Keerthana Erukala

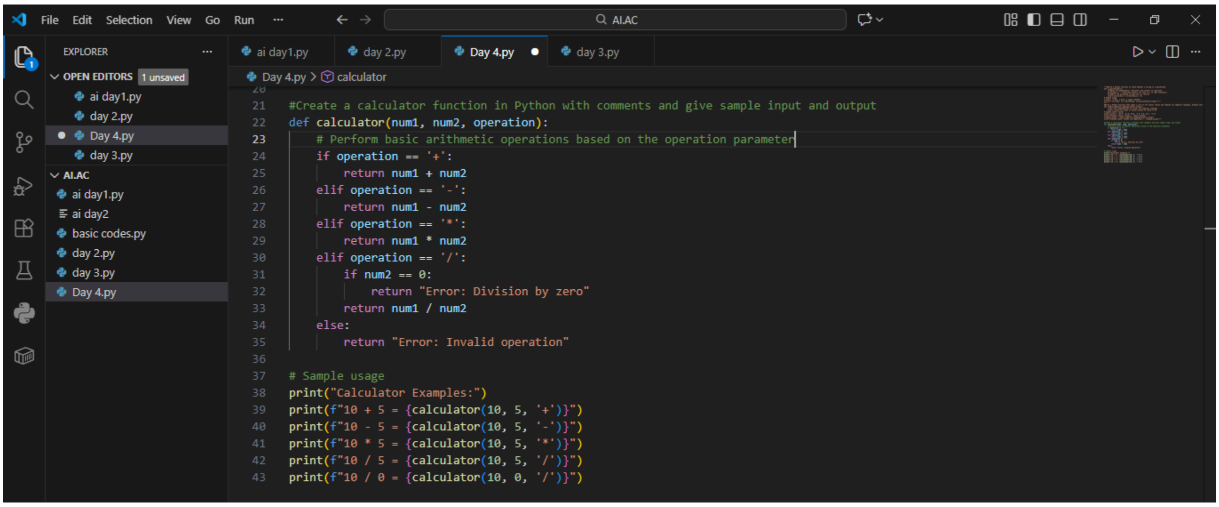
2403A51L06

B-51

ASSIGNMENT -3.2

Task 1: Progressive Prompting – Calculator Design

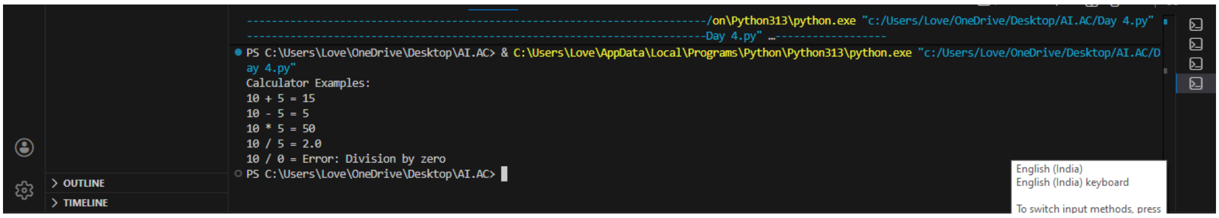
PROMPT: Create a calculator function in Python with comments and give sample input and output.



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

OUTPUT:



```
-----------------------------------------------------------------------/on\Python313\python.exe "c:/Users/Love/OneDrive/Desktop/AI.AC/Day 4.py"
-----------------------------------------------------------------------Day 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"
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.

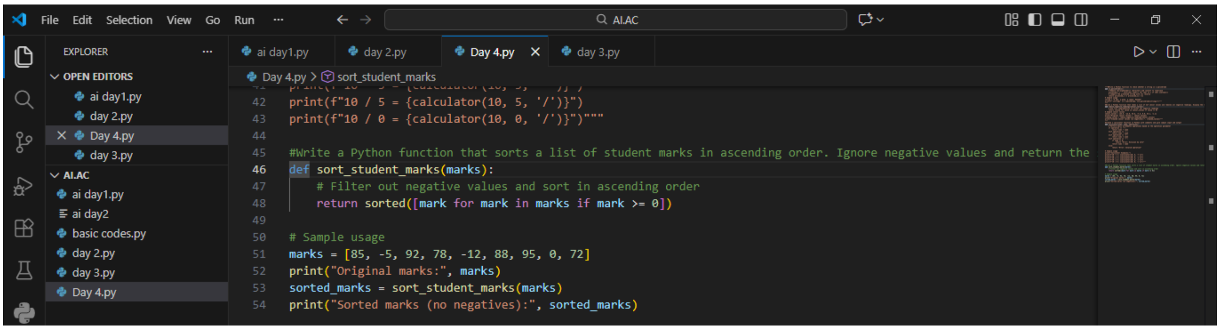
As we gradually add comments, requirements, and examples, the AI understands better and produces:

- 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.

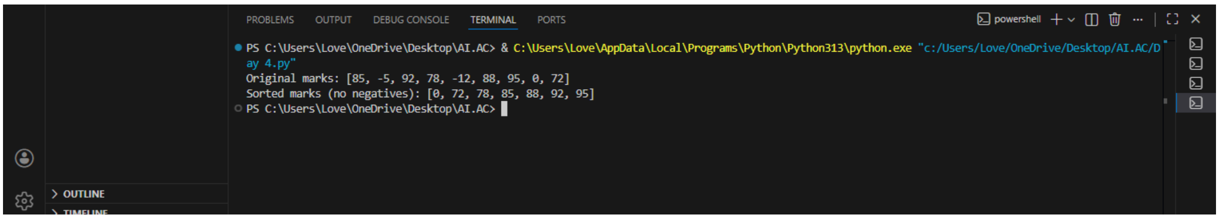


```python
print(f"10 / 5 = {calculator(10, 5, '/')}")
print(f"10 / 0 = {calculator(10, 0, '/')}")"""

#Write a Python function that sorts a list of student marks in ascending order. Ignore negative values and return the
def sort_student_marks(marks):
    # Filter out negative values and sort in ascending order
    return sorted([mark for mark in marks if mark >= 0])

# 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:



```
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): [0, 72, 78, 85, 88, 92, 95]
PS C:\Users\Love\OneDrive\Desktop\AI.AC>
```

EXPLANATION:

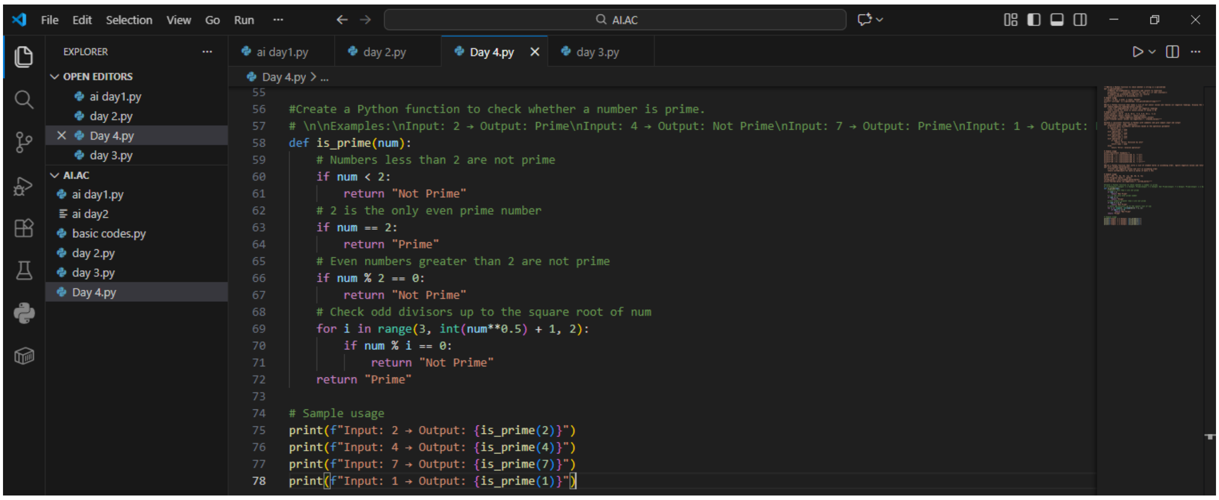
This task demonstrates how **vague prompts cause ambiguous results**. Initially, the AI may not know:

- 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 . Use these examples to design the logic

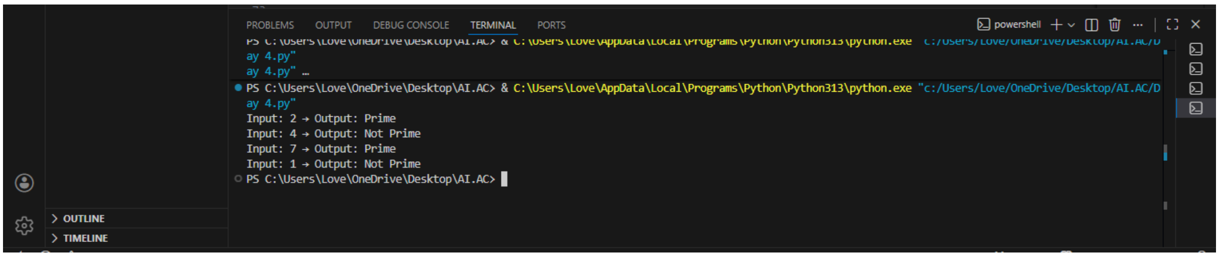


```python

#Create a Python function to check whether a number is prime.
# \n\nExamples:\nInput: 2 → Output: Prime\nInput: 4 → Output: Not Prime\nInput: 7 → Output: Prime\nInput: 1 → Output:
def is_prime(num):
    # Numbers less than 2 are not prime
    if num < 2:
        return "Not Prime"
    # 2 is the only even prime number
    if num == 2:
        return "Prime"
    # Even numbers greater than 2 are not prime
    if num % 2 == 0:
        return "Not Prime"
    # Check odd divisors up to the square root of num
    for i in range(3, int(num**0.5) + 1, 2):
        if num % i == 0:
            return "Not Prime"
    return "Prime"

# Sample usage
print(f"Input: 2 → Output: {is_prime(2)}")
print(f"Input: 4 → Output: {is_prime(4)}")
print(f"Input: 7 → Output: {is_prime(7)}")
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"
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:

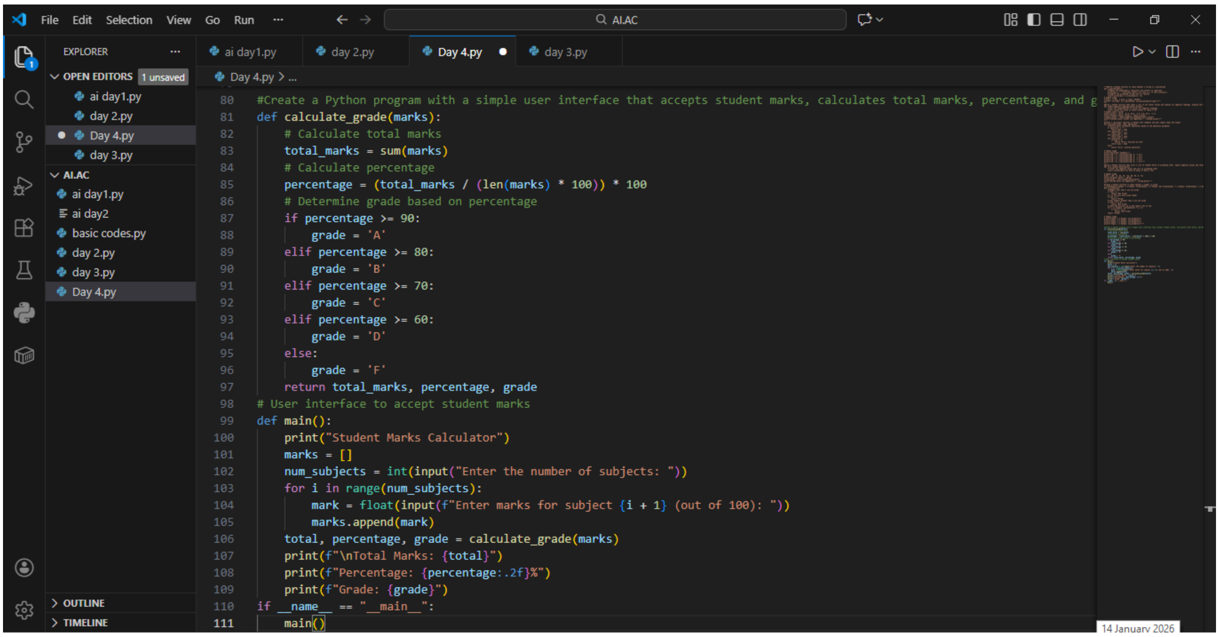
Few-shot prompting means providing **example inputs and outputs** along with the prompt.
This helps the AI:

- 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

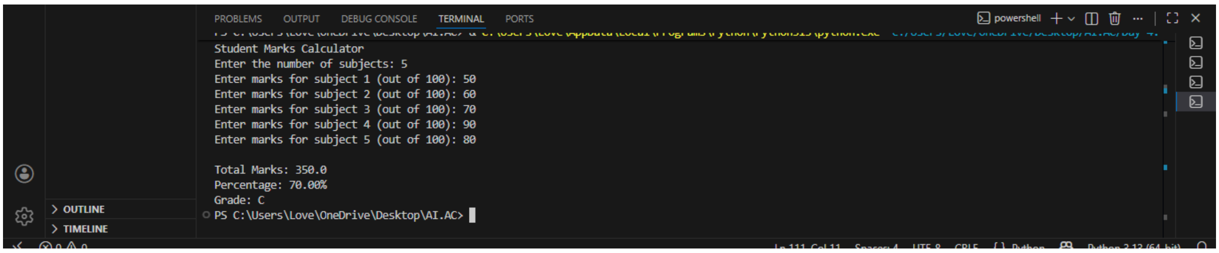
Prompt : Create a Python program with a simple user interface that accepts student marks, calculates total marks, percentage, and grade, and displays the result.



```python
#Create a Python program with a simple user interface that accepts student marks, calculates total marks, percentage, and g
def calculate_grade(marks):
    # Calculate total marks
    total_marks = sum(marks)
    # Calculate percentage
    percentage = (total_marks / (len(marks) * 100)) * 100
    # Determine grade based on percentage
    if percentage >= 90:
        grade = 'A'
    elif percentage >= 80:
        grade = 'B'
    elif percentage >= 70:
        grade = 'C'
    elif percentage >= 60:
        grade = 'D'
    else:
        grade = 'F'
    return total_marks, percentage, grade
# User interface to accept student marks
def main():
    print("Student Marks Calculator")
    marks = []
    num_subjects = int(input("Enter the number of subjects: "))
    for i in range(num_subjects):
        mark = float(input(f"Enter marks for subject {i + 1} (out of 100): "))
        marks.append(mark)
    total, percentage, grade = calculate_grade(marks)
    print(f"\nTotal Marks: {total}")
    print(f"Percentage: {percentage:.2f}%")
    print(f"Grade: {grade}")
if __name__ == "__main__":
    main()
```

14 January 2026

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
PS C:\Users\Love\OneDrive\Desktop\AI.AC>
```

Explanation:

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

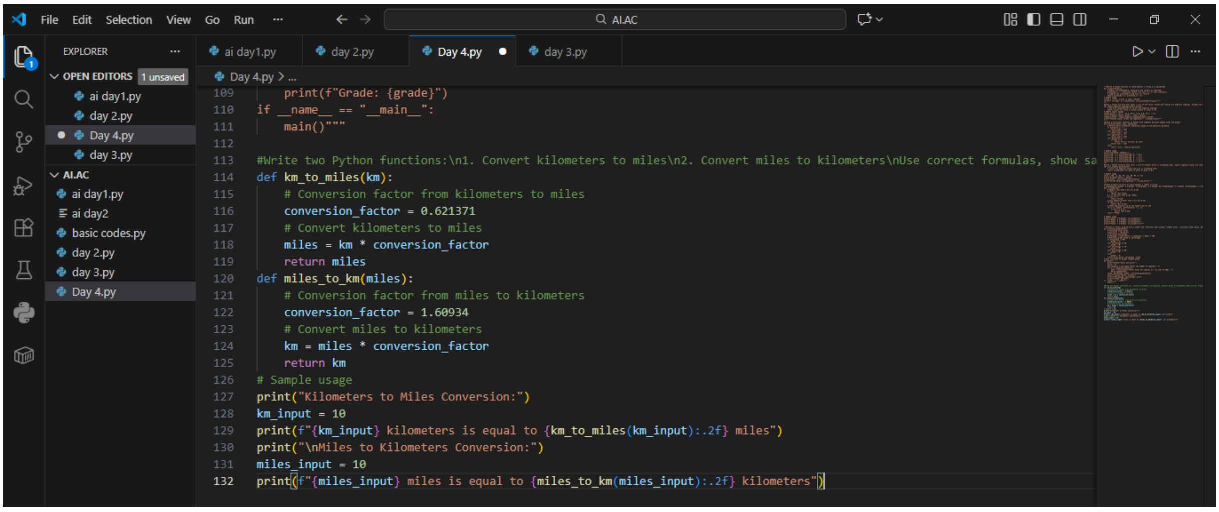
Instead of a graphical UI, a console-based UI is used for:

- Simplicity , Code compatibility , Clear user interaction

Task 5: Prompt Specificity – Unit Conversion Function

Prompt:

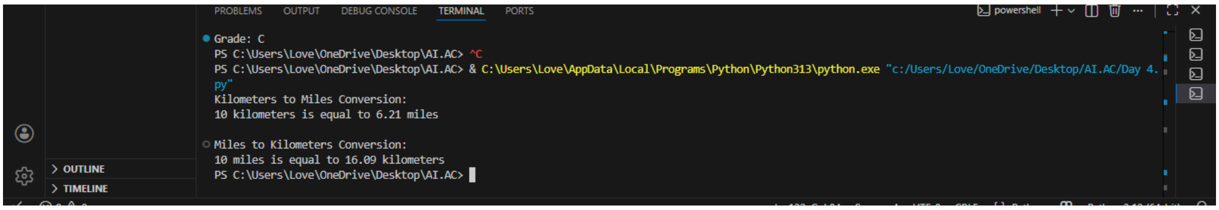
Write two Python functions:\n1. Convert kilometers to miles\n2. Convert miles to kilometers\nUse correct formulas, show sample input/output, and add comments explaining the logic



```python
    print(f"Grade: {grade}")
if __name__ == "__main__":
    main()"""

#Write two Python functions:\n1. Convert kilometers to miles\n2. Convert miles to kilometers\nUse correct formulas, show sa
def km_to_miles(km):
    # Conversion factor from kilometers to miles
    conversion_factor = 0.621371
    # Convert kilometers to miles
    miles = km * conversion_factor
    return miles
def miles_to_km(miles):
    # Conversion factor from miles to kilometers
    conversion_factor = 1.60934
    # Convert miles to kilometers
    km = miles * conversion_factor
    return km
# Sample usage
print("Kilometers to Miles Conversion:")
km_input = 10
print(f"{km_input} kilometers is equal to {km_to_miles(km_input):.2f} miles")
print("\nMiles to Kilometers Conversion:")
miles_input = 10
print(f"{miles_input} miles is equal to {miles_to_km(miles_input):.2f} kilometers")
```

Output:



```
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**.