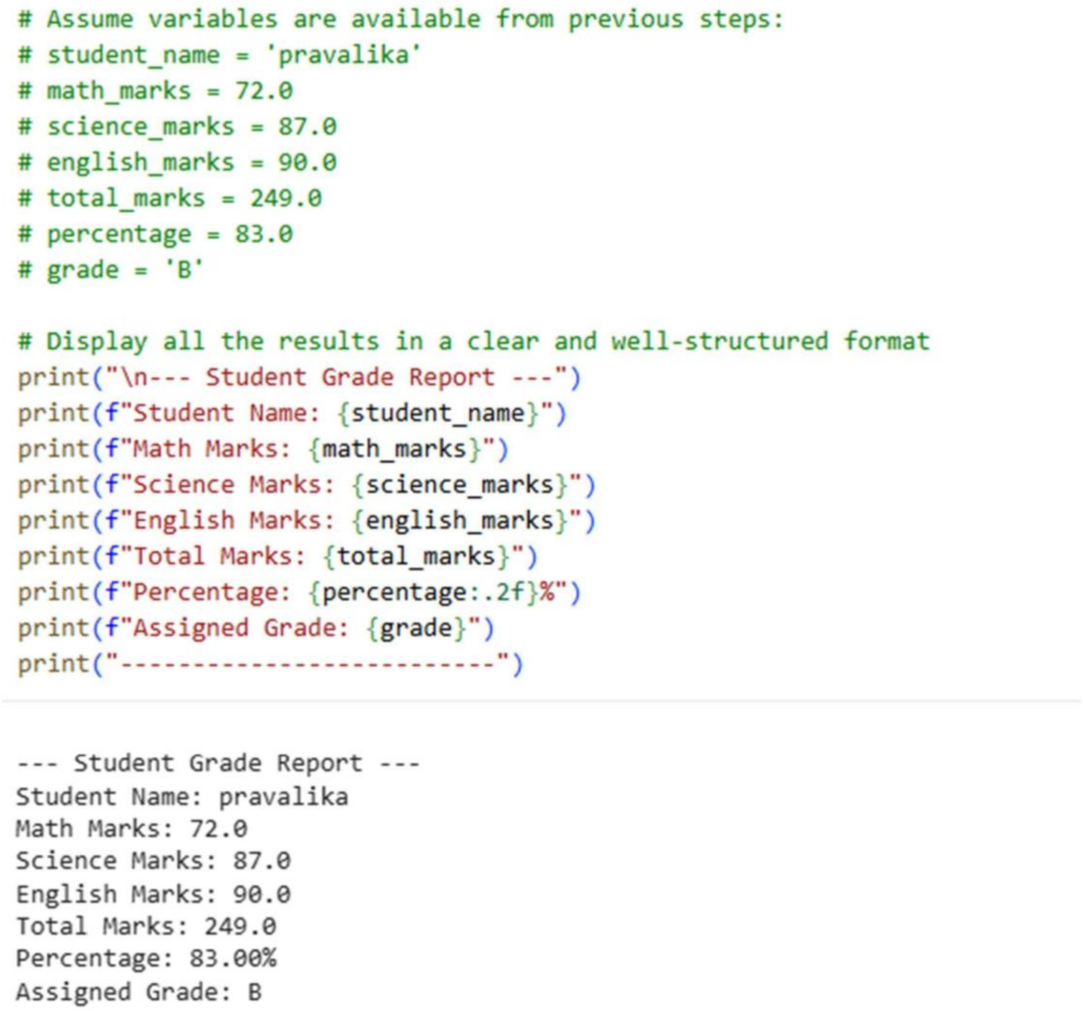
```
import math

# Prompt for student's name
student_name = input("Enter student's name: ")

# Prompt for marks for at least three subjects
math_marks = float(input("Enter Math marks: "))
science_marks = float(input("Enter Science marks: "))
english_marks = float(input("Enter English marks: "))

# Display captured inputs for verification
print(f"\nStudent Name: {student_name}")
print(f"Math Marks: {math_marks}")
print(f"Science Marks: {science_marks}")
print(f"English Marks: {english_marks}")
```

```
Enter student's name: pravalika
Enter Math marks: 72
Enter Science marks: 87
Enter English marks: 90

Student Name: pravalika
Math Marks: 72.0
Science Marks: 87.0
English Marks: 90.0
```



```python
import math

# Assume subject marks are already defined from previous steps:
# student_name = 'pravalika'
# math_marks = 72.0
# science_marks = 87.0
# english_marks = 90.0

# 1. Calculate total_marks
total_marks = math_marks + science_marks + english_marks

# 2. Define max_total_marks (assuming each subject has a maximum score of 100)
max_total_marks = 3 * 100  # For 3 subjects, each out of 100

# 3. Calculate percentage
percentage = (total_marks / max_total_marks) * 100

# 4. Print the total_marks and percentage for verification
print(f"\nTotal Marks: {total_marks}")
print(f"Percentage: {percentage:.2f}%")
```

```
Total Marks: 249.0
Percentage: 83.00%
```



```python
# Assume variables are available from previous steps:
# student_name = 'pravalika'
# math_marks = 72.0
# science_marks = 87.0
# english_marks = 90.0
# total_marks = 249.0
# percentage = 83.0
# grade = 'B'

# Display all the results in a clear and well-structured format
print("\n--- Student Grade Report ---")
print(f"Student Name: {student_name}")
print(f"Math Marks: {math_marks}")
print(f"Science Marks: {science_marks}")
print(f"English Marks: {english_marks}")
print(f"Total Marks: {total_marks}")
print(f"Percentage: {percentage:.2f}%")
print(f"Assigned Grade: {grade}")
print("--------------------------")
```

```
--- Student Grade Report ---
Student Name: pravalika
Math Marks: 72.0
Science Marks: 87.0
English Marks: 90.0
Total Marks: 249.0
Percentage: 83.00%
Assigned Grade: B
```

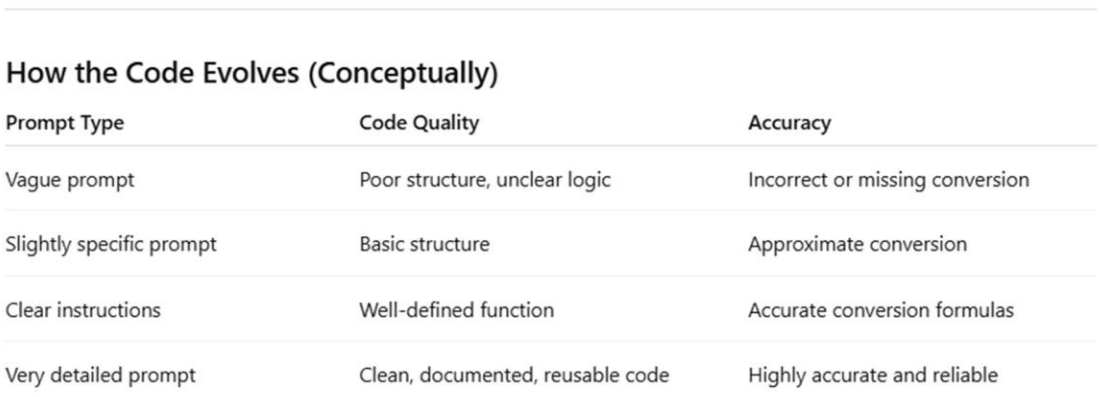
Task Description-5

- Analyzing Prompt Specificity in Unit Conversion Functions: Improving a Unit Conversion Function (Kilometers to Miles and Miles to Kilometers) Using Clear

Instructions.

Expected Output-5

- Analysis of code quality and accuracy differences across multiple prompt variations.



How the Code Evolves (Conceptually)

Prompt Type	Code Quality	Accuracy
Vague prompt	Poor structure, unclear logic	Incorrect or missing conversion
Slightly specific prompt	Basic structure	Approximate conversion
Clear instructions	Well-defined function	Accurate conversion formulas
Very detailed prompt	Clean, documented, reusable code	Highly accurate and reliable

Expected Output-5 requires explaining how clearer prompts lead to better AI-generated unit conversion functions, showing visible improvements in both code quality and accuracy across multiple prompt variations.