

Varshini Girugula

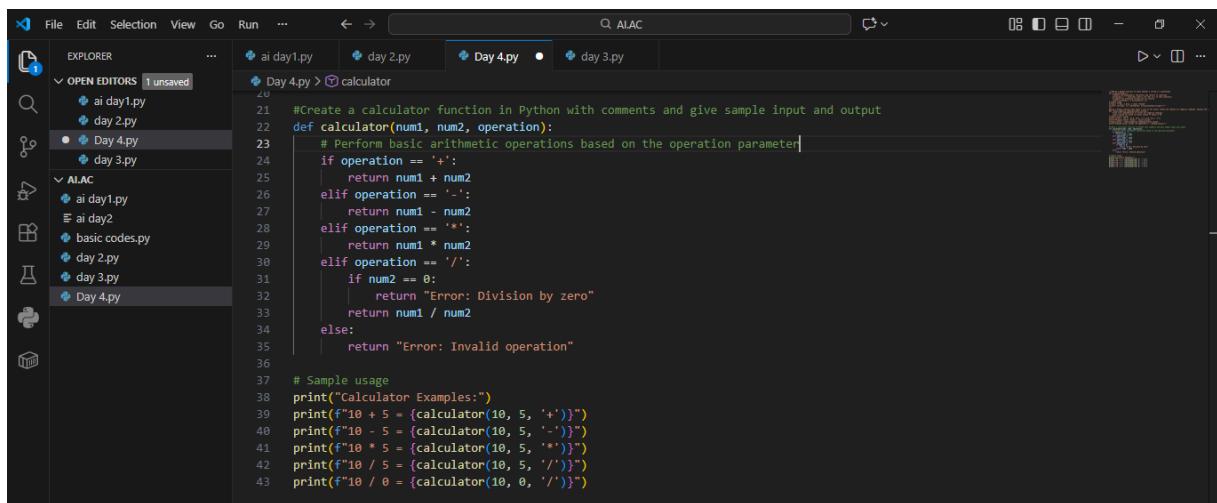
2403A51L14

B-51

## ASSIGNMENT -3.2

### Task 1: Progressive Prompting – Calculator Design

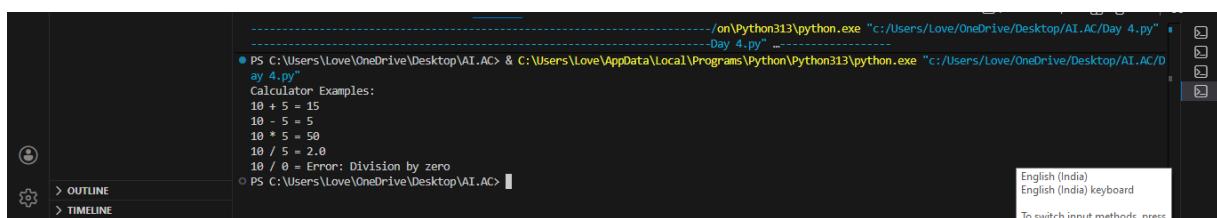
**PROMPT:** Create a calculator function in Python with comments and give sample input and output.



```
#Create a calculator function in Python with comments and give sample input and output
def calculator(num1, num2, operation):
    # Perform basic arithmetic operations based on the operation parameter
    if operation == '+':
        return num1 + num2
    elif operation == '-':
        return num1 - num2
    elif operation == '*':
        return num1 * num2
    elif operation == '/':
        if num2 == 0:
            return "Error: Division by zero"
        return num1 / num2
    else:
        return "Error: Invalid operation"

# Sample usage
print("Calculator Examples:")
print("10 + 5 = {calculator(10, 5, '+')}")
print("10 - 5 = {calculator(10, 5, '-')}")
print("10 * 5 = {calculator(10, 5, '*')}")
print("10 / 5 = {calculator(10, 5, '/')}")
print("10 / 0 = {calculator(10, 0, '/')}")
```

### OUTPUT:



```
PS C:\Users\Love\OneDrive\Desktop\AI.AC> & C:\Users\Love\AppData\Local\Programs\Python\Python313\python.exe "c:/Users/Love/OneDrive/Desktop/AI.AC/Day 4.py"
Calculator Examples:
10 + 5 = 15
10 - 5 = 5
10 * 5 = 50
10 / 5 = 2.0
10 / 0 = Error: Division by zero
PS C:\Users\Love\OneDrive\Desktop\AI.AC>
```

### EXPLANATION:

When we give only a function name, the AI generates very basic or incomplete code.

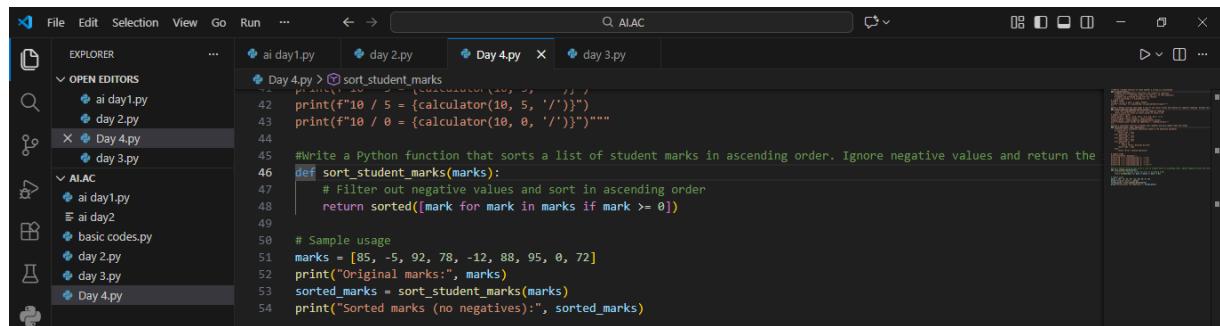
As we gradually add comments, requirements, and examples, the AI understands better and produces:

- Proper logic ,Error handling , Cleaner structure

This shows that well-defined prompts lead to better AI-generated programs.

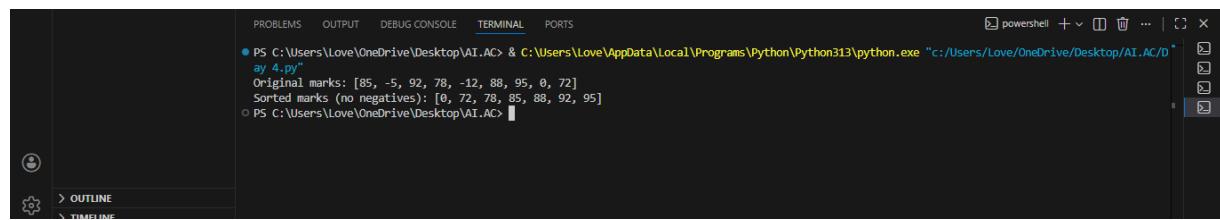
## Task 2: Refining Prompts – Sorting Student Marks

**PROMPT:** Write a Python function that sorts a list of student marks in ascending order. Ignore negative values and return the sorted list using efficient logic.



```
Day 4.py > sort_student_marks
1 print("10 / 5 = {calculator(10, 5, '/')}")
2 print(f"10 / 0 = {calculator(10, 0, '/')}"")
3
4 #Write a Python function that sorts a list of student marks in ascending order. Ignore negative values and return the
5 def sort_student_marks(marks):
6     # Filter out negative values and sort in ascending order
7     return sorted([mark for mark in marks if mark >= 0])
8
9 # Sample usage
10 marks = [85, -5, 92, 78, -12, 88, 95, 0, 72]
11 print("Original marks:", marks)
12 sorted_marks = sort_student_marks(marks)
13 print("Sorted marks (no negatives):", sorted_marks)
```

## OUTPUT:



```
ps C:\Users\Love\OneDrive\Desktop\AI.AC & C:\Users\Love\AppData\Local\Programs\Python\Python313\python.exe "c:/Users/Love/OneDrive/Desktop/AI.AC/D
ay 4.py"
Original marks: [85, -5, 92, 78, -12, 88, 95, 0, 72]
Sorted marks (no negatives): [85, 72, 78, 88, 92, 95]
```

## EXPLANATION:

This task demonstrates how **vague prompts cause ambiguous results**. Initially, the AI may not know:

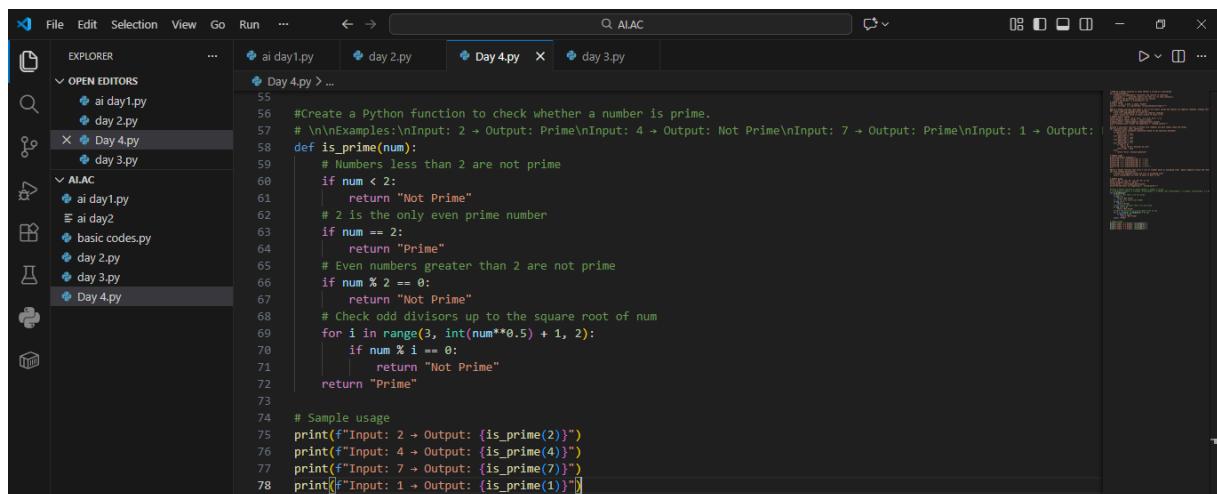
- Sorting order , Data constraints , Output format

By refining the prompt, we guide the AI to generate **accurate and efficient sorting logic**.

This highlights the importance of **specific instructions in prompt engineering**.

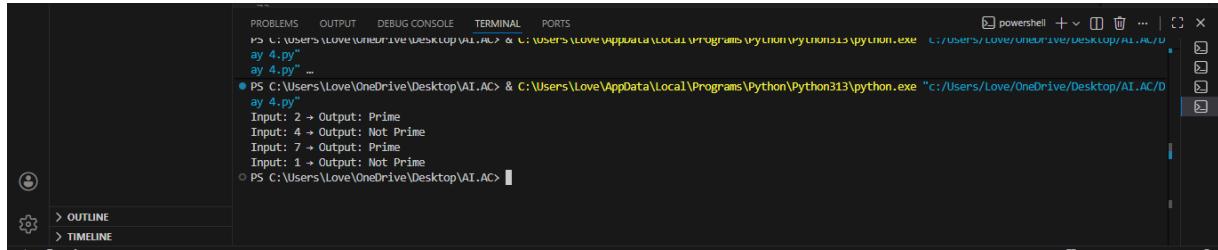
## Task 3: Few-Shot Prompting – Prime Number Validation

**Prompt:** Create a Python function to check whether a number is prime. Examples:  
Input: 2 → Output: Prime  
Input: 4 → Output: Not Prime  
Input: 7 . Use these examples to design the logic



```
55 #Create a Python function to check whether a number is prime.
56 # \nExamples:\nInput: 2 -> Output: Prime\nInput: 4 -> Output: Not Prime\nInput: 7 -> Output: Prime\nInput: 1 -> Output:
57
58 def is_prime(num):
59     # Numbers less than 2 are not prime
60     if num < 2:
61         return "Not Prime"
62     # 2 is the only even prime number
63     if num == 2:
64         return "Prime"
65     # Even numbers greater than 2 are not prime
66     if num % 2 == 0:
67         return "Not Prime"
68     # Check odd divisors up to the square root of num
69     for i in range(3, int(num**0.5) + 1, 2):
70         if num % i == 0:
71             return "Not Prime"
72     return "Prime"
73
74 # Sample usage
75 print(f"Input: 2 -> Output: {is_prime(2)}")
76 print(f"Input: 4 -> Output: {is_prime(4)}")
77 print(f"Input: 7 -> Output: {is_prime(7)}")
78 print(f"Input: 1 -> Output: {is_prime(1)}")
```

## OUTPUT:



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
powershell + v ⌂ ⌂ ... | x
rs C:\Users\Love\OneDrive\Desktop\AI.AC & C:\Users\Love\AppData\Local\Programs\Python\Python313\python.exe "C:/Users/Love/OneDrive/Desktop/AI.AC/D
ay 4.py"
ay 4.py" ...
● ps C:\Users\Love\OneDrive\Desktop\AI.AC & C:\Users\Love\AppData\Local\Programs\Python\Python313\python.exe "C:/Users/Love/OneDrive/Desktop/AI.AC/D
ay 4.py"
Input: 2 → Output: Prime
Input: 7 → Output: Prime
Input: 1 → Output: Not Prime
○ ps C:\Users\Love\OneDrive\Desktop\AI.AC
```

## EXPLANATION:

Few-shot prompting means providing **example inputs and outputs** along with the prompt.

This helps the AI:

- Understand edge cases , Improve accuracy , Avoid logical mistakes

Compared to a simple prompt, few-shot prompting results in **more reliable prime-checking logic**.

## Task 4: Prompt-Guided UI Design – Student Grading System

**Prompt :** Create a Python program with a simple user interface that accepts student marks, calculates total marks, percentage, and grade, and displays the result.

```

80 #Create a Python program with a simple user interface that accepts student marks, calculates total marks, percentage, and grade
81 def calculate_grade(marks):
82     # Calculate total marks
83     total_marks = sum(marks)
84     # Calculate percentage
85     percentage = (total_marks / (len(marks) * 100)) * 100
86     # Determine grade based on percentage
87     if percentage >= 90:
88         grade = 'A'
89     elif percentage >= 80:
90         grade = 'B'
91     elif percentage >= 70:
92         grade = 'C'
93     elif percentage >= 60:
94         grade = 'D'
95     else:
96         grade = 'F'
97     return total_marks, percentage, grade
98 # User interface to accept student marks
99 def main():
100     print("Student Marks Calculator")
101     marks = []
102     num_subjects = int(input("Enter the number of subjects: "))
103     for i in range(num_subjects):
104         mark = float(input(f"Enter marks for subject {i + 1} (out of 100): "))
105         marks.append(mark)
106     total, percentage, grade = calculate_grade(marks)
107     print(f"\nTotal Marks: {total}")
108     print(f"Percentage: {percentage:.2f}%")
109     print(f"Grade: {grade}")
110 if __name__ == "__main__":
111     main()

```

14 Januver 2026

## Output:

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
powershell + ×
Student Marks Calculator
Enter the number of subjects: 5
Enter marks for subject 1 (out of 100): 50
Enter marks for subject 2 (out of 100): 60
Enter marks for subject 3 (out of 100): 70
Enter marks for subject 4 (out of 100): 90
Enter marks for subject 5 (out of 100): 80

Total Marks: 350.0
Percentage: 70.00%
Grade: C
PS C:\Users\Love\OneDrive\Desktop\AI.AC>

```

## Explanation:

This task focuses on using prompts to guide program structure and user interaction.

Instead of a graphical UI, a console-based UI is used for:

- Simplicity , Code compatibility , Clear user interaction

## Task 5: Prompt Specificity – Unit Conversion Function

## Prompt:

Write two Python functions:  
n1. Convert kilometers to miles  
n2.

Convert miles to kilometers  
Use correct formulas, show sample input/output, and add comments explaining the logic

The screenshot shows the Microsoft Visual Studio Code (VS Code) interface. The title bar displays "File Edit Selection View Go Run ... ALAC". The left sidebar has icons for Explorer, Open Editors, and AI Assistant. The "OPEN EDITORS" section shows "1 unsaved" file: "Day 4.py". The main editor area contains Python code for converting kilometers to miles and vice versa:

```
ai day1.py ai day2.py Day 4.py ● day 3.py

Day 4.py > ...
199     print(f"Grade: {grade}")
110 if __name__ == "__main__":
111     main()
112
113 #Write two Python functions:\n1. Convert kilometers to miles\n2. Convert miles to kilometers\nUse correct formulas, show so
114 def km_to_miles(km):
115     # Conversion factor from kilometers to miles
116     conversion_factor = 0.621371
117     # Convert kilometers to miles
118     miles = km * conversion_factor
119     return miles
120
121 def miles_to_km(miles):
122     # Conversion factor from miles to kilometers
123     conversion_factor = 1.60934
124     # Convert miles to kilometers
125     km = miles * conversion_factor
126     return km
127
128 # Sample usage
129 print("Kilometers to Miles Conversion:")
130 km_input = 10
131 print(f'{km_input} kilometers is equal to {km_to_miles(km_input):.2f} miles')
132 print("\nMiles to Kilometers Conversion:")
133 miles_input = 10
134 print(f'{miles_input} miles is equal to {miles_to_km(miles_input):.2f} kilometers')
```

## Output:

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS powershell + ⌂ ⌂ X

● Grade: C
PS C:\Users\Love\OneDrive\Desktop\AI.AC> ^C
PS C:\Users\Love\OneDrive\Desktop\AI.AC> & C:\Users\Love\AppData\Local\Programs\Python\Python313\python.exe "c:/Users/Love/OneDrive/Desktop/AI.AC/Day 4.py"
Kilometers to Miles Conversion:
10 kilometers is equal to 6.21 miles

○ Miles to Kilometers Conversion:
10 miles is equal to 16.09 kilometers
PS C:\Users\Love\OneDrive\Desktop\AI.AC>
```

## Explanation:

This task highlights how clear and specific prompts improve code accuracy.

A vague prompt may produce incomplete or incorrect conversions.

When formulas and requirements are clearly stated, the AI generates:

- Accurate calculations, Reusable functions, Well-documented code

This proves that **prompt specificity directly affects output quality**.