

A.Abhiram

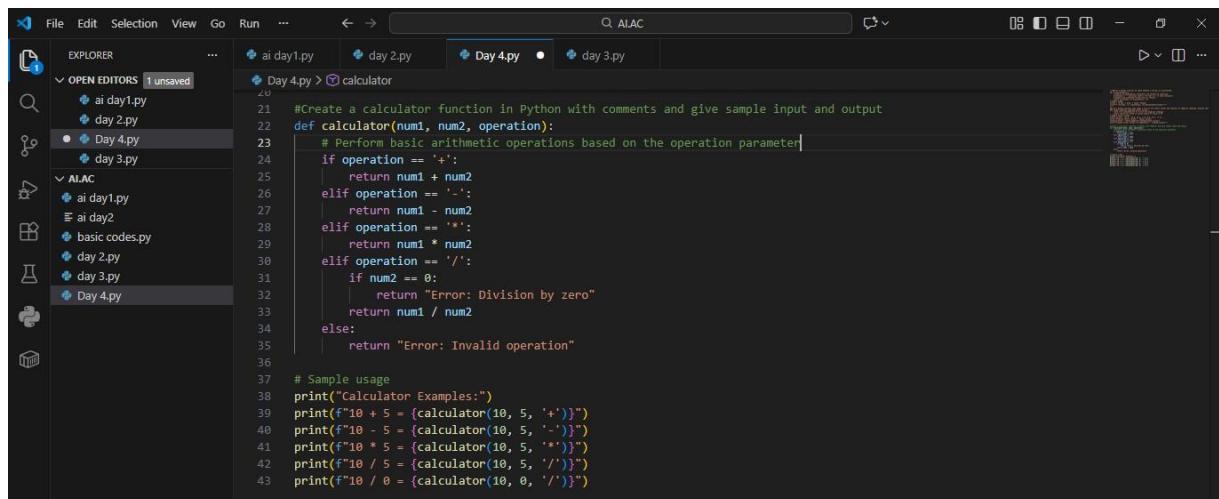
2403A51L13

Batch-51

ASSIGNMENT -3.2

Task 1: Progressive Prompting – Calculator Design

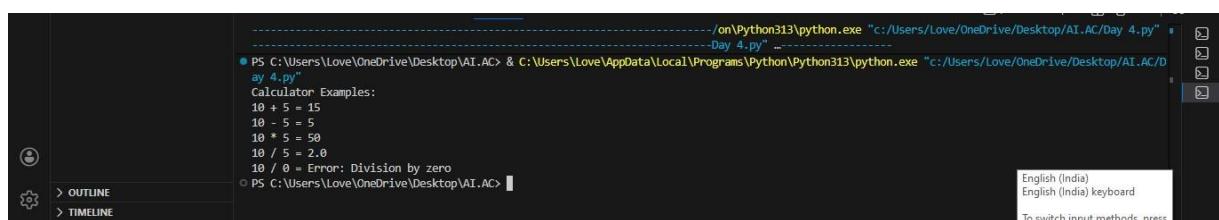
PROMPT: Create a calculator function in Python with comments and give sample input and output.



```
#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 = {}".format(calculator(10, 5, '+')))
print("10 - 5 = {}".format(calculator(10, 5, '-')))
print("10 * 5 = {}".format(calculator(10, 5, '*')))
print("10 / 5 = {}".format(calculator(10, 5, '/')))
print("10 / 0 = {}".format(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.



```
File Edit Selection View Go Run ... ← → Q ALAC

EXPLORER ...
OPEN EDITORS
ai day1.py day 2.py Day 4.py x day 3.py
Day 4.py > sort_student_marks
42 print(f"10 / 5 = {calculator(10, 5, '/')}")
43 print(f"10 / 0 = {calculator(10, 0, '/')}"")
44
45 #Write a Python function that sorts a list of student marks in ascending order. Ignore negative values and return the
46 def sort_student_marks(marks):
47     # Filter out negative values and sort in ascending order
48     return sorted([mark for mark in marks if mark >= 0])
49
50 # Sample usage
51 marks = [85, -5, 92, 78, -12, 88, 95, 0, 72]
52 print("Original marks:", marks)
53 sorted_marks = sort_student_marks(marks)
54 print("Sorted marks (no negatives):", sorted_marks)
```

OUTPUT:

EXPLANATION:

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

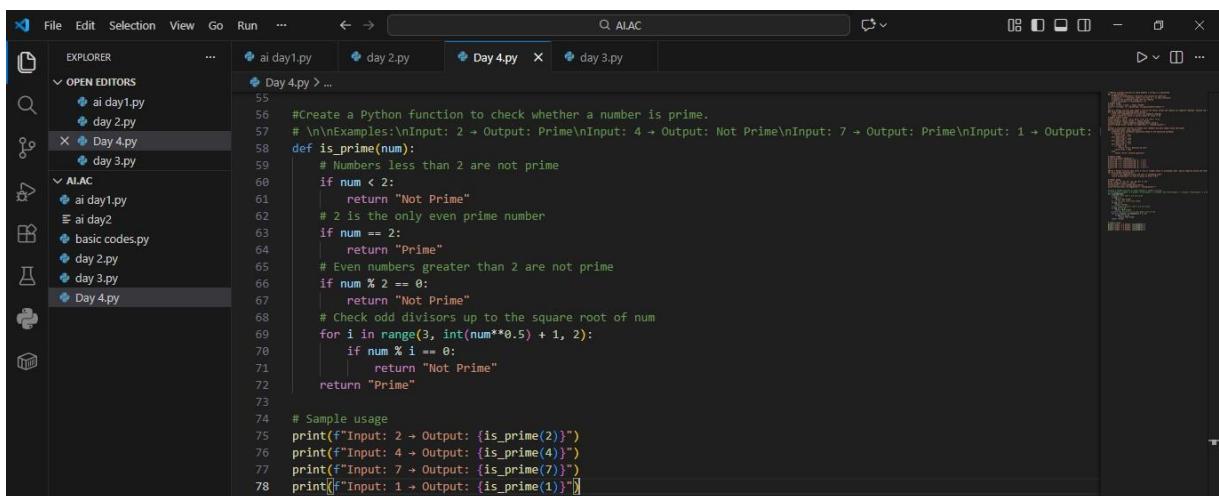
- Sorting order , Data constraints , Output format

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**.

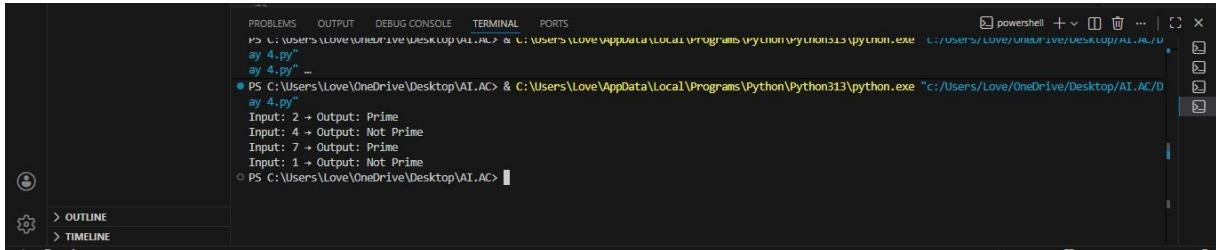
Task 3: Few-Shot Prompting – Prime Number Validation

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 # \n\nExamples:\nInput: 2 → Output: Prime\nInput: 4 → Output: Not Prime\nInput: 7 → Output: Prime\nInput: 1 → Output:
57 # Numbers less than 2 are not prime
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("Input: 2 → Output: {is_prime(2)}")
76 print("Input: 4 → Output: {is_prime(4)}")
77 print("Input: 7 → Output: {is_prime(7)}")
78 print("Input: 1 → Output: {is_prime(1)}")
```

OUTPUT:



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
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"
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:

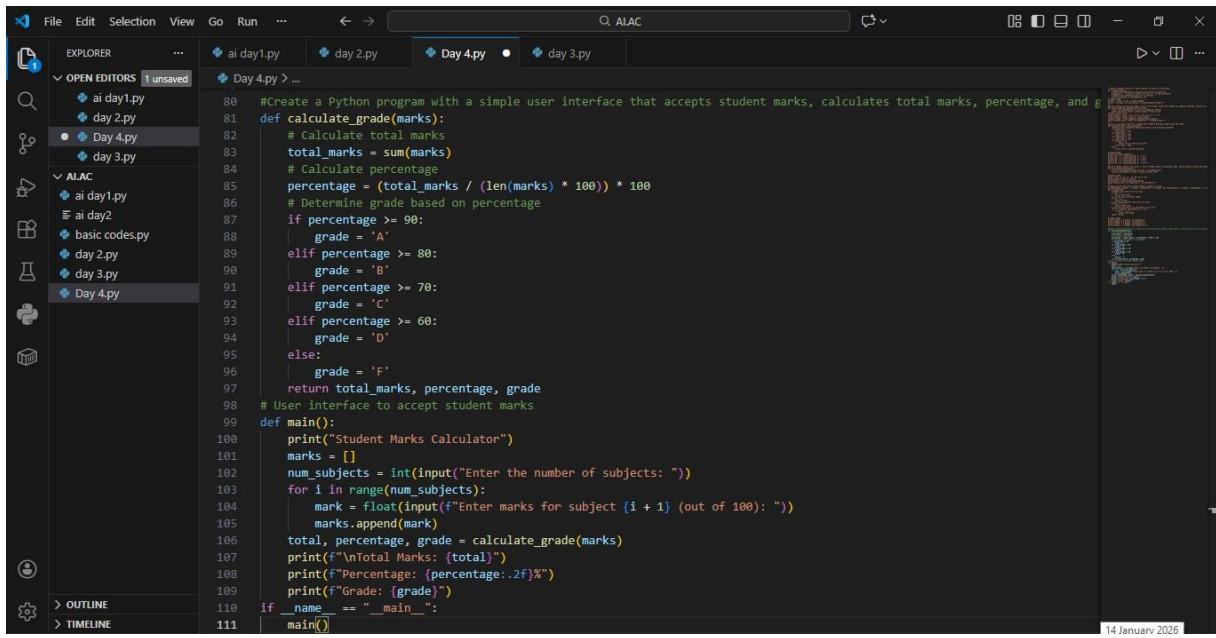
Few-shot prompting means providing **example inputs and outputs** along with the prompt. This helps the AI:

- 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.



```
File Edit Selection View Go Run ... ← → 🔍 ALAC
```

```
EXPLORER OPEN EDITORS 1 unsaved ai day1.py Day 4.py • day 2.py day 3.py Day 4 > ...  
ALAC ai day1.py ai day2 basic codes.py day 2.py day 3.py Day 4.py  
Day 4.py  
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()
```

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): 80  
Enter marks for subject 5 (out of 100): 90  
  
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.

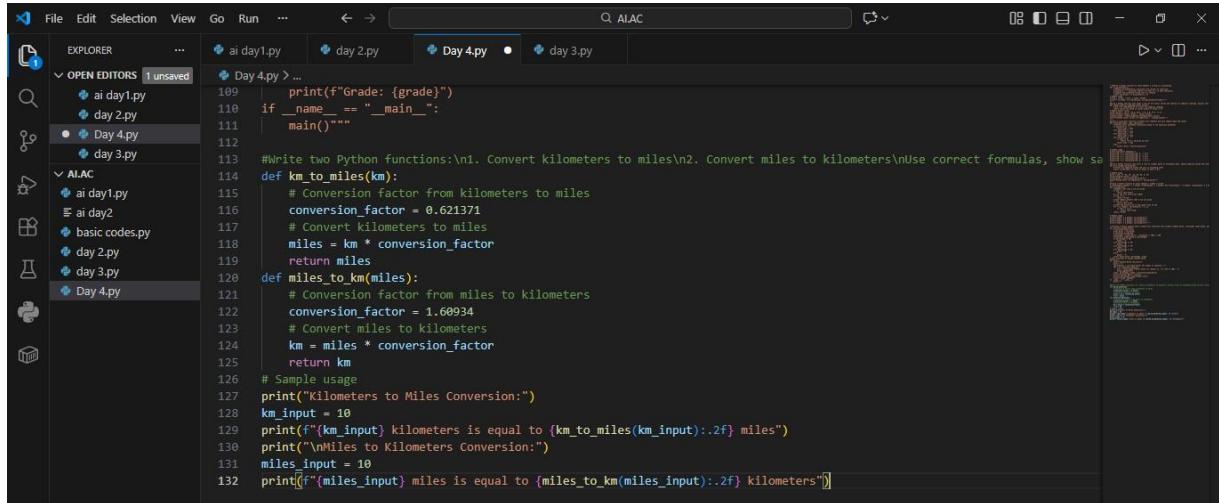
Instead of a graphical UI, a console-based UI is used for:

- Simplicity , Code compatibility , Clear user interaction

Task 5: Prompt Specificity – Unit Conversion Function

Prompt:

Write two Python functions:
1. Convert kilometers to miles
2. Convert miles to kilometers
Use correct formulas, show sample input/output, and add comments explaining the logic

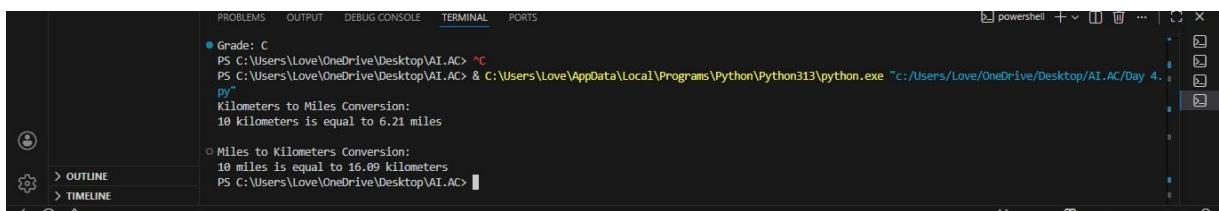


The screenshot shows the VS Code interface with the following details:

- File Bar:** File, Edit, Selection, View, Go, Run, ...
- Search Bar:** AI.AC
- Explorer:** OPEN EDITORS 1 unsaved, showing files: ai day1.py, ai day2.py, Day 4.py (selected), Day 4.py > ..., day 2.py, day 3.py.
- AI.AC:** Shows basic codes.py, ai day1.py, ai day2.py, day 2.py, day 3.py, Day 4.py.
- Code Editor:** Displays Python code for two conversion functions: km_to_miles and miles_to_km, along with sample usage and output.

```
114 #Write two Python functions:\n1. Convert kilometers to miles\n2.\n115 # Convert kilometers to miles\n116 conversion_factor = 0.621371\n117 # Convert miles to kilometers\n118 miles = km * conversion_factor\n119 return miles\n120 def miles_to_km(miles):\n121     # Conversion factor from miles to kilometers\n122     conversion_factor = 1.60934\n123     # Convert miles to kilometers\n124     km = miles * conversion_factor\n125     return km\n126 # Sample usage\n127 print("Kilometers to Miles Conversion:")\n128 km_input = 10\n129 print(f"\n{km_input} Kilometers is equal to {km_to_miles(km_input):.2f} miles")\n130 print("\nMiles to Kilometers Conversion:")\n131 miles_input = 10\n132 print(f"\n{miles_input} miles is equal to {miles_to_km(miles_input):.2f} kilometers")
```

Output:



The screenshot shows the terminal tab in VS Code with the following output:

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS\n● Grade: C\nPS C:\Users\Love\OneDrive\Desktop\AI.AC> ^C\nPS 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"\nKilometers to Miles Conversion:\n10 kilometers is equal to 6.21 miles\n\nMiles to Kilometers Conversion:\n10 miles is equal to 16.09 kilometers\nPS 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**.