AI Assisted Coding

Assignment – 3.2

Name: Bandi Akiranandini
Batch:22

Hallticket:2303a51516

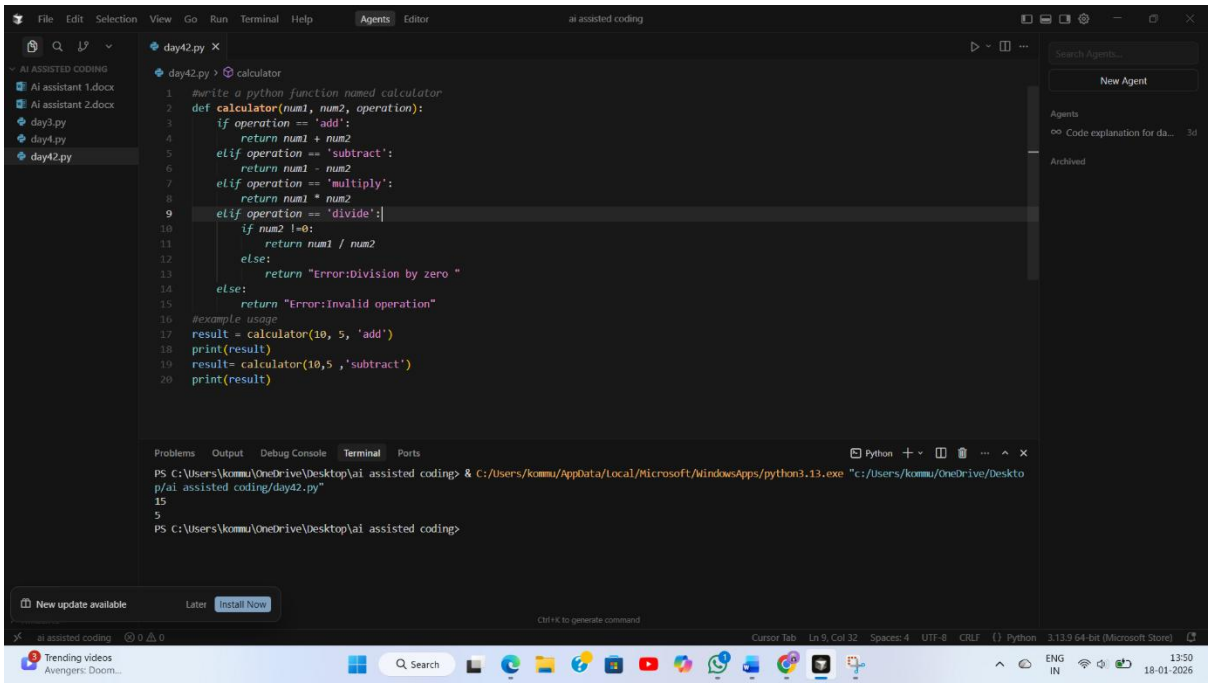
Task Description-1

• 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.

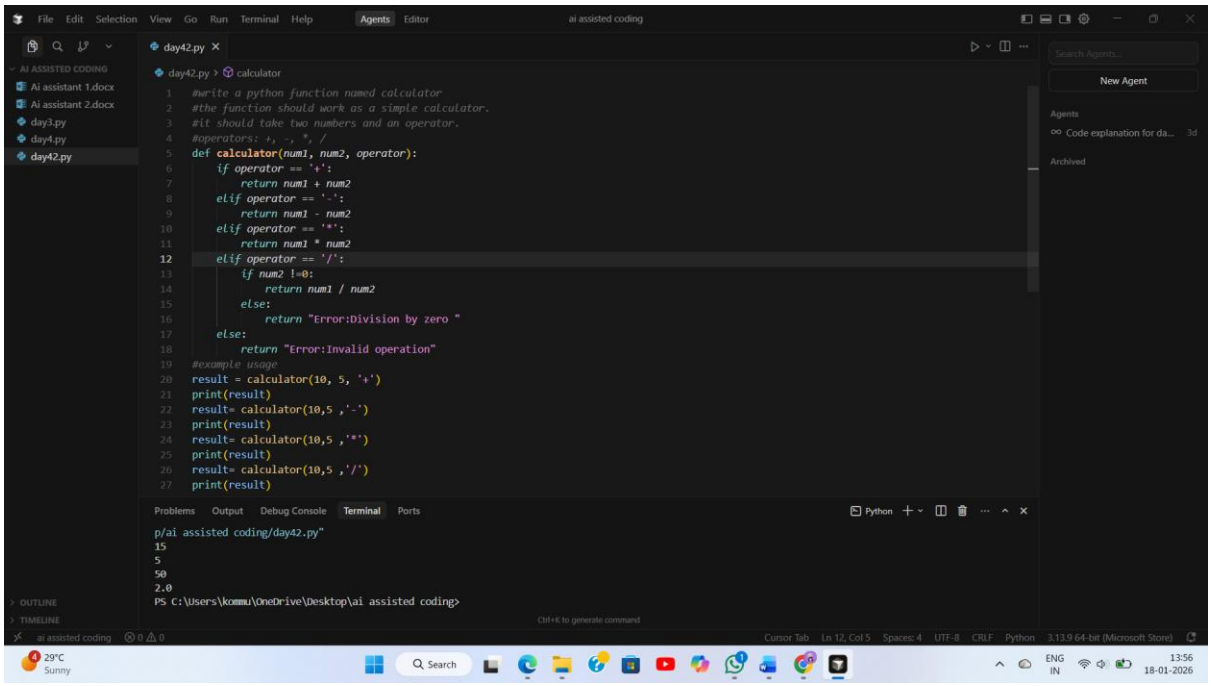
Expected Output-1

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

Stage 1:



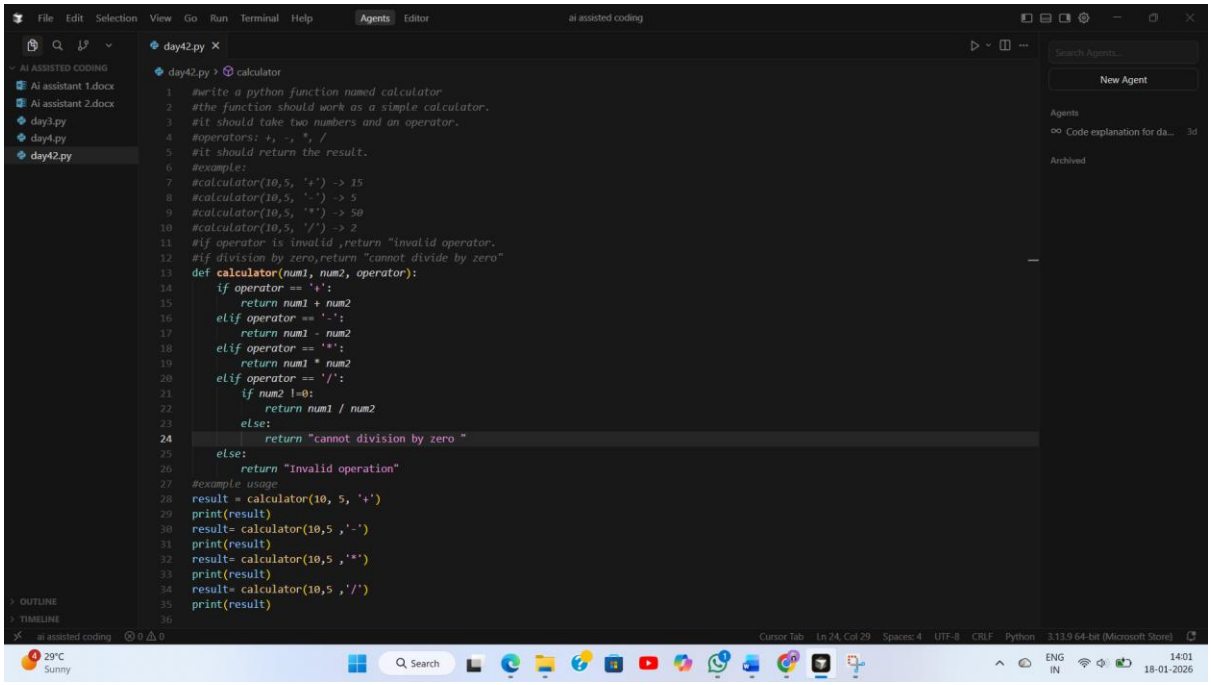
Stage 2:



```python
#write a python function named calculator
#the function should work as a simple calculator.
#it should take two numbers and an operator.
#operators: +, -, *, /
def calculator(num1, num2, operator):
    if operator == '+':
        return num1 + num2
    elif operator == '-':
        return num1 - num2
    elif operator == '*':
        return num1 * num2
    elif operator == '/':
        if num2 !=0:
            return num1 / num2
        else:
            return "Error:Division by zero "
    else:
        return "Error:Invalid operation"
#example usage
result = calculator(10, 5, '+')
print(result)
result= calculator(10,5 ,'-')
print(result)
result= calculator(10,5 ,'*')
print(result)
result= calculator(10,5 ,'/')
print(result)
```

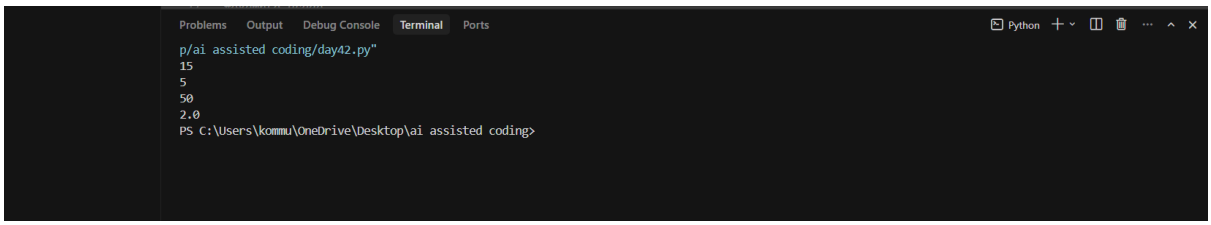
```
p/ai assisted coding/day42.py"
15
5
50
2.0
PS C:\Users\kommu\OneDrive\Desktop\ai assisted coding>
```

Stage3:



```python
#write a python function named calculator
#the function should work as a simple calculator.
#it should take two numbers and an operator.
#operators: +, -, *, /
#it should return the result.
#example:
#calculator(10,5, '+') -> 15
#calculator(10,5, '-') -> 5
#calculator(10,5, '*') -> 50
#calculator(10,5, '/') -> 2
#if operator is invalid ,return "invalid operator.
#if division by zero,return "cannot divide by zero"
def calculator(num1, num2, operator):
    if operator == '+':
        return num1 + num2
    elif operator == '-':
        return num1 - num2
    elif operator == '*':
        return num1 * num2
    elif operator == '/':
        if num2 !=0:
            return num1 / num2
        else:
            return "cannot division by zero "
    else:
        return "Invalid operation"
#example usage
result = calculator(10, 5, '+')
print(result)
result= calculator(10,5 ,'-')
print(result)
result= calculator(10,5 ,'*')
print(result)
result= calculator(10,5 ,'/')
print(result)
```

Output:



```
p/ai assisted coding/day42.py"
15
5
50
2.0
PS C:\Users\kommu\OneDrive\Desktop\ai assisted coding>
```

Own Experience or observation:

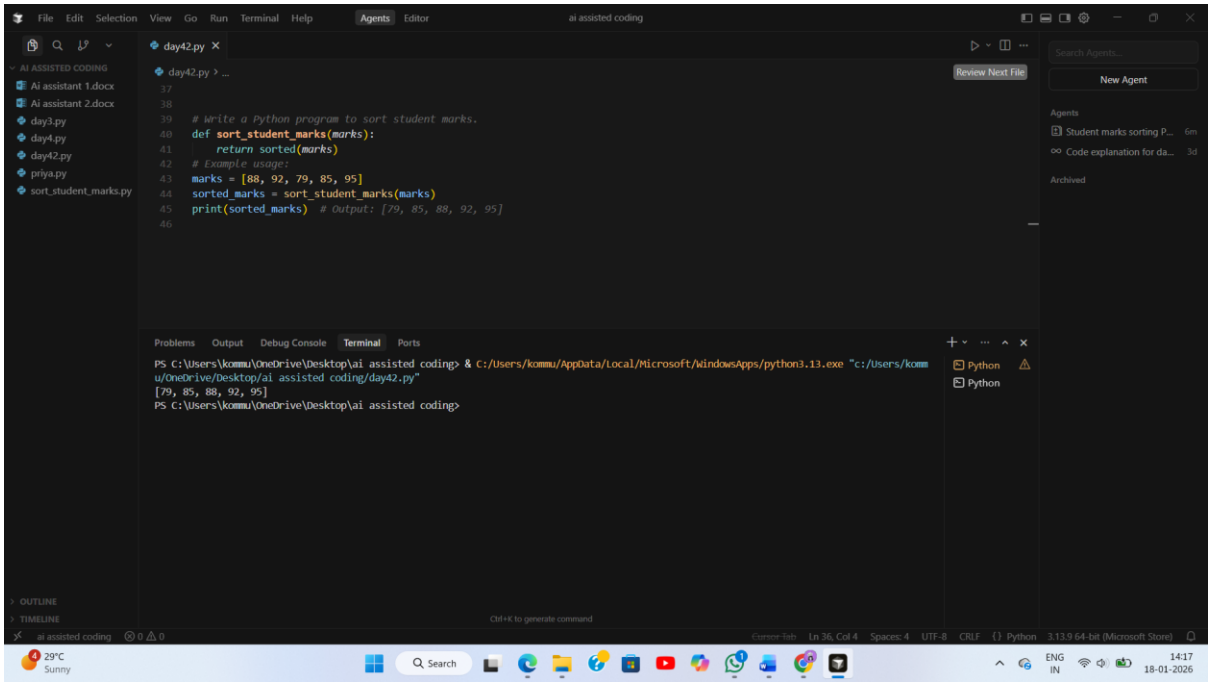
At first, when only the function name was given, the AI generated a very basic and incomplete calculator function with little or no logic. After adding comments, the AI started including parameters and arithmetic operations. When usage examples were finally added, the AI produced a complete and well-structured calculator program with proper conditions and error handling. This clearly shows that progressive prompting improves both the logic and structure of the generated code.

Question 2:

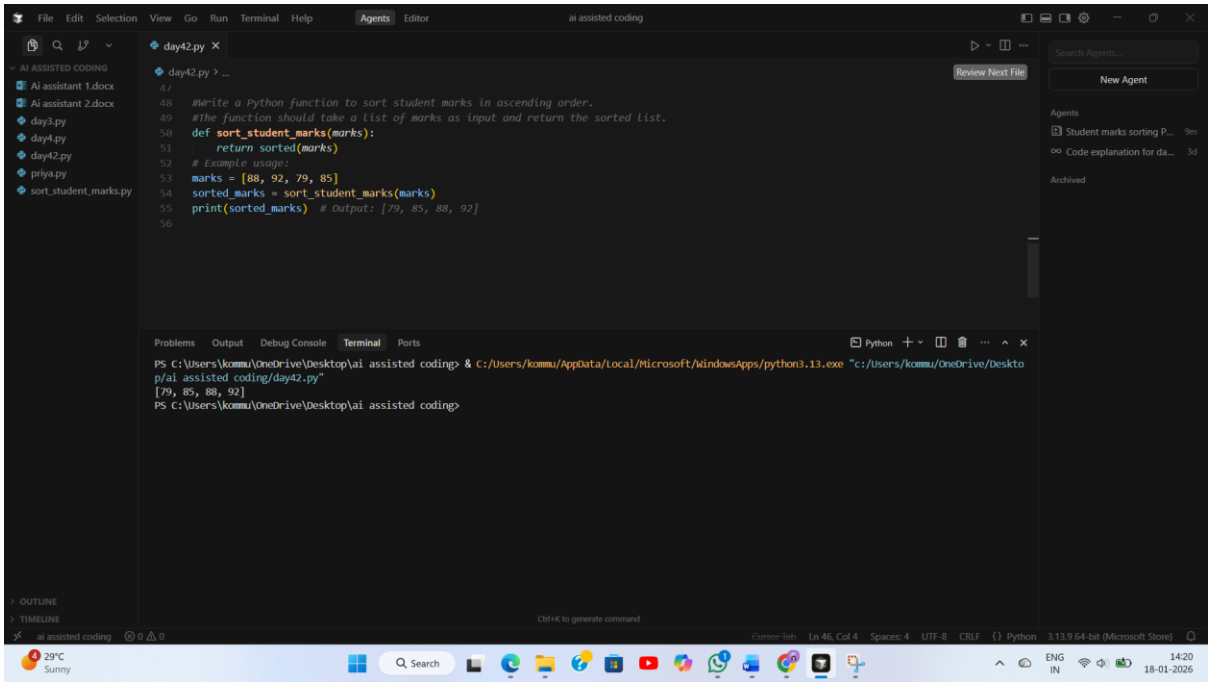
Task Description-2 Refining Prompts for Sorting Logic: Start with a vague prompt for sorting student marks, then refine it to clearly specify sorting order and constraints.

Expected Output-2 AI-generated sorting function evolves from ambiguous logic to an accurate and efficient implementation.

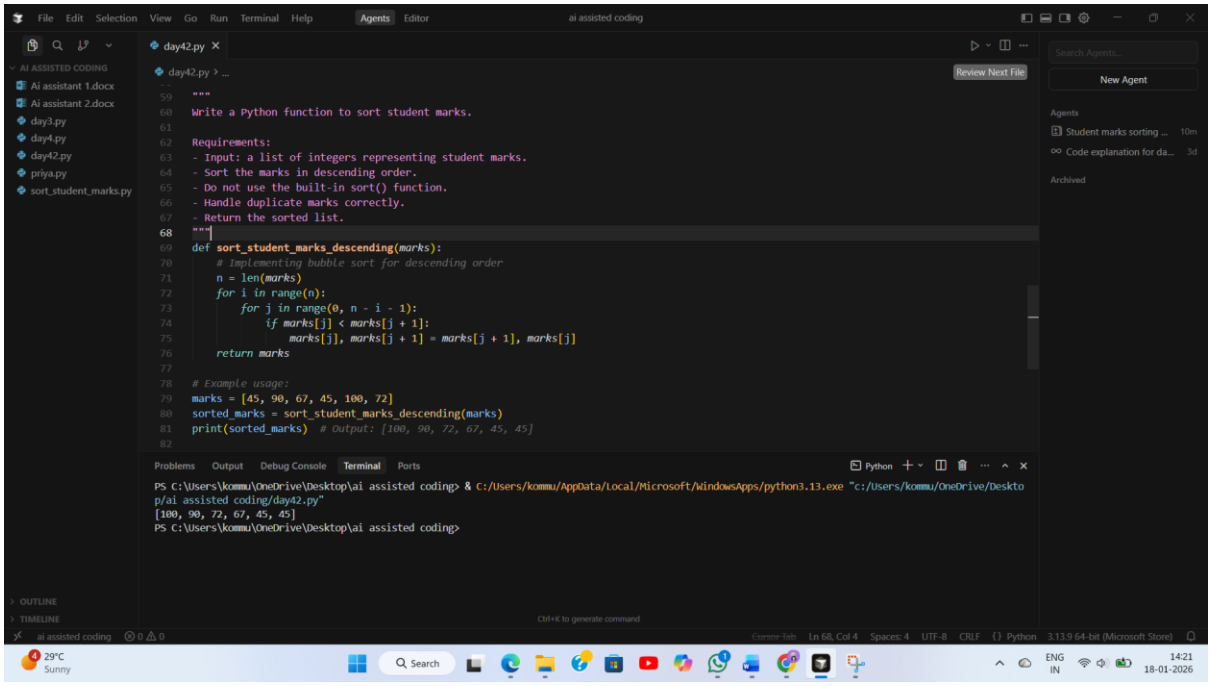
Stage 1:



Stage2:



Stage3:



Observation:

With a vague prompt, the AI produced a simple sorting solution without clear direction or constraints. After refining the prompt to specify sorting order, the output became more accurate and meaningful. When clear constraints and examples were added, the AI generated a more structured

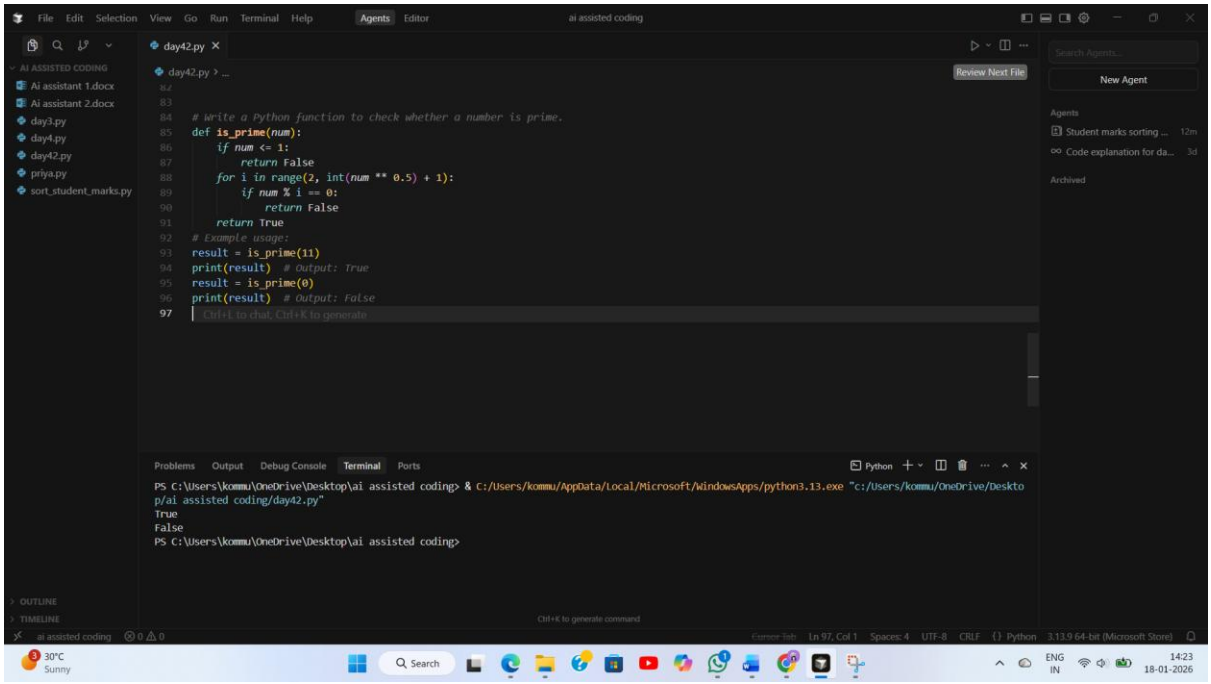
and efficient sorting function. This demonstrates that refining prompts helps the AI move from ambiguous logic to a correct and reliable implementation.

Question 3:

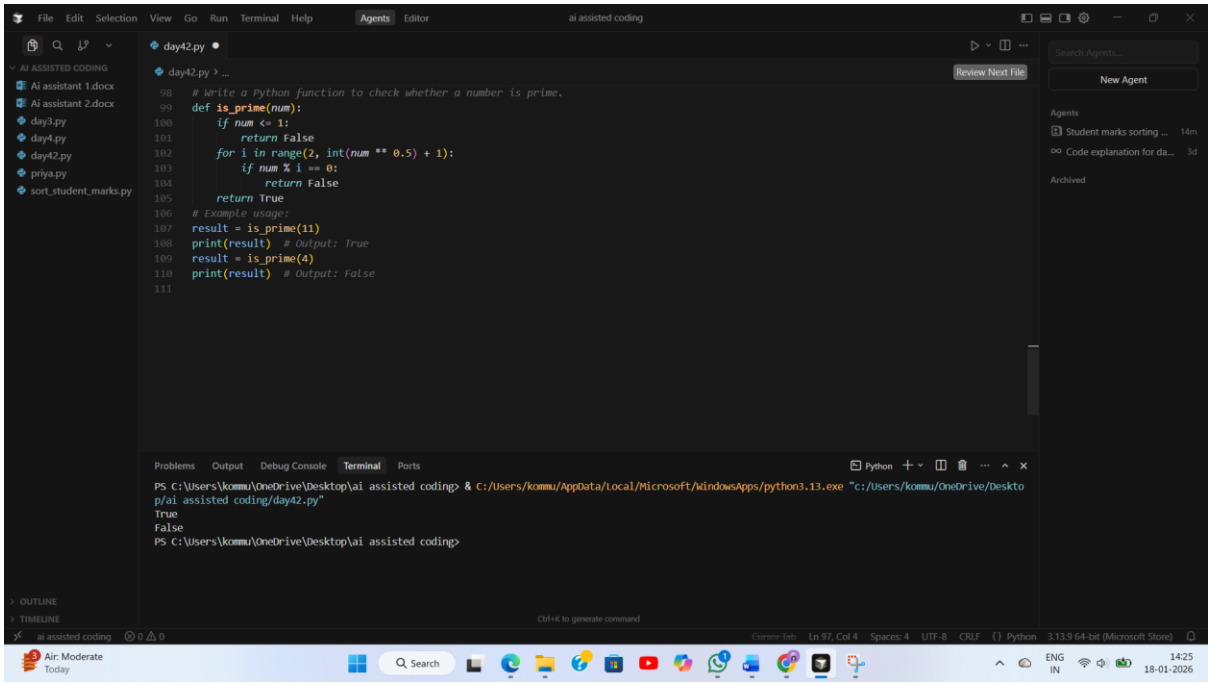
Task Description-3 • Few-Shot Prompting for Prime Number Validation: Provide multiple input-output examples for a function that checks whether a number is prime. Observe how few-shot prompting improves correctness.

Expected Output-3 • Improved prime-checking function with better edge-case handling.

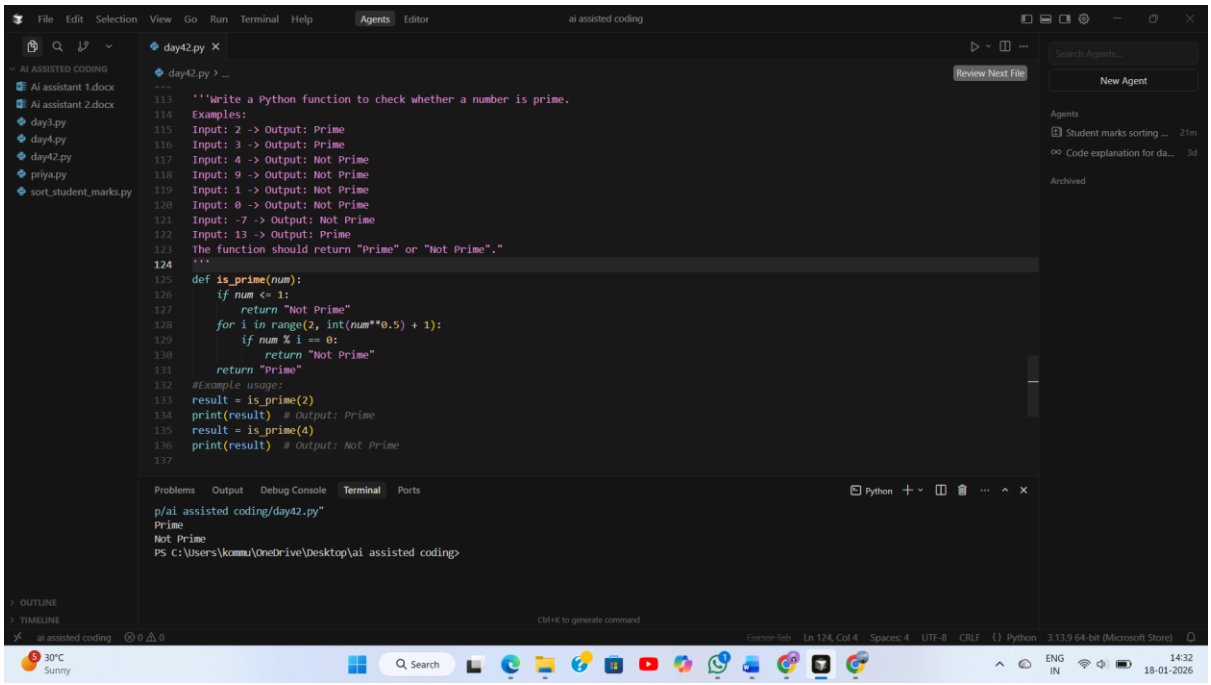
Stage 1:



Stage2:



Stage3:



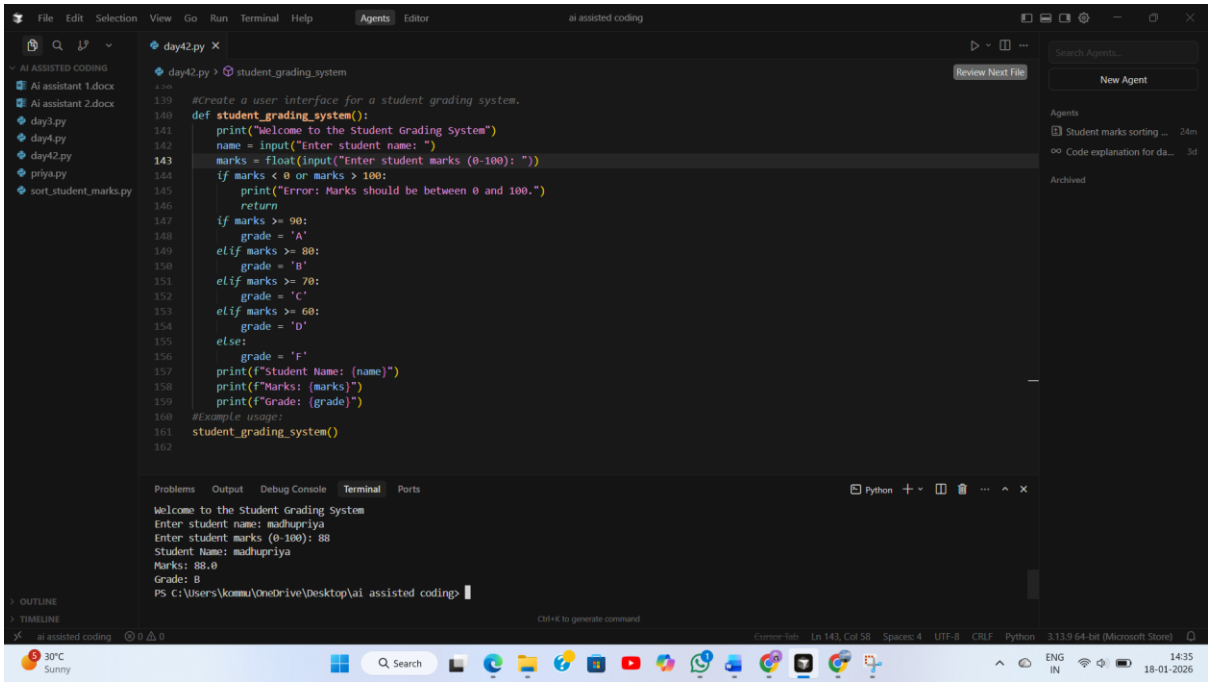
Observation: In the initial prompt without examples, the AI generated a basic prime-checking function that could miss important edge cases. When one example was provided, the result improved slightly. After giving multiple input-output examples (few-shot prompting), the AI clearly handled cases like 0, 1, and negative numbers and produced a more accurate and robust prime-checking function. This shows that few-shot prompting improves correctness and edge-case handling.

Question 4:

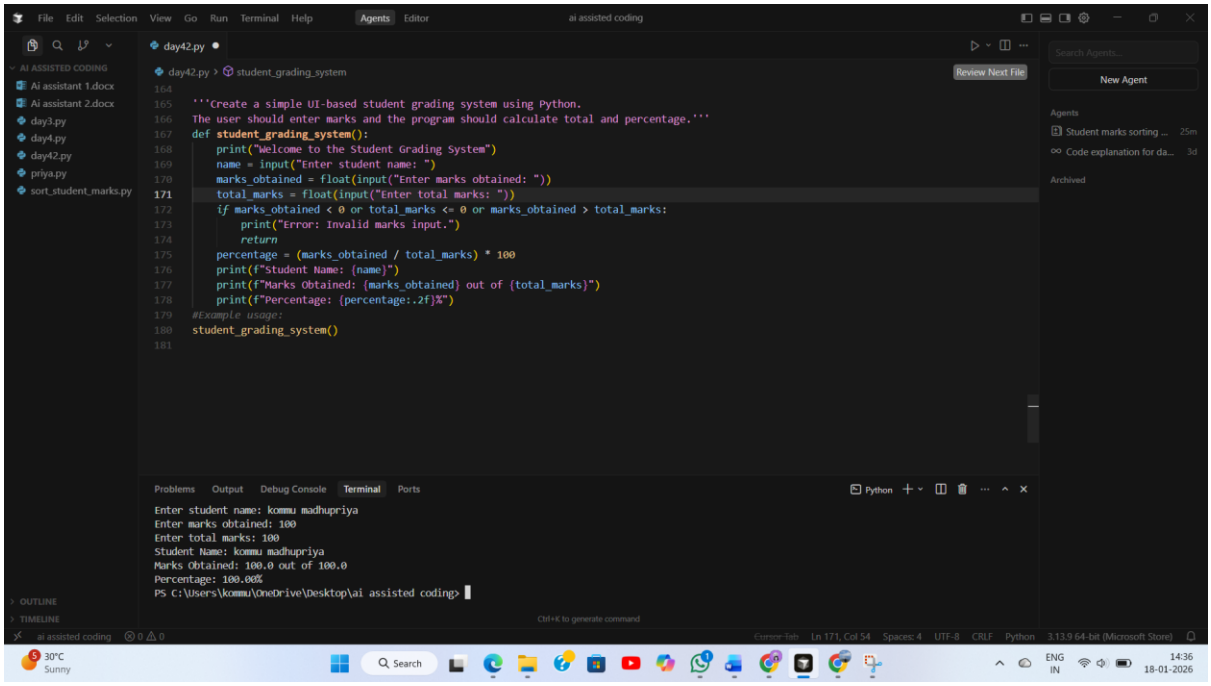
Task Description-4 • Prompt-Guided UI Design for Student Grading System: Create a user interface for a student grading system that calculates total marks, percentage, and grade based on user input.

Expected Output-4 • Well-structured UI code with accurate calculations and clear output display.

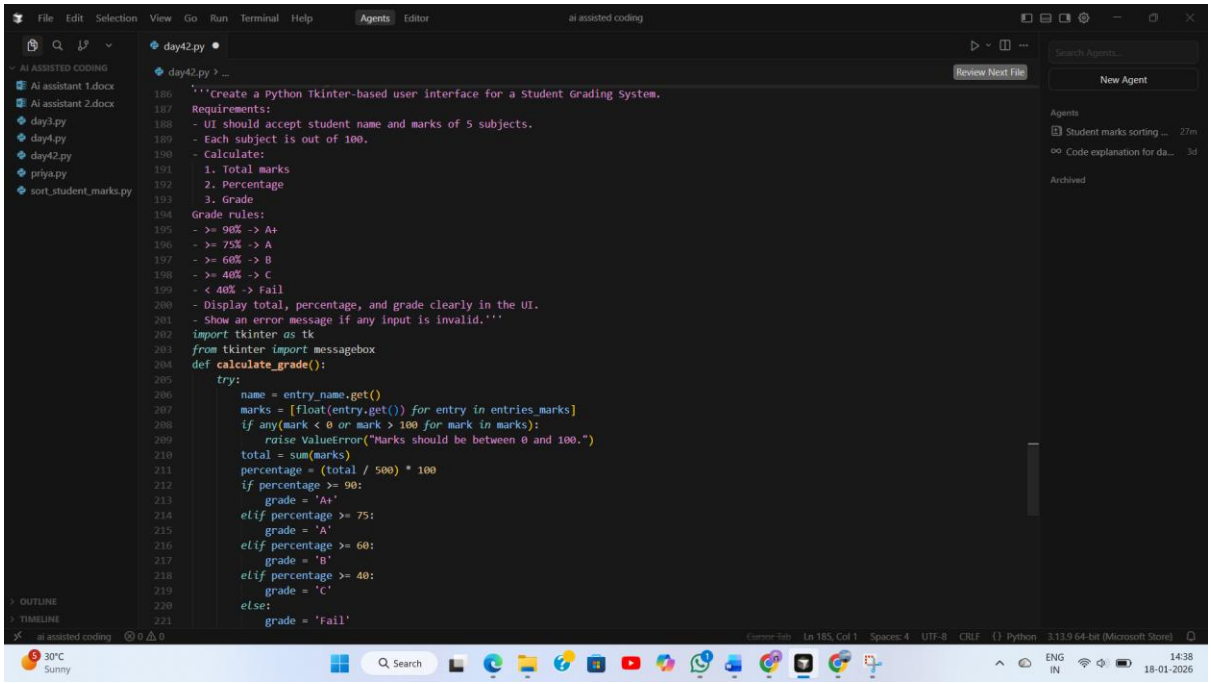
Stage1:



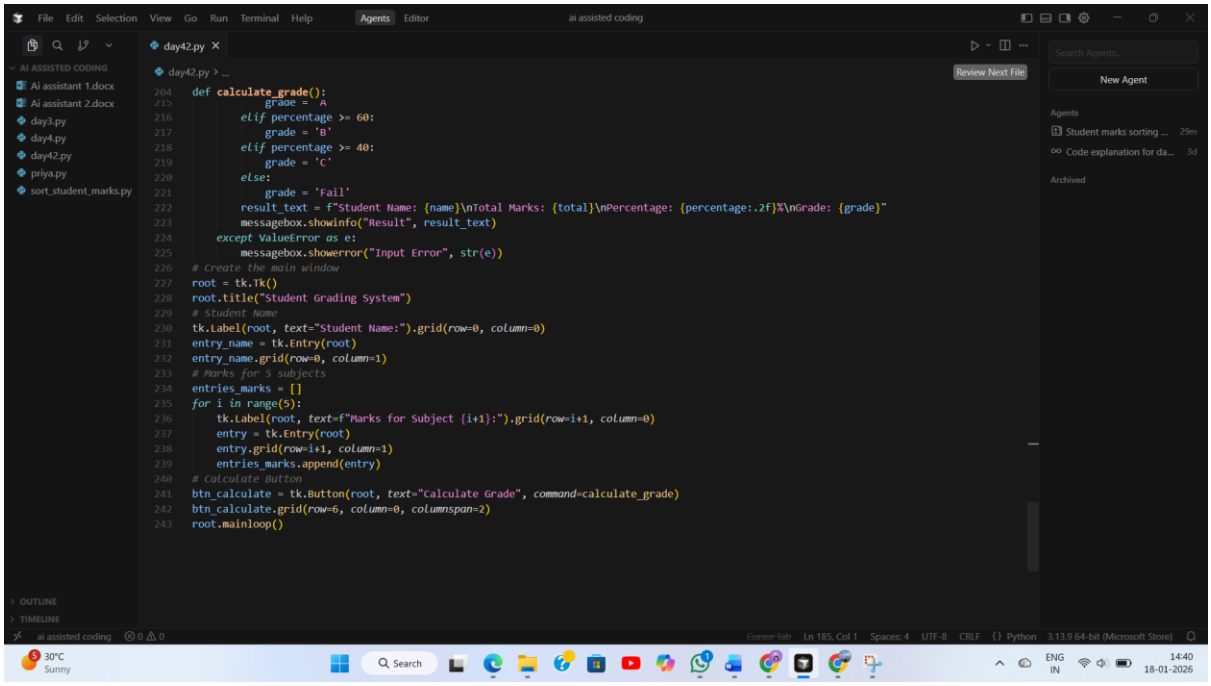
Stage2:



Stage3:

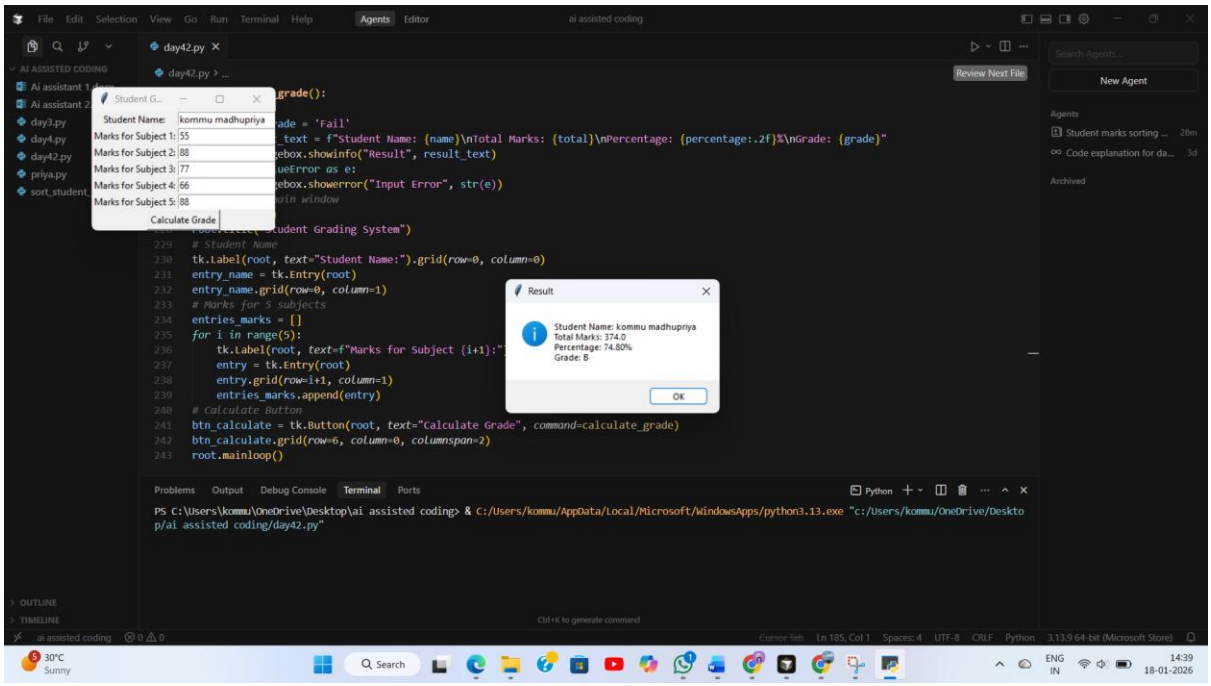


```
'''Create a Python Tkinter-based user interface for a Student Grading System.
Requirements:
- UI should accept student name and marks of 5 subjects.
- Each subject is out of 100.
- Calculate:
  1. Total marks
  2. Percentage
  3. Grade
Grade rules:
- >= 90% -> A+
- >= 75% -> A
- >= 60% -> B
- >= 40% -> C
- < 40% -> Fail
- Display total, percentage, and grade clearly in the UI.
- Show an error message if any input is invalid.'''
import tkinter as tk
from tkinter import messagebox
def calculate_grade():
    try:
        name = entry_name.get()
        marks = [float(entry.get()) for entry in entries_marks]
        if any(mark < 0 or mark > 100 for mark in marks):
            raise ValueError("Marks should be between 0 and 100.")
        total = sum(marks)
        percentage = (total / 500) * 100
        if percentage >= 90:
            grade = 'A+'
        elif percentage >= 75:
            grade = 'A'
        elif percentage >= 60:
            grade = 'B'
        elif percentage >= 40:
            grade = 'C'
        else:
            grade = 'Fail'
```



```
def calculate_grade():
            grade = 'A'
        elif percentage >= 60:
            grade = 'B'
        elif percentage >= 40:
            grade = 'C'
        else:
            grade = 'Fail'
        result_text = f"Student Name: {name}\nTotal Marks: {total}\nPercentage: {percentage:.2f}%\nGrade: {grade}"
        messagebox.showinfo("Result", result_text)
    except ValueError as e:
        messagebox.showerror("Input Error", str(e))
# Create the main window
root = tk.Tk()
root.title("Student Grading System")
# Student Name
tk.Label(root, text="Student Name:").grid(row=0, column=0)
entry_name = tk.Entry(root)
entry_name.grid(row=0, column=1)
# Marks for 5 subjects
entries_marks = []
for i in range(5):
    tk.Label(root, text=f"Marks for Subject {i+1}:").grid(row=i+1, column=0)
    entry = tk.Entry(root)
    entry.grid(row=i+1, column=1)
    entries_marks.append(entry)
# Calculate Button
btn_calculate = tk.Button(root, text="Calculate Grade", command=calculate_grade)
btn_calculate.grid(row=6, column=0, columnspan=2)
root.mainloop()
```

Output:



Observation:

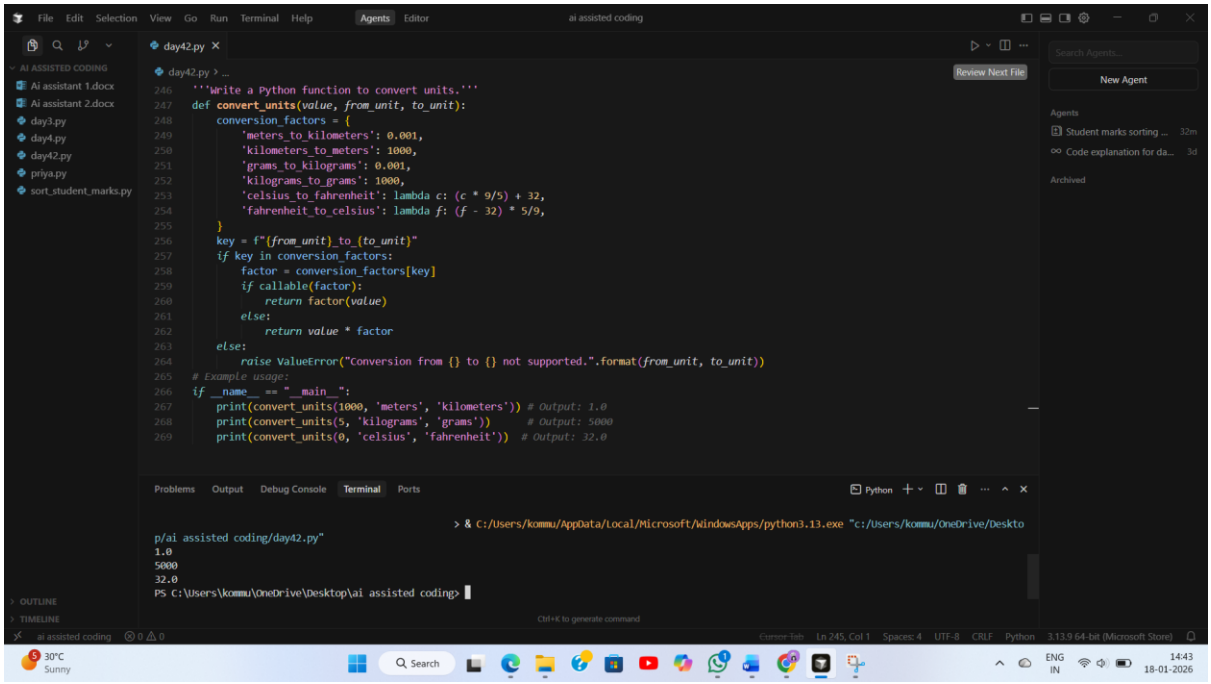
With a vague UI prompt, the AI produced only a simple or unclear interface idea. As the prompt was refined to include calculation requirements, the UI output became more meaningful. When full instructions were given (inputs, calculations, grade rules, and display), the AI generated a well-structured user interface with correct total, percentage, and grade calculation along with clear result display. This shows that prompt guidance greatly improves UI structure and usability.

Question 5:

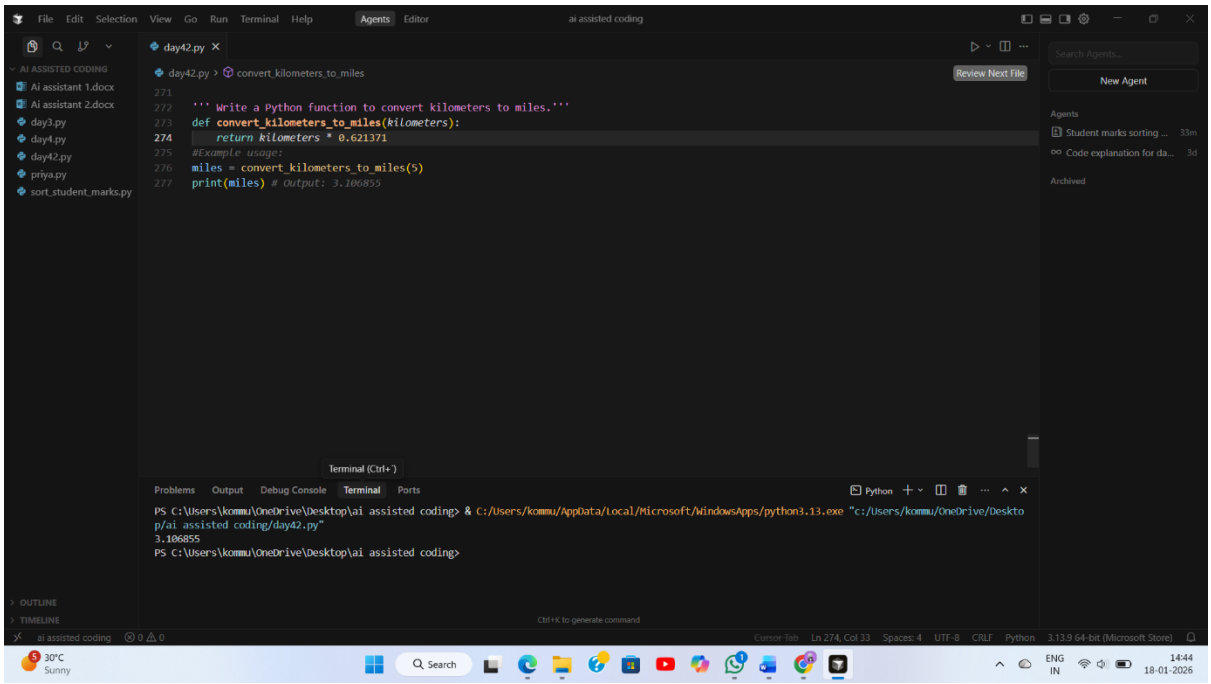
Task Description-5 • Analyzing Prompt Specificity in Unit Conversion Functions: Improving a Unit Conversion Function (Kilometers to Miles and Miles to Kilometers) Using Clear Instructions.

Expected Output-5 • Analysis of code quality and accuracy differences across multiple prompt variations.

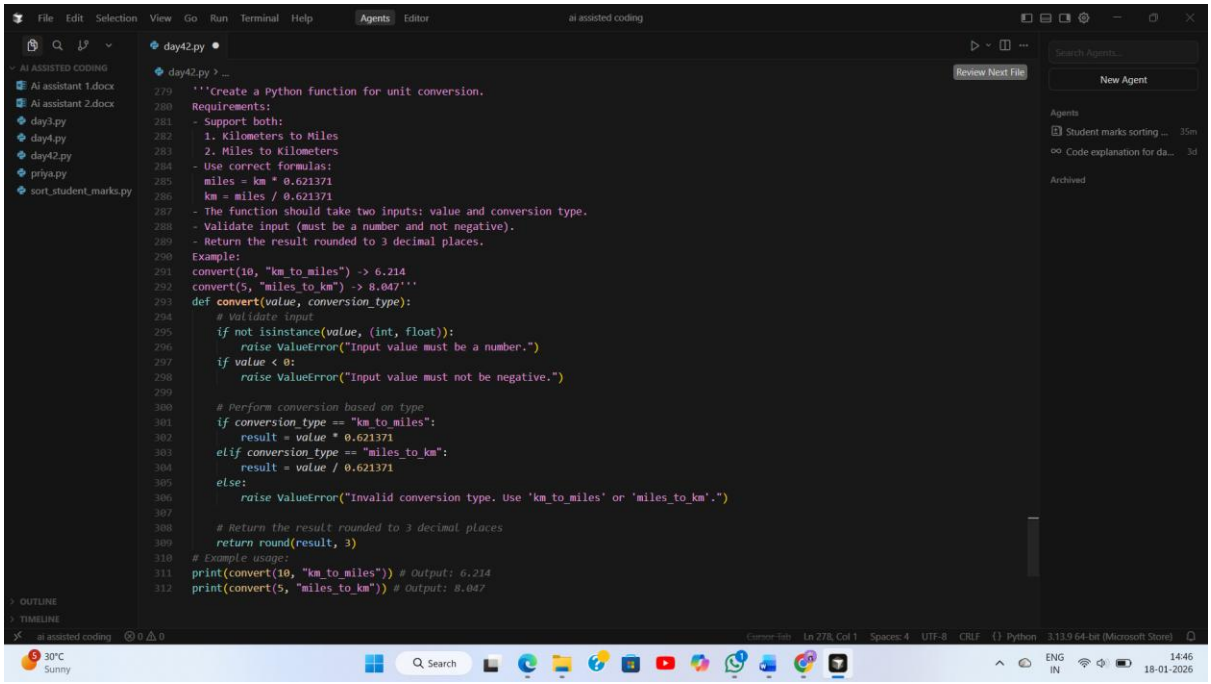
Stage1:



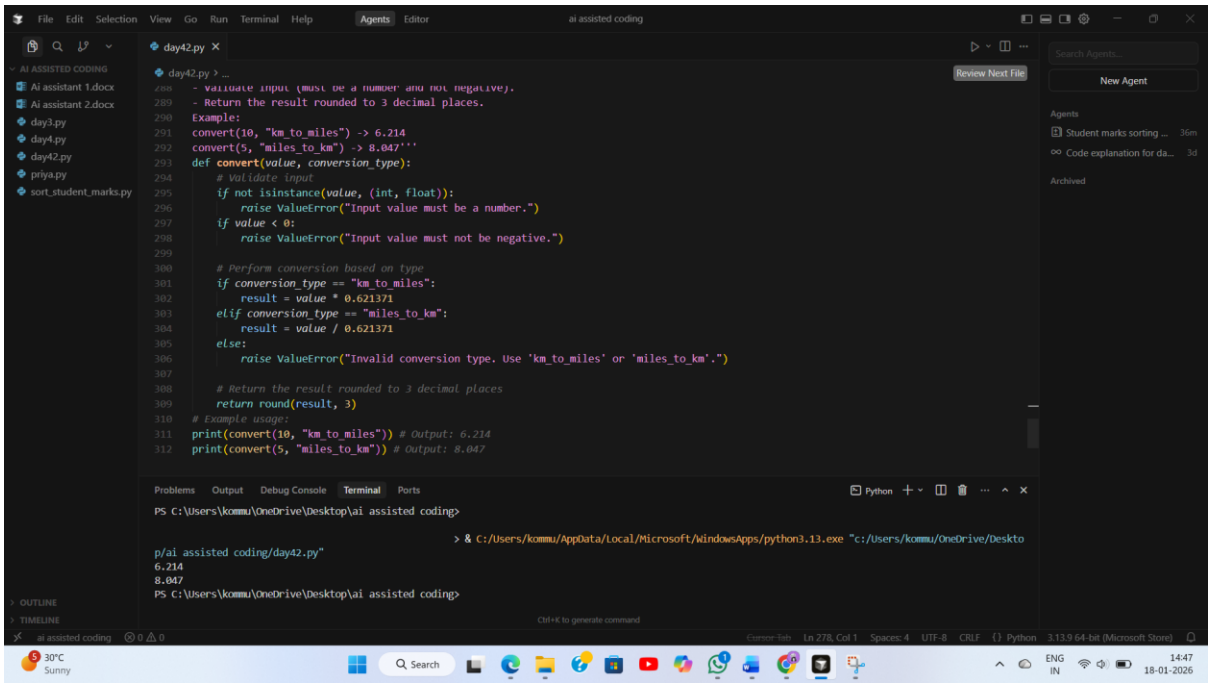
Stage2:



Stage3:



Output:



Observation:

When a vague prompt was used, the AI generated unclear or very general conversion code. After specifying the type of conversion, the AI produced a basic one-way converter. When detailed instructions, formulas, and validation rules were added, the AI generated an accurate, well-structured, and reusable unit conversion function. This proves that higher prompt specificity leads to better code quality, accuracy, and reliability.