

Amruth Sagar . Vemuganti

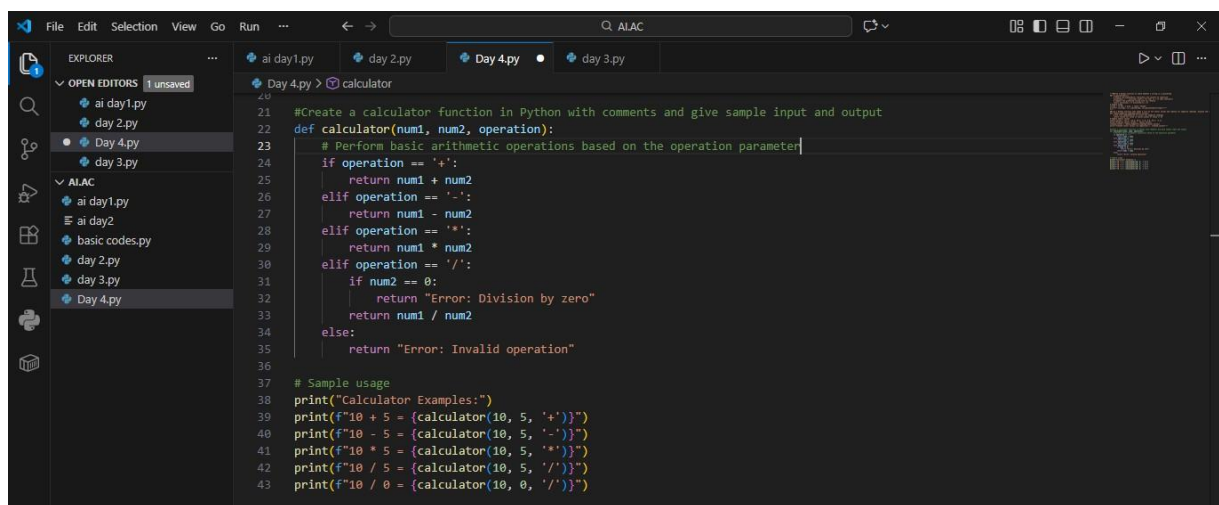
2403A51L44

B-52

ASSIGNMENT -3.2

Task 1: Progressive Prompting – Calculator Design

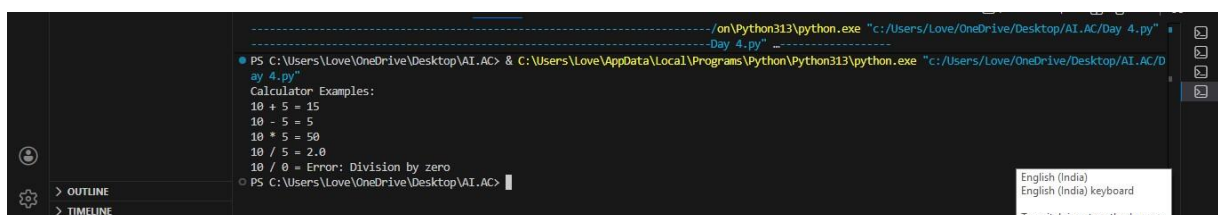
PROMPT: Create a calculator function in Python with comments and give sample input and output.



The screenshot shows a code editor with a file explorer on the left and a code editor on the right. The file explorer shows a project named 'AIAC' with files 'ai day1.py', 'day 2.py', 'Day 4.py', and 'day 3.py'. The code editor shows the following Python code:

```
21 #Create a calculator function in Python with comments and give sample input and output
22 def calculator(num1, num2, operation):
23     # Perform basic arithmetic operations based on the operation parameter
24     if operation == '+':
25         return num1 + num2
26     elif operation == '-':
27         return num1 - num2
28     elif operation == '*':
29         return num1 * num2
30     elif operation == '/':
31         if num2 == 0:
32             return "Error: Division by zero"
33         return num1 / num2
34     else:
35         return "Error: Invalid operation"
36
37 # Sample usage
38 print("Calculator Examples:")
39 print(f"10 + 5 = {calculator(10, 5, '+')}")
40 print(f"10 - 5 = {calculator(10, 5, '-')}")
41 print(f"10 * 5 = {calculator(10, 5, '*')}")
42 print(f"10 / 5 = {calculator(10, 5, '/')}")
43 print(f"10 / 0 = {calculator(10, 0, '/')}")
```

OUTPUT:



The screenshot shows a terminal window with the following 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
```

EXPLANATION:

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

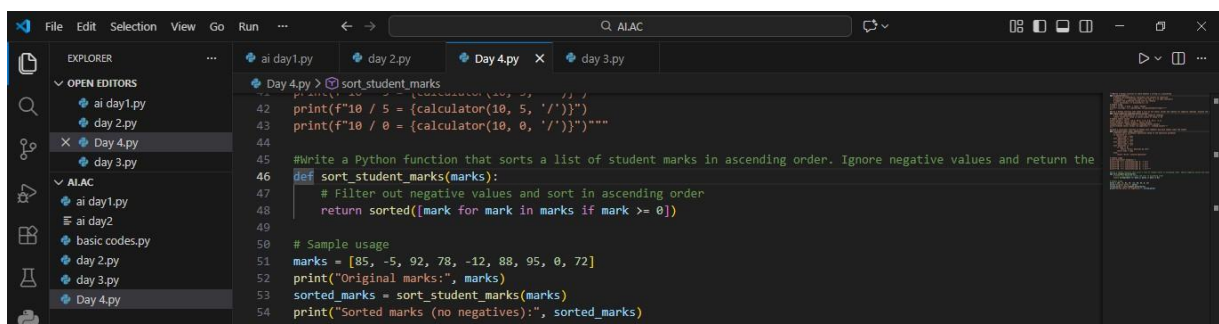
As we gradually add comments, requirements, and examples, the AI understands better and produces:

- 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.



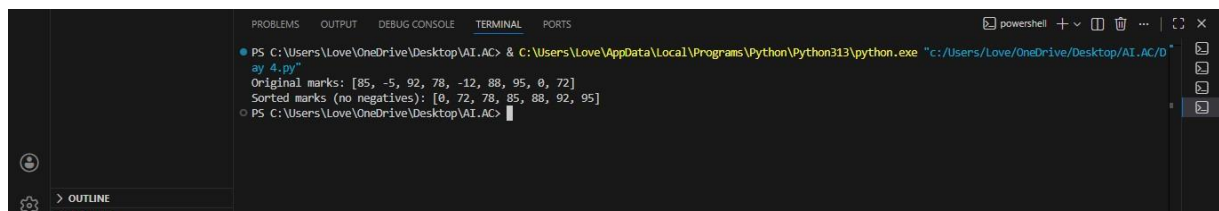
The screenshot shows a code editor with a file explorer on the left and a code editor on the right. The file explorer shows a project named 'AIAC' with several files: 'ai day1.py', 'day 2.py', 'Day 4.py', and 'day 3.py'. The code editor shows the content of 'Day 4.py', which contains a Python function 'sort_student_marks' that sorts a list of marks in ascending order, ignoring negative values. The function is defined as follows:

```
def sort_student_marks(marks):  
    # Filter out negative values and sort in ascending order  
    return sorted([mark for mark in marks if mark >= 0])
```

The code also includes a sample usage example:

```
# Sample usage  
marks = [85, -5, 92, 78, -12, 88, 95, 0, 72]  
print("Original marks:", marks)  
sorted_marks = sort_student_marks(marks)  
print("Sorted marks (no negatives):", sorted_marks)
```

OUTPUT:



The screenshot shows a terminal window with the following 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"  
Original marks: [85, -5, 92, 78, -12, 88, 95, 0, 72]  
Sorted marks (no negatives): [0, 72, 78, 85, 88, 92, 95]  
PS C:\Users\Love\OneDrive\Desktop\AI.AC>
```

EXPLANATION:

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

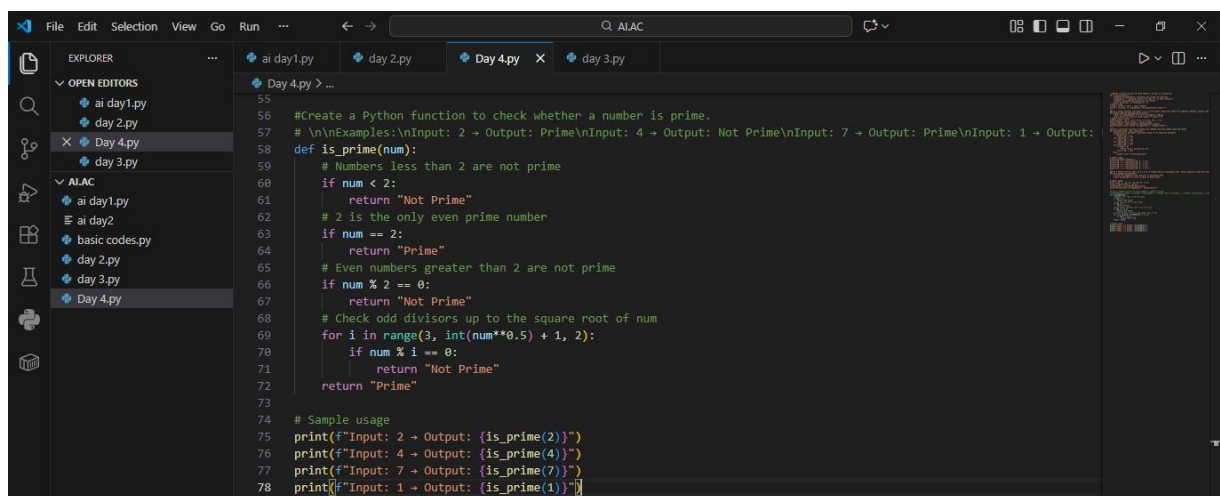
- Sorting order , Data constraints , Output format

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**.

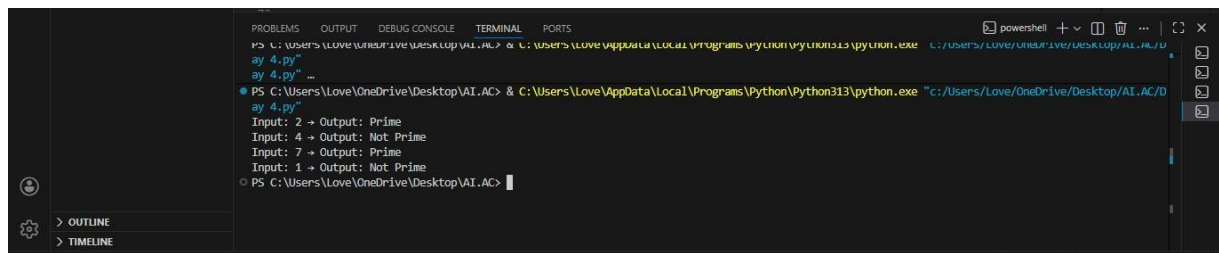
Task 3: Few-Shot Prompting – Prime Number Validation

Prompt: Create a Python function to check whether a number is prime. Examples:\n Input: 2 → Output: Prime\n Input: 4 → Output: Not Prime\n Input: 7 → Output: Prime\n Input: 1 → Output: Not Prime . Use these examples to design the logic



```
55
56 #Create a Python function to check whether a number is prime.
57 # \n\nExamples:\nInput: 2 → Output: Prime\nInput: 4 → Output: Not Prime\nInput: 7 → Output: Prime\nInput: 1 → Output:
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:



```
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" -
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: 4 -> Output: Not Prime
Input: 7 -> Output: Prime
Input: 1 -> Output: Not Prime
PS C:\Users\Love\OneDrive\Desktop\AI.AC>
```

EXPLANATION:

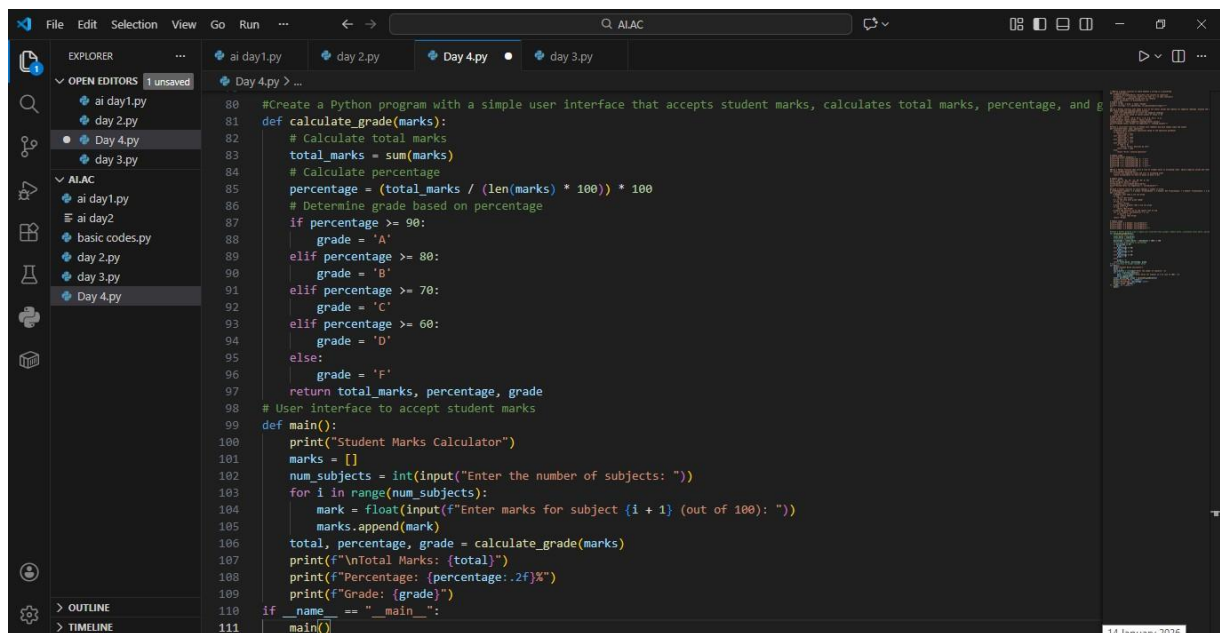
Few-shot prompting means providing **example inputs and outputs** along with the prompt. This helps the AI:

- Understand edge cases , Improve accuracy , Avoid logical mistakes

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

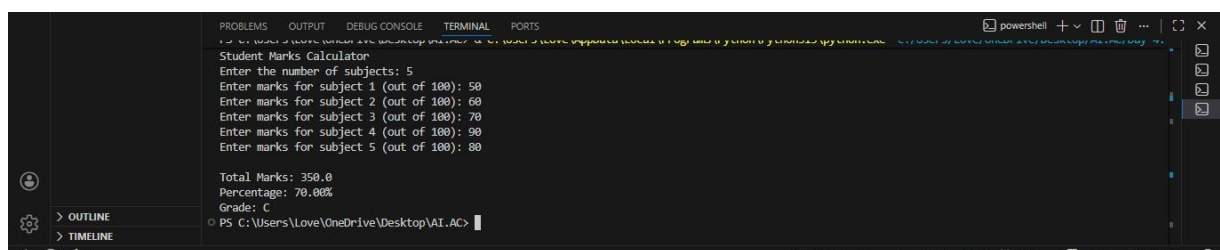
Task 4: Prompt-Guided UI Design – Student Grading System

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 g
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()
```

Output:



```
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
```

Explanation:

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

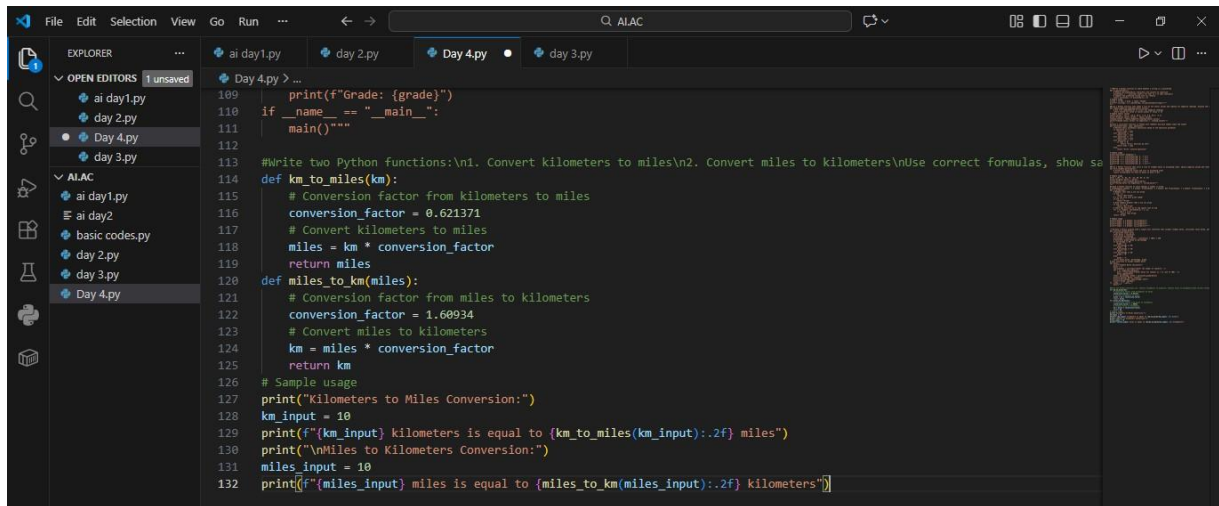
Instead of a graphical UI, a console-based UI is used for:

- Simplicity , Code compatibility , Clear user interaction

Task 5: Prompt Specificity – Unit Conversion Function

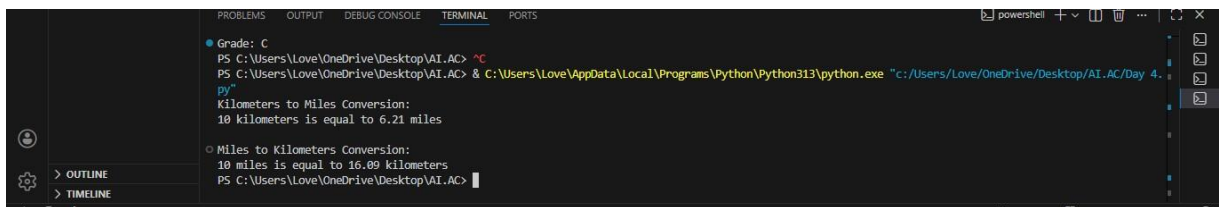
Prompt:

Write two Python functions:\n1. Convert kilometers to miles\n2. Convert miles to kilometers\nUse correct formulas, show sample input/output, and add comments explaining the logic



```
109 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 sa
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
133 print("\nMiles to Kilometers Conversion:")
134 miles_input = 10
135 print(f"{miles_input} miles is equal to {miles_to_km(miles_input):.2f} kilometers")
```

Output:



```
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**.