Name:Syeda hafsa fathima

batch :39

HT NO:2303A52381

Lab 3: Prompt Engineering – Improving Prompts and Context Management

Lab Objectives:

- To understand how prompt structure and wording influence AI-generated code.
- To explore how context (like comments and function names) helps AI generate relevant output.
- To evaluate the quality and accuracy of code based on prompt clarity.
- To develop effective prompting strategies for AI-assisted programming.

Lab Outcomes (LOs):

After completing this lab, students will be able to:

- Generate Python code using Google Gemini in Google Colab.
- Analyze the effectiveness of code explanations and suggestions by Gemini.
- Set up and use Cursor AI for AI-powered coding assistance.
- Evaluate and refactor code using Cursor AI features.
- Compare AI tool behavior and code quality across different platforms.

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.

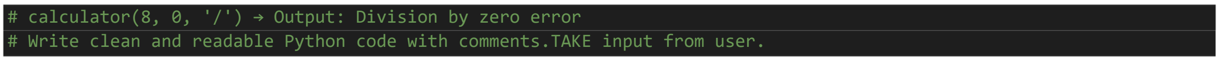
Expected Output-1

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

PROMPT:

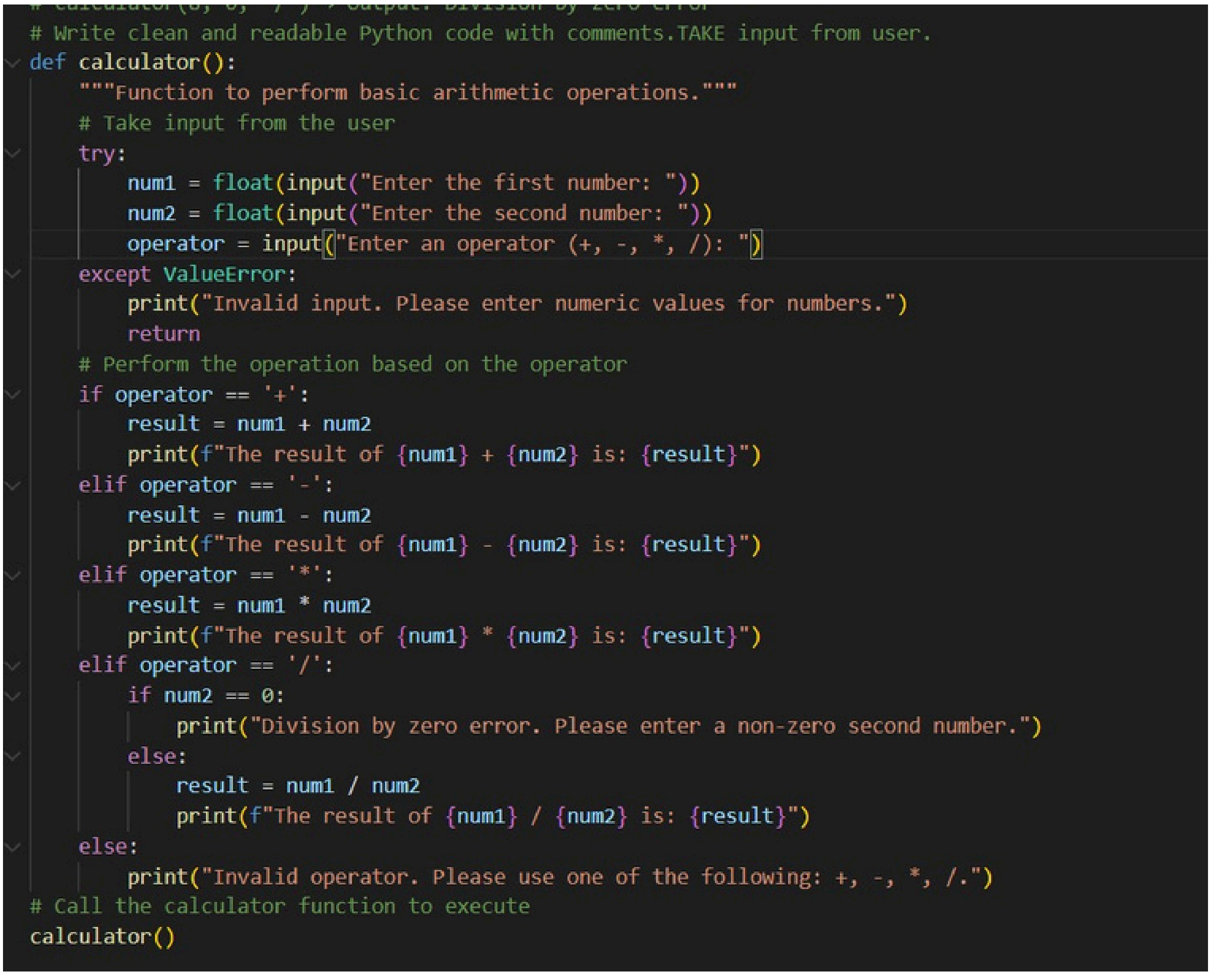
```
#Create a Python function named calculator().The function should perform basic arithmetic operations.
```

```
# Create a Python function named calculator().
# The function should:
# - Take two numbers and an operator as input
# - Support addition, subtraction, multiplication, and division
# - Use if-elif-else logic
# - Handle division by zero
# Add proper comments explaining each step.
# Create a Python function named calculator().
# Requirements:
# - Inputs: two numbers and an operator (+, -, *, /)
# - Perform the selected operation
# - Handle invalid operators and division by zero
# - Print meaningful error messages
# Example usage:
# calculator(10, 5, '+') → Output: 15

```



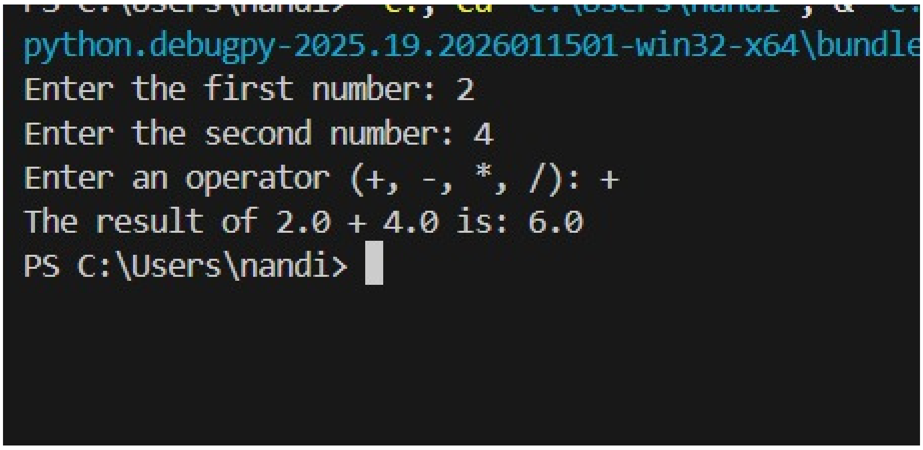
```
# calculator(8, 0, '/') → Output: Division by zero error
# Write clean and readable Python code with comments.TAKE input from user.
```

CODE:



```python
# Write clean and readable Python code with comments.TAKE input from user.
def calculator():
    """Function to perform basic arithmetic operations."""
    # Take input from the user
    try:
        num1 = float(input("Enter the first number: "))
        num2 = float(input("Enter the second number: "))
        operator = input("Enter an operator (+, -, *, /): ")
    except ValueError:
        print("Invalid input. Please enter numeric values for numbers.")
        return
    # Perform the operation based on the operator
    if operator == '+':
        result = num1 + num2
        print(f"The result of {num1} + {num2} is: {result}")
    elif operator == '-':
        result = num1 - num2
        print(f"The result of {num1} - {num2} is: {result}")
    elif operator == '*':
        result = num1 * num2
        print(f"The result of {num1} * {num2} is: {result}")
    elif operator == '/':
        if num2 == 0:
            print("Division by zero error. Please enter a non-zero second number.")
        else:
            result = num1 / num2
            print(f"The result of {num1} / {num2} is: {result}")
    else:
        print("Invalid operator. Please use one of the following: +, -, *, /.")
# Call the calculator function to execute
calculator()
```

OUTPUT:



```
python.debugpy-2025.19.2026011501-win32-x64\bundle
Enter the first number: 2
Enter the second number: 4
Enter an operator (+, -, *, /): +
The result of 2.0 + 4.0 is: 6.0
PS C:\Users\nandi>
```

ANALYSIS:

As prompt details increased, the AI-generated calculator improved in structure, logic clarity, and error handling. Progressive prompting helped produce more reliable and readable code.

Task Description-2

•RefiningPromptsforSorting Logic: Start with a vague prompt for sorting student marks,

then refine it to clearly specify sorting order and constraints.

Expected Output-2

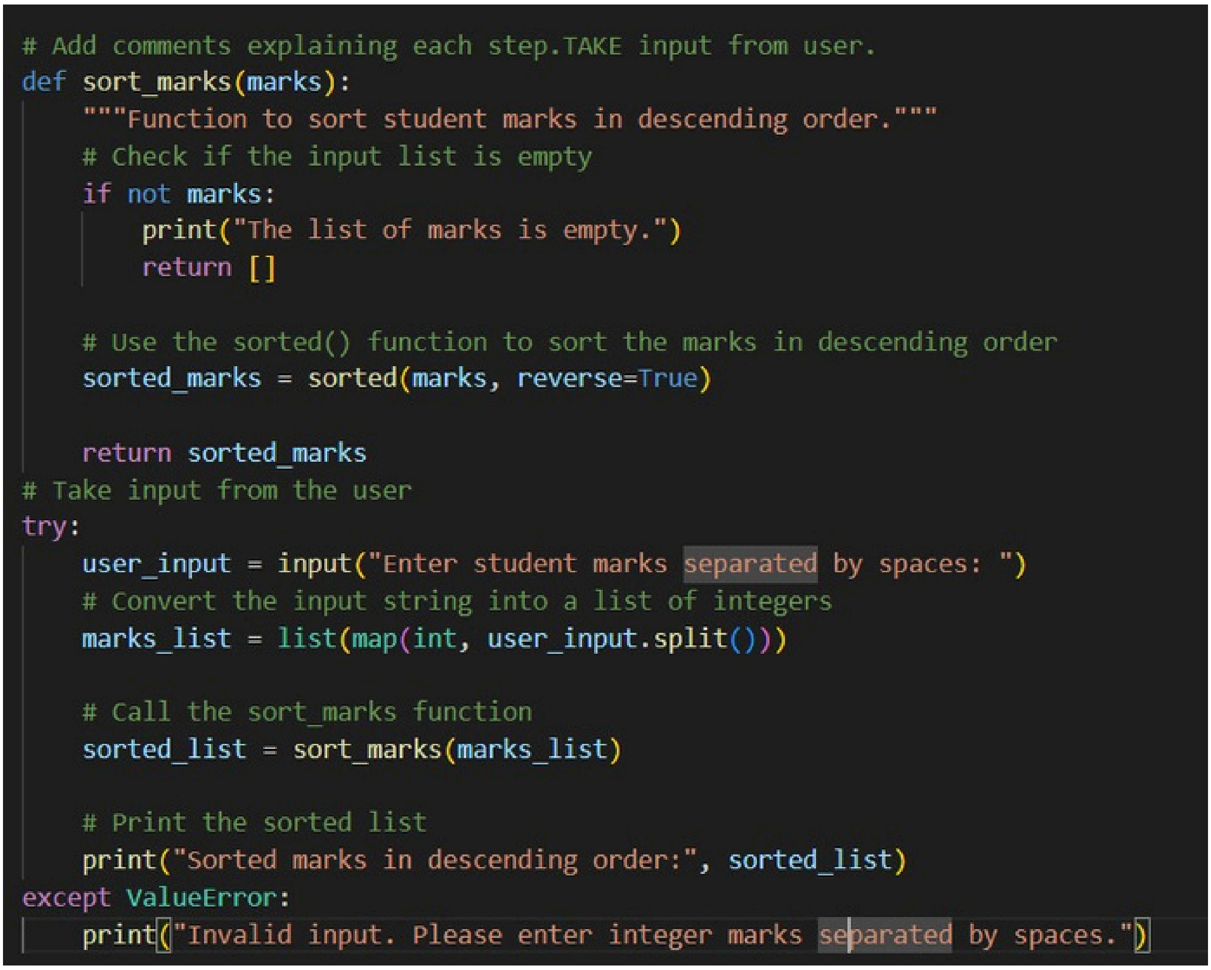
• AI-generated sorting function evolves from ambiguous logic to an accurate and efficient implementation.

PROMPT:

```
# Write a Python function to sort student marks.
# Write a Python function named sort_marks().
```

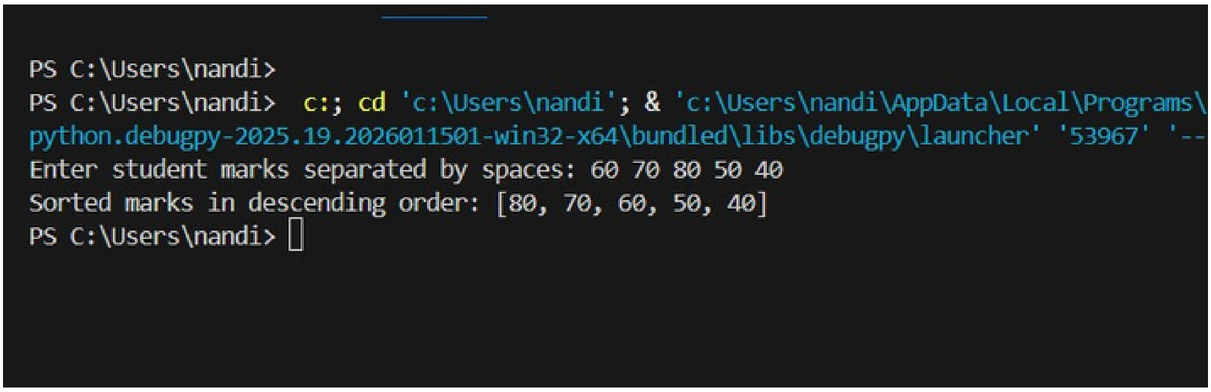
```



# The function should:
# - Take a list of student marks as input
# - Sort the marks in ascending order
# - Return the sorted list

# Use Python built-in sorting methods.
# Write a Python function named sort_marks().

# Requirements:
# - Input: list of integer student marks
# - Sort marks in descending order
# - Do not modify the original list
# - Handle empty lists
# - Return the sorted list

# Example:
# Input: [78, 90, 45, 88]
# Output: [90, 88, 78, 45]

# Add comments explaining each step.TAKE input from user.
```

CODE:



```python
# Add comments explaining each step.TAKE input from user.
def sort_marks(marks):
    """Function to sort student marks in descending order."""
    # Check if the input list is empty
    if not marks:
        print("The list of marks is empty.")
        return []

    # Use the sorted() function to sort the marks in descending order
    sorted_marks = sorted(marks, reverse=True)

    return sorted_marks
# Take input from the user
try:
    user_input = input("Enter student marks separated by spaces: ")
    # Convert the input string into a list of integers
    marks_list = list(map(int, user_input.split()))

    # Call the sort_marks function
    sorted_list = sort_marks(marks_list)

    # Print the sorted list
    print("Sorted marks in descending order:", sorted_list)
except ValueError:
    print("Invalid input. Please enter integer marks separated by spaces.")
```

OUTPUT:



```
PS C:\Users\nandi>
PS C:\Users\nandi>  c:; cd 'c:\Users\nandi'; & 'c:\Users\nandi\AppData\Local\Programs\
python.debugpy-2025.19.2026011501-win32-x64\bundled\libs\debugpy\launcher' '53967' '--
Enter student marks separated by spaces: 60 70 80 50 40
Sorted marks in descending order: [80, 70, 60, 50, 40]
PS C:\Users\nandi> 
```

ANALYSIS:

Vague prompts resulted in ambiguous sorting logic, while refined prompts clearly defined order and constraints. This led to accurate and efficient sorting implementations.

Task Description-3

• Few-Shot Prompting for Prime Number Validation: Provide multiple input-output

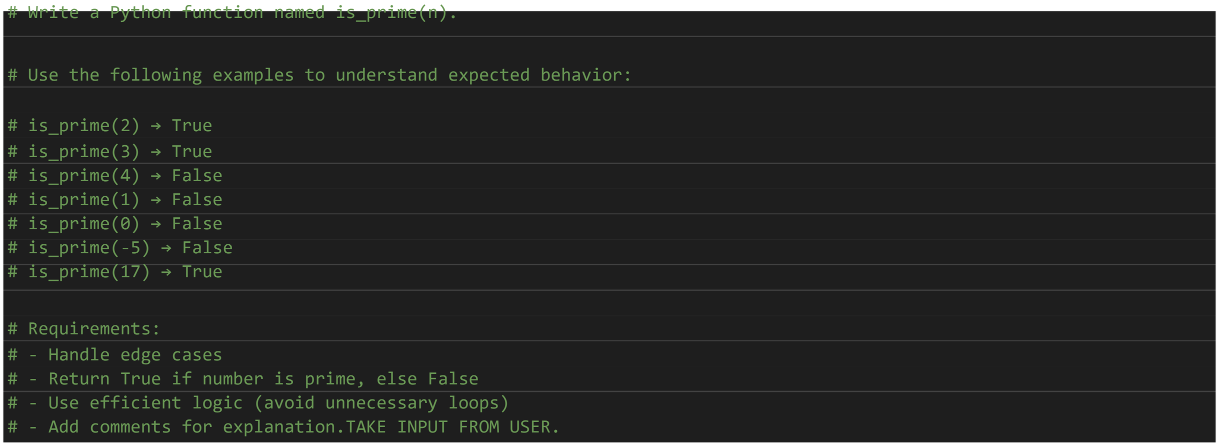
examples for a function that checks whether a number is prime. Observe how few-shot

prompting improves correctness.

Expected Output-3

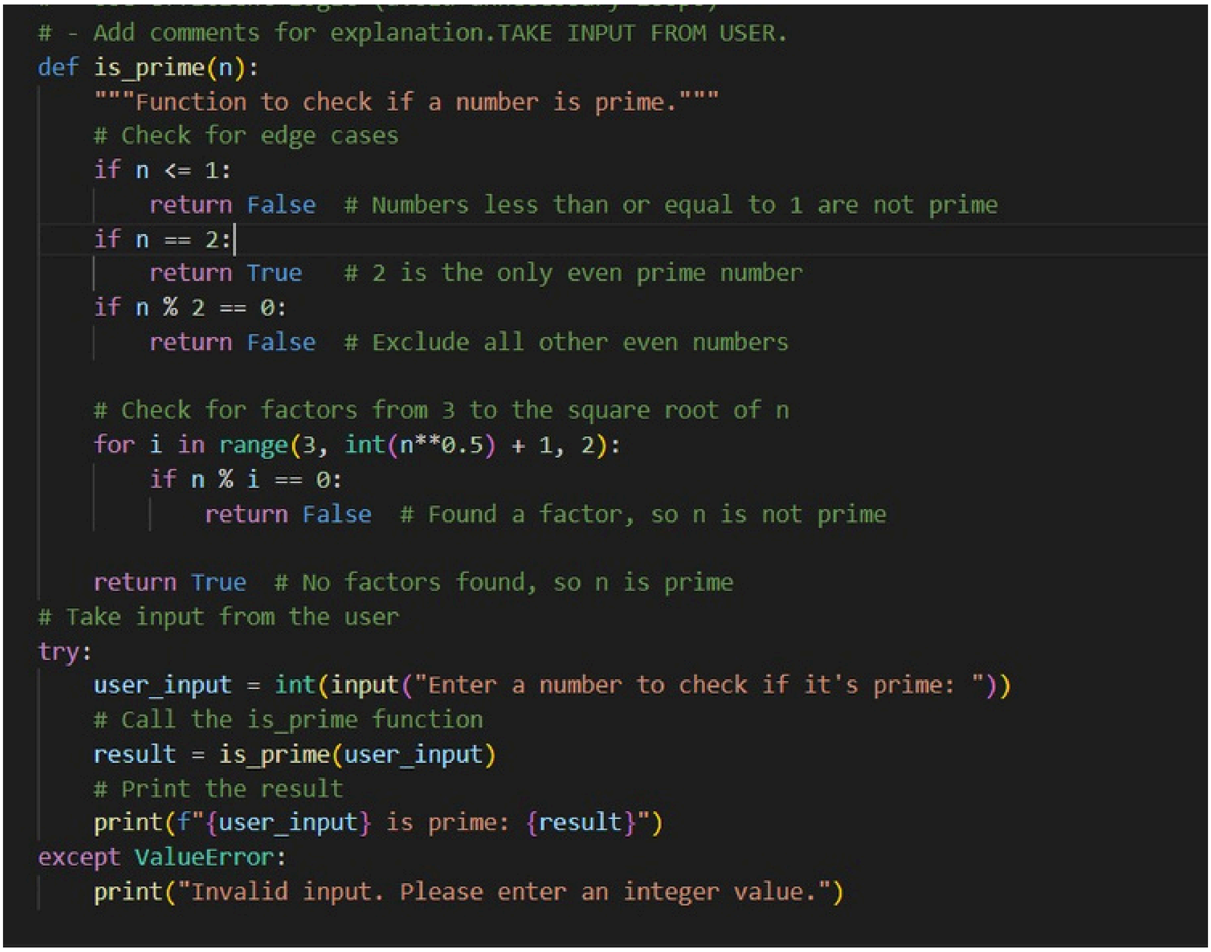
• Improved prime-checking function with better edge-case handling.

PROMPT:



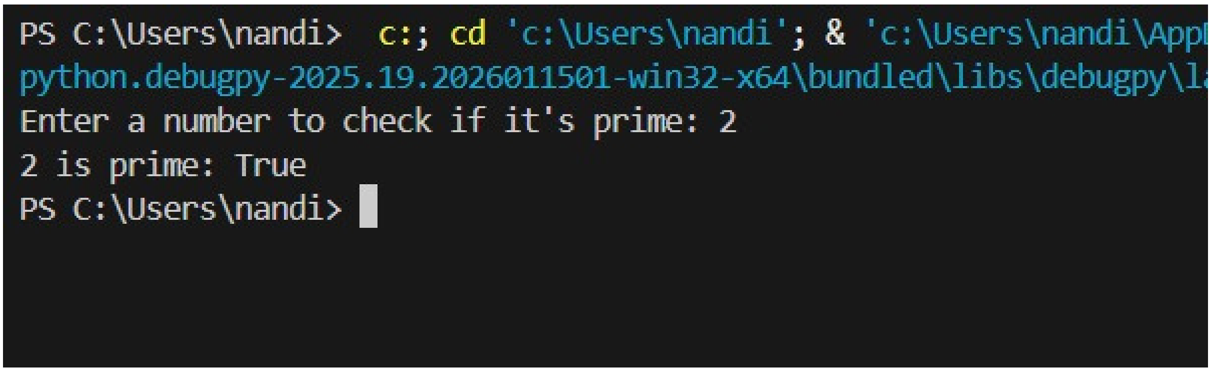
```
# Write a Python function named is_prime(n).

# Use the following examples to understand expected behavior:

# is_prime(2) → True
# is_prime(3) → True
# is_prime(4) → False
# is_prime(1) → False
# is_prime(0) → False
# is_prime(-5) → False
# is_prime(17) → True

# Requirements:
# - Handle edge cases
# - Return True if number is prime, else False
# - Use efficient logic (avoid unnecessary loops)
# - Add comments for explanation.TAKE INPUT FROM USER.
```

CODE:



```
# - Add comments for explanation.TAKE INPUT FROM USER.
def is_prime(n):
    """Function to check if a number is prime."""
    # Check for edge cases
    if n <= 1:
        return False  # Numbers less than or equal to 1 are not prime
    if n == 2:
        return True   # 2 is the only even prime number
    if n % 2 == 0:
        return False  # Exclude all other even numbers

    # Check for factors from 3 to the square root of n
    for i in range(3, int(n**0.5) + 1, 2):
        if n % i == 0:
            return False  # Found a factor, so n is not prime

    return True  # No factors found, so n is prime
# Take input from the user
try:
    user_input = int(input("Enter a number to check if it's prime: "))
    # Call the is_prime function
    result = is_prime(user_input)
    # Print the result
    print(f"{user_input} is prime: {result}")
except ValueError:
    print("Invalid input. Please enter an integer value.")
```

OUTPUT:



```
PS C:\Users\nandi>  c:; cd 'c:\Users\nandi'; & 'c:\Users\nandi\App
python.debugpy-2025.19.2026011501-win32-x64\bundled\libs\debugpy\l
Enter a number to check if it's prime: 2
2 is prime: True
PS C:\Users\nandi> 
```

ANALYSIS:

Providing input-output examples significantly improved prime number validation and edge-case handling. Few-shot prompting guided the AI toward correct logical behavior.

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.

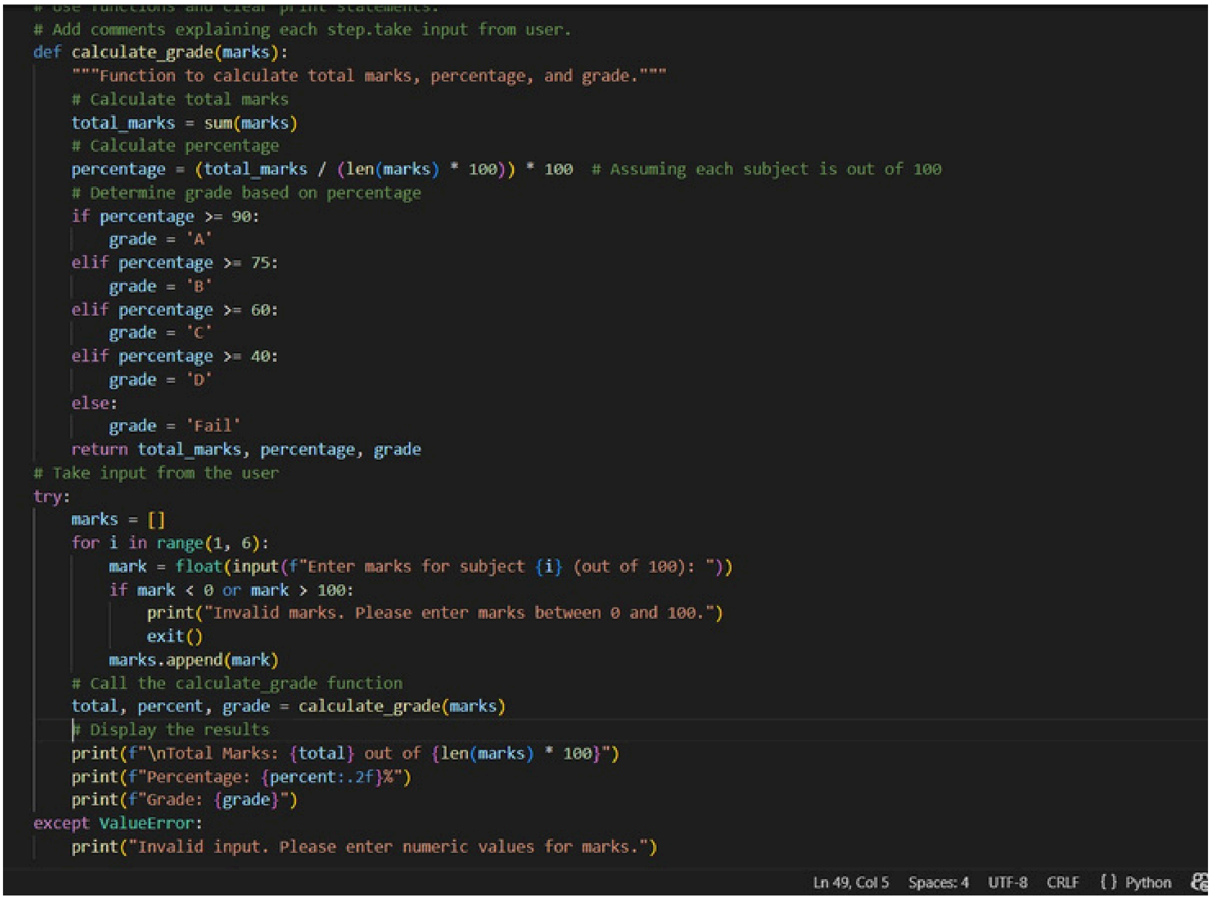
Expected Output-4

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

PROMPT:

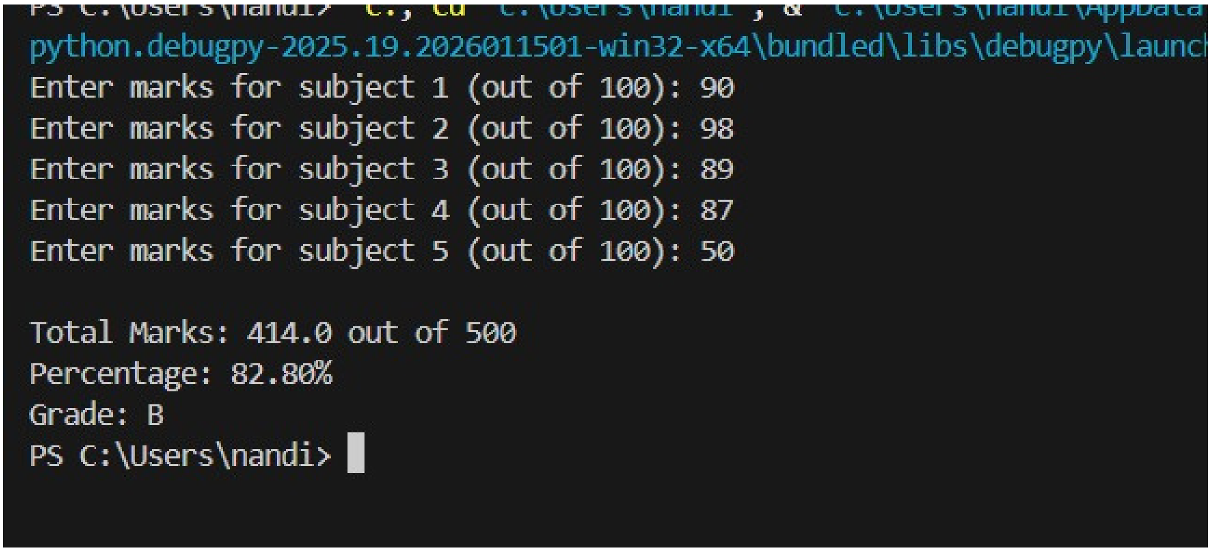
```
# Create a Python console-based student grading system.
# Requires:
# - Ask the user to enter marks for 5 subjects
# - Calculate total marks
# - Calculate percentage
# - Assign grade based on percentage:


#     * ≥ 90 → A * ≥
#     75 → B * ≥ 60
#     → C * ≥ 40 → D
#     * < 40 → Fail
#


# Display:
# - Total marks
# - Percentage
# - Grade

# Use functions and clear print statements.
# Add comments explaining each step.take input from user.
```

CODE:



```
# use functions and clear print statements.
# Add comments explaining each step.take input from user.
def calculate_grade(marks):
    """Function to calculate total marks, percentage, and grade."""
    # Calculate total marks
    total_marks = sum(marks)
    # Calculate percentage
    percentage = (total_marks / (len(marks) * 100)) * 100  # Assuming each subject is out of 100
    # Determine grade based on percentage
    if percentage >= 90:
        grade = 'A'
    elif percentage >= 75:
        grade = 'B'
    elif percentage >= 60:
        grade = 'C'
    elif percentage >= 40:
        grade = 'D'
    else:
        grade = 'Fail'
    return total_marks, percentage, grade
# Take input from the user
try:
    marks = []
    for i in range(1, 6):
        mark = float(input(f"Enter marks for subject {i} (out of 100): "))
        if mark < 0 or mark > 100:
            print("Invalid marks. Please enter marks between 0 and 100.")
            exit()
        marks.append(mark)
    # Call the calculate_grade function
    total, percent, grade = calculate_grade(marks)
    # Display the results
    print(f"\nTotal Marks: {total} out of {len(marks) * 100}")
    print(f"Percentage: {percent:.2f}%")
    print(f"Grade: {grade}")
except ValueError:
    print("Invalid input. Please enter numeric values for marks.")
```

OUTPUT:



```
PS C:\Users\nandi> c:; cd c:\Users\nandi ; & c:\Users\nandi\AppData
python.debugpy-2025.19.2026011501-win32-x64\bundled\libs\debugpy\launch
Enter marks for subject 1 (out of 100): 90
Enter marks for subject 2 (out of 100): 98
Enter marks for subject 3 (out of 100): 89
Enter marks for subject 4 (out of 100): 87
Enter marks for subject 5 (out of 100): 50

Total Marks: 414.0 out of 500
Percentage: 82.80%
Grade: B
PS C:\Users\nandi> 
```

ANALYSIS:

Detailed prompts enabled the AI to generate structured, user-friendly grading system code with accurate calculations. Clear instructions improved output clarity and usability.

Task Description-5

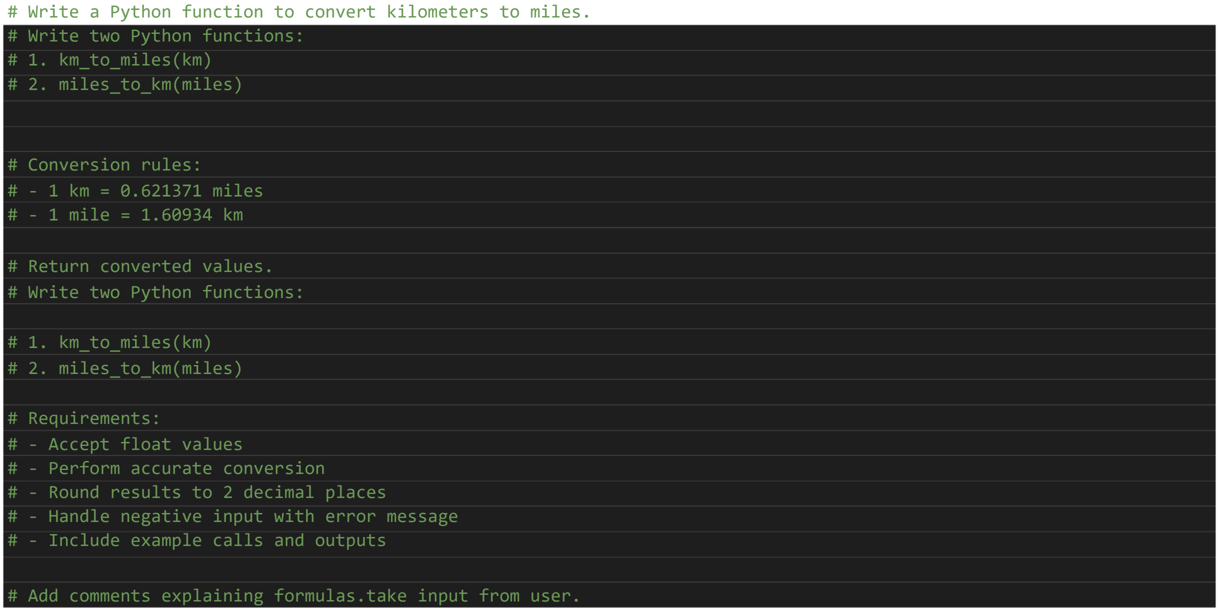
- ***Analyzing Prompt Specificity in Unit Conversion Functions: Improving a Unit***

Conversion Function (Kilometers to Miles and Miles to Kilometers) Using Clear Instructions.

Expected Output-5

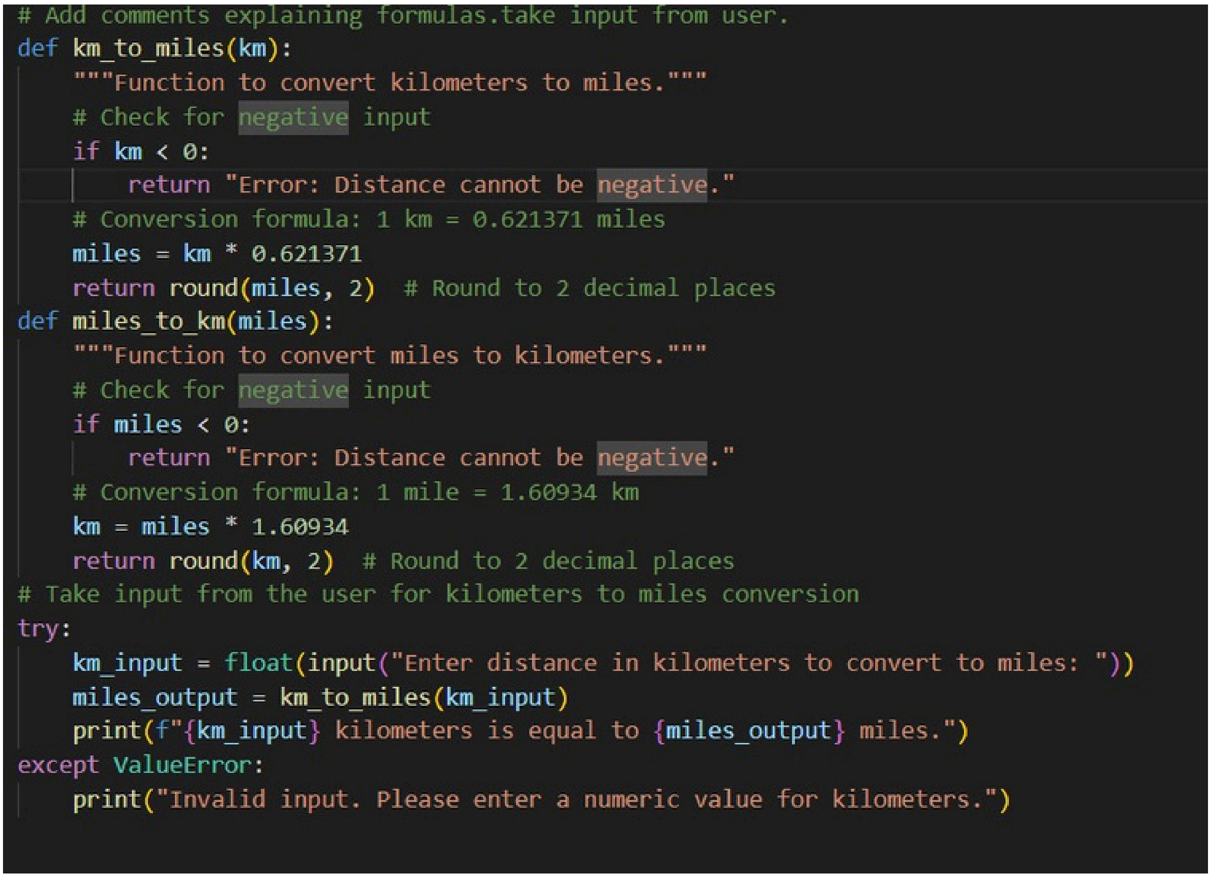
- ***Analysis of code quality and accuracy differences across multiple prompt variations.***

PROMPT:



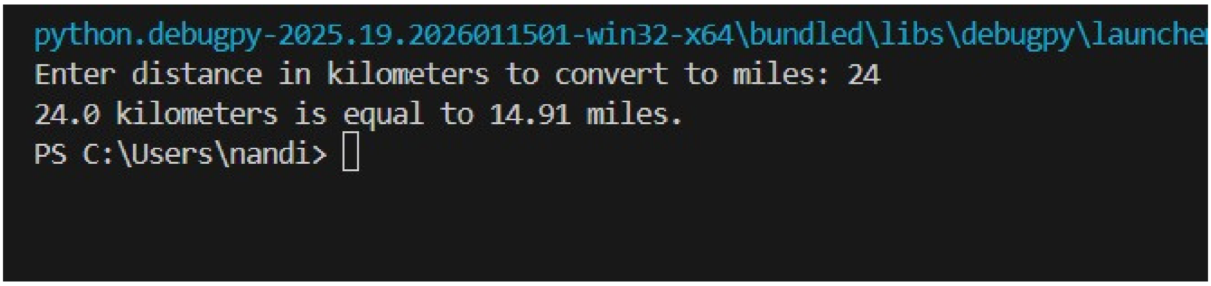
```
# Write a Python function to convert kilometers to miles.
# Write two Python functions:
# 1. km_to_miles(km)
# 2. miles_to_km(miles)


# Conversion rules:
# - 1 km = 0.621371 miles
# - 1 mile = 1.60934 km

# Return converted values.
# Write two Python functions:

# 1. km_to_miles(km)
# 2. miles_to_km(miles)

# Requirements:
# - Accept float values
# - Perform accurate conversion
# - Round results to 2 decimal places
# - Handle negative input with error message
# - Include example calls and outputs

# Add comments explaining formulas.take input from user.
```

CODE:



```python
# Add comments explaining formulas.take input from user.
def km_to_miles(km):
    """Function to convert kilometers to miles."""
    # Check for negative input
    if km < 0:
        return "Error: Distance cannot be negative."
    # Conversion formula: 1 km = 0.621371 miles
    miles = km * 0.621371
    return round(miles, 2)  # Round to 2 decimal places
def miles_to_km(miles):
    """Function to convert miles to kilometers."""
    # Check for negative input
    if miles < 0:
        return "Error: Distance cannot be negative."
    # Conversion formula: 1 mile = 1.60934 km
    km = miles * 1.60934
    return round(km, 2)  # Round to 2 decimal places
# Take input from the user for kilometers to miles conversion
try:
    km_input = float(input("Enter distance in kilometers to convert to miles: "))
    miles_output = km_to_miles(km_input)
    print(f"{km_input} kilometers is equal to {miles_output} miles.")
except ValueError:
    print("Invalid input. Please enter a numeric value for kilometers.")
```

OUTPUT:



```
python.debugpy-2025.19.2026011501-win32-x64\bundled\libs\debugpy\launche
Enter distance in kilometers to convert to miles: 24
24.0 kilometers is equal to 14.91 miles.
PS C:\Users\nandi> 
```

ANALYSIS:

Highly specific prompts improved unit conversion accuracy, validation, and rounding. Clear instructions resulted in more robust and professional-quality code.