

AI Assisted Coding

Assignment – 3.2

Name: Bandi Akiranandini

Batch:22

Hallticket:2303a51516

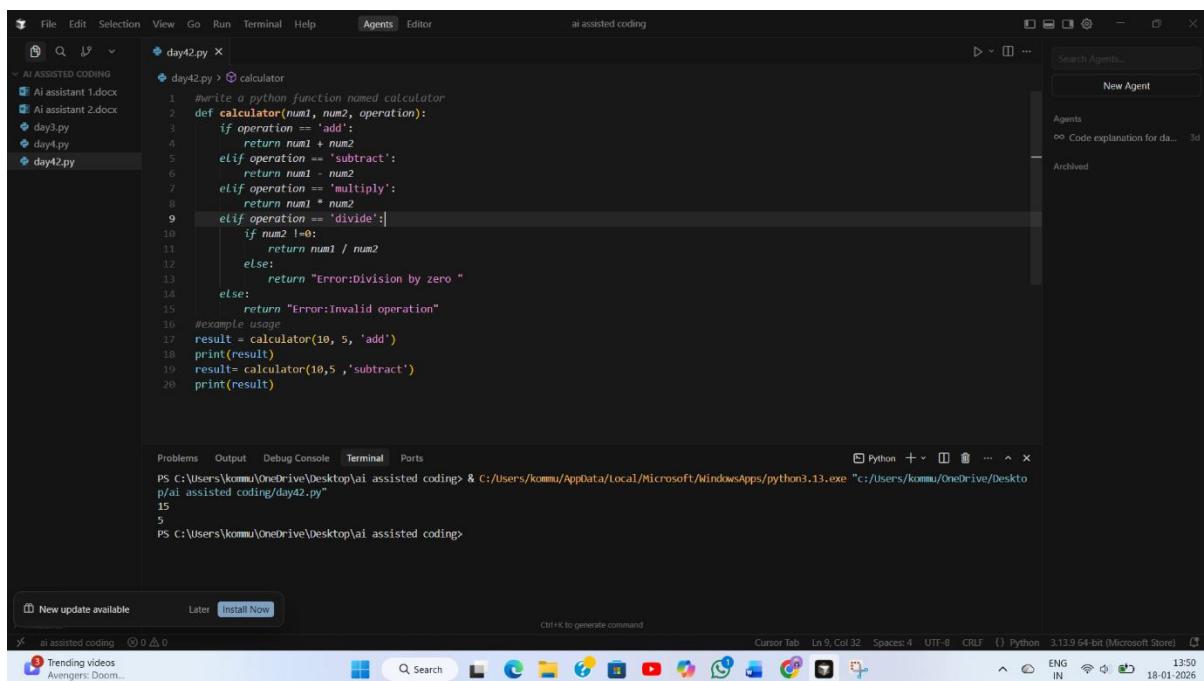
Task Description-1

- 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:



```
#write a python function named calculator
def calculator(num1, num2, operation):
    if operation == 'add':
        return num1 + num2
    elif operation == 'subtract':
        return num1 - num2
    elif operation == 'multiply':
        return num1 * num2
    elif operation == 'divide':
        if num2 != 0:
            return num1 / num2
        else:
            return "Error:Division by zero "
    else:
        return "Error:Invalid operation"
#example usage
result = calculator(10, 5, 'add')
print(result)
result= calculator(10,5 , 'subtract')
print(result)
```

PS C:\Users\kommu\OneDrive\Desktop\ai assisted coding> & C:/Users/kommu/AppData/Local/Microsoft/WindowsApps/python3.13.exe "c:/Users/kommu/OneDrive/Desktop\ai assisted coding/day42.py"
15
5
PS C:\Users\kommu\OneDrive\Desktop\ai assisted coding>

Stage 2:

A screenshot of Microsoft Visual Studio Code. The left sidebar shows a tree view with 'AI ASSISTED CODING' expanded, containing files: 'Ai assistant 1.docx', 'Ai assistant 2.docx', 'day3.py', 'day4.py', and 'day42.py'. The main editor area displays the following Python code:

```
day42.py > calculator
1  #write a python function named calculator
2  #the function should work as a simple calculator.
3  #it should take two numbers and an operator.
4  #operators: +, -, *, /
5  def calculator(num1, num2, operator):
6      if operator == '+':
7          return num1 + num2
8      elif operator == '-':
9          return num1 - num2
10     elif operator == '*':
11         return num1 * num2
12     elif operator == '/':
13         if num2 != 0:
14             return num1 / num2
15         else:
16             return "Error:Division by zero"
17     else:
18         return "Error:Invalid operation"
19
20     #example usage
21     result = calculator(10, 5, '+')
22     print(result)
23     result= calculator(10,5 ,'-')
24     print(result)
25     result= calculator(10,5 ,'*')
26     print(result)
27     result= calculator(10,5 ,'/')
28     print(result)
```

The terminal tab at the bottom shows the command 'p/ai assisted coding/day42.py' and its output: 15, 5, 50, 2.0.

Stage3:

A screenshot of Microsoft Visual Studio Code, identical to Stage 2 but with a few changes in the code. The main editor area now includes error handling for division by zero and invalid operators:

```
day42.py > calculator
1  #write a python function named calculator
2  #the function should work as a simple calculator.
3  #it should take two numbers and an operator.
4  #operators: +, -, *, /
5  #it should return the result.
6  #example:
7  #calculator(10,5, '+') -> 15
8  #calculator(10,5, '-') -> 5
9  #calculator(10,5, '*') -> 50
10  #calculator(10,5, '/') -> 2
11  #if operator is invalid ,return "invalid operator,
12  #if division by zero,return "cannot divide by zero"
13  def calculator(num1, num2, operator):
14      if operator == '+':
15          return num1 + num2
16      elif operator == '-':
17          return num1 - num2
18      elif operator == '*':
19          return num1 * num2
20      elif operator == '/':
21          if num2 != 0:
22              return num1 / num2
23          else:
24              return "cannot division by zero"
25      else:
26          return "Invalid operation"
27  #example usage
28  result = calculator(10, 5, '+')
29  print(result)
30  result= calculator(10,5 ,'-')
31  print(result)
32  result= calculator(10,5 ,'*')
33  print(result)
34  result= calculator(10,5 ,'/')
35  print(result)
```

Output:

A screenshot of the terminal window from Microsoft Visual Studio Code. It shows the command 'p/ai assisted coding/day42.py' and its output: 15, 5, 50, 2.0.

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.

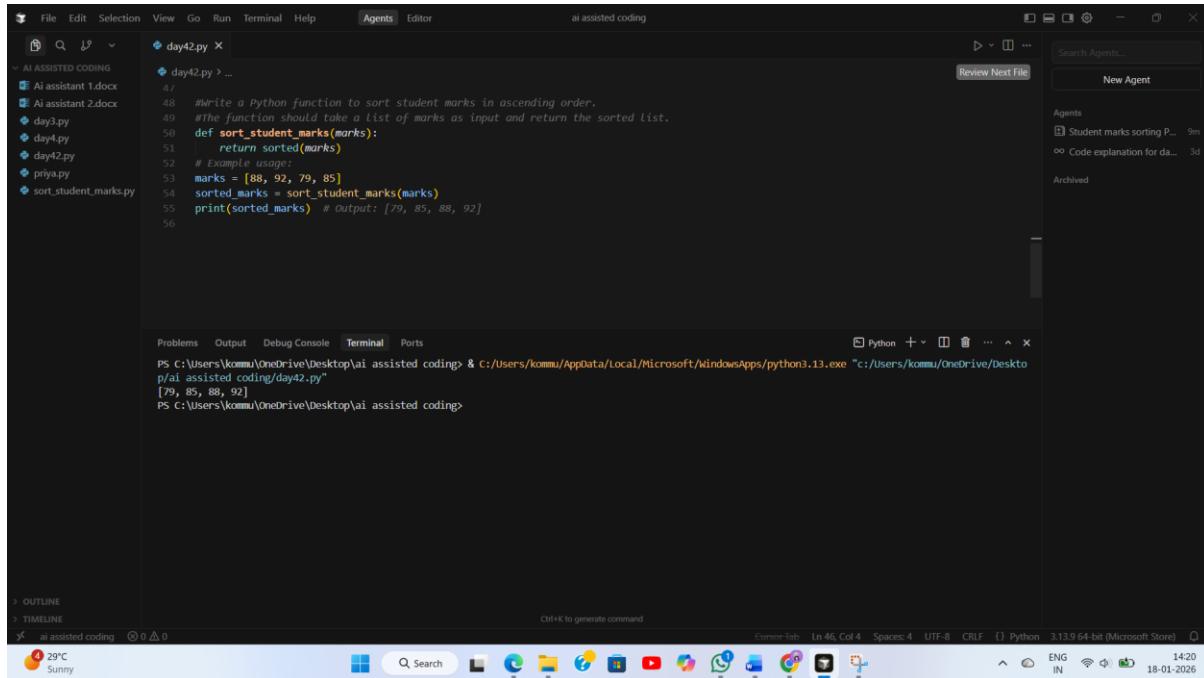
Expected Output-2 AI-generated sorting function evolves from ambiguous logic to an accurate and efficient implementation.

Stage 1:

```
# Write a Python program to sort student marks.
def sort_student_marks(marks):
    return sorted(marks)
# Example usage:
marks = [88, 92, 79, 85, 95]
sorted_marks = sort_student_marks(marks)
print(sorted_marks) # Output: [79, 85, 88, 92, 95]
```

The screenshot shows a code editor interface with a sidebar for 'AI ASSISTED CODING' containing files like 'Ai assistant 1.docx', 'Ai assistant 2.docx', 'day3.py', 'day4.py', 'day42.py', 'priya.py', and 'sort_student_marks.py'. The main area displays the content of 'day42.py'. Below the editor is a terminal window showing the command 'python day42.py' being run and its output: '[79, 85, 88, 92, 95]'. The bottom status bar indicates the file is 3.13.9 64-bit (Microsoft Store) and the date is 18-01-2026.

Stage2:

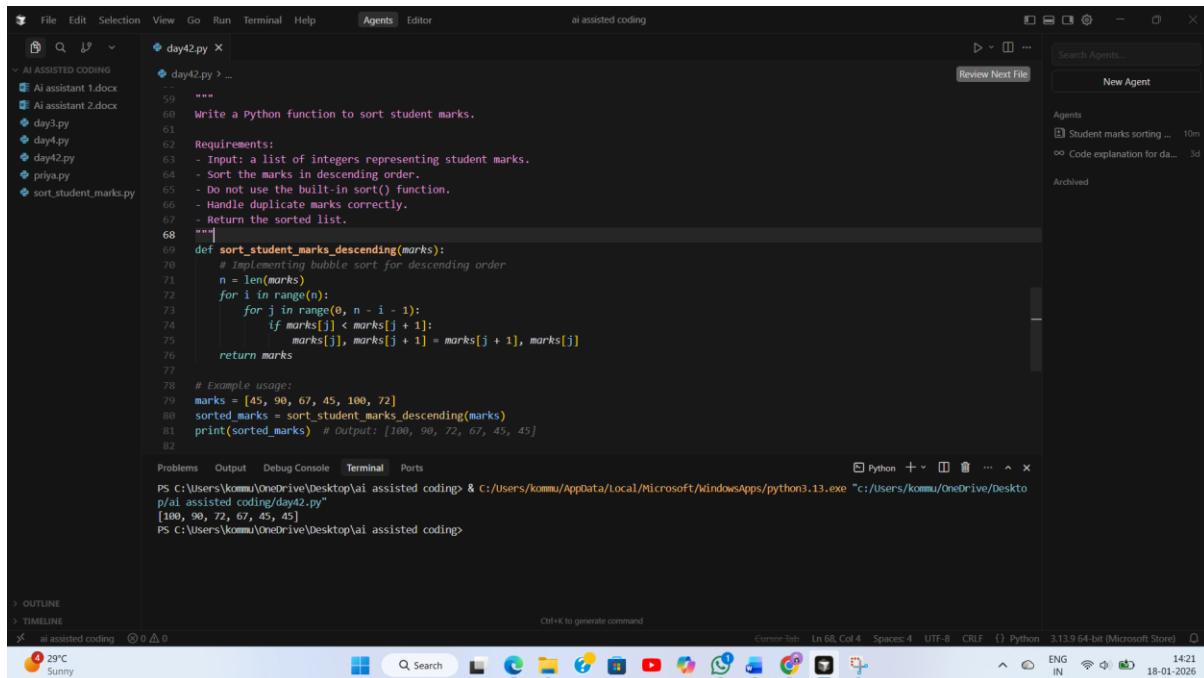


```
#Write a Python function to sort student marks in ascending order.
#The function should take a list of marks as input and return the sorted list.
def sort_student_marks(marks):
    return sorted(marks)

# Example usage:
marks = [88, 92, 79, 85]
sorted_marks = sort_student_marks(marks)
print(sorted_marks) # Output: [79, 85, 88, 92]
```

```
PS C:\Users\kommu\OneDrive\Desktop\ai assisted coding> & C:/users/kommu/AppData/Local/Microsoft/WindowsApps/python3.13.exe "c:/Users/kommu/OneDrive/Desktop\ai assisted coding/day42.py"
[79, 85, 88, 92]
PS C:\Users\kommu\OneDrive\Desktop\ai assisted coding>
```

Stage3:



```
"""
Write a Python function to sort student marks.

Requirements:
- Input: a list of integers representing student marks.
- Sort the marks in descending order.
- Do not use the built-in sort() function.
- Handle duplicate marks correctly.
- Return the sorted list.
"""

def sort_student_marks_descending(marks):
    # Implementing bubble sort for descending order
    n = len(marks)
    for i in range(n):
        for j in range(0, n - i - 1):
            if marks[j] < marks[j + 1]:
                marks[j], marks[j + 1] = marks[j + 1], marks[j]

    return marks

# Example usage:
marks = [45, 90, 67, 45, 100, 72]
sorted_marks = sort_student_marks_descending(marks)
print(sorted_marks) # Output: [100, 90, 72, 67, 45, 45]
```

```
PS C:\Users\kommu\OneDrive\Desktop\ai assisted coding> & C:/users/kommu/AppData/Local/Microsoft/WindowsApps/python3.13.exe "c:/Users/kommu/OneDrive/Desktop\ai assisted coding/day42.py"
[100, 90, 72, 67, 45, 45]
PS C:\Users\kommu\OneDrive\Desktop\ai assisted coding>
```

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

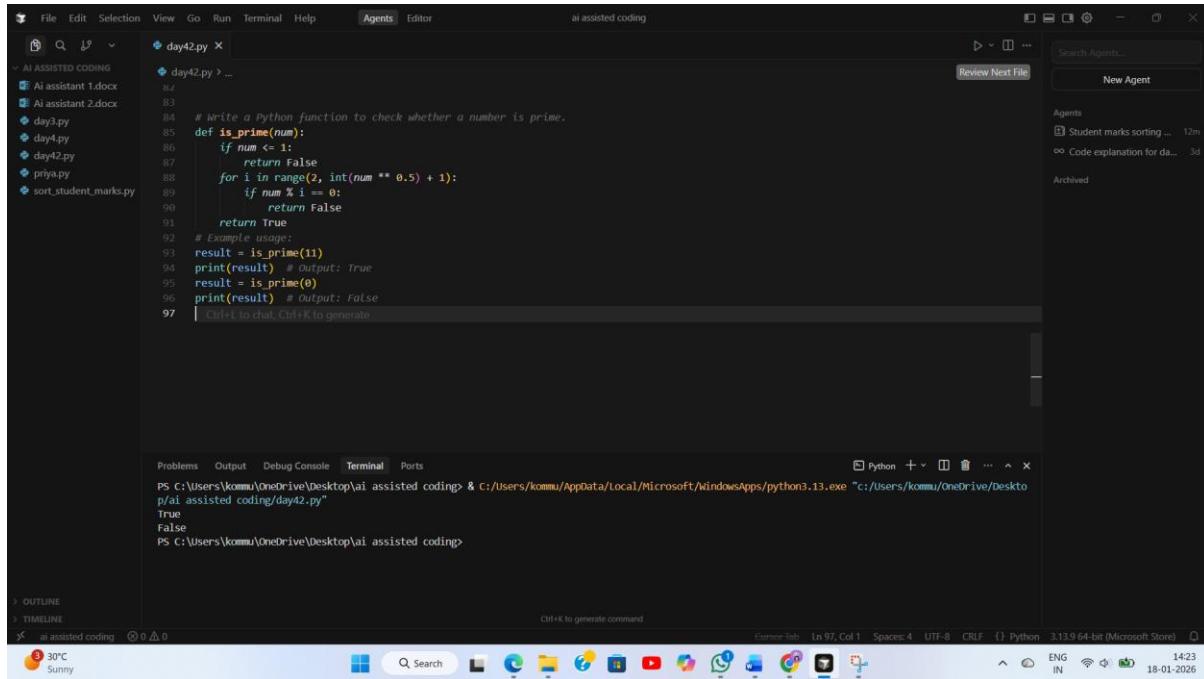
and efficient sorting function. This demonstrates that refining prompts helps the AI move from ambiguous logic to a correct and reliable implementation.

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:



The screenshot shows a code editor interface with a dark theme. At the top, there's a navigation bar with File, Edit, Selection, View, Go, Run, Terminal, Help, Agents, and Editor. Below the navigation bar is a sidebar titled "ai assisted coding" containing a file tree with files like "Ai assistant 1.docx", "Ai assistant 2.docx", "day3.py", "day4.py", "day42.py", "priya.py", and "sort_student_marks.py". The main area displays a Python script named "day42.py". The code defines a function "is_prime" that checks if a number is prime. It includes comments explaining the logic and example usages for numbers 11 and 9. The terminal tab at the bottom shows the command "python day42.py" being run, resulting in output "True" and "False" respectively. The status bar at the bottom right shows the date as 18-01-2026.

```
# Write a Python function to check whether a number is prime.
def is_prime(num):
    if num <= 1:
        return False
    for i in range(2, int(num ** 0.5) + 1):
        if num % i == 0:
            return False
    return True
# Example usage:
result = is_prime(11)
print(result) # output: True
result = is_prime(9)
print(result) # output: False
```

Stage2:

The screenshot shows the AI assisted coding interface. The code editor displays a Python script named day42.py with the following content:

```
98 # Write a Python function to check whether a number is prime.
99 def is_prime(num):
100     if num <= 1:
101         return False
102     for i in range(2, int(num ** 0.5) + 1):
103         if num % i == 0:
104             return False
105     return True
106
107 # Example usage:
108 result = is_prime(11)
109 print(result) # Output: True
110 result = is_prime(4)
111 print(result) # Output: False
```

The terminal below shows the execution of the script:

```
PS C:\Users\kommu\OneDrive\Desktop\ai assisted coding> python day42.py
True
False
PS C:\Users\kommu\OneDrive\Desktop\ai assisted coding>
```

The status bar at the bottom indicates the system is at 14:25 on 18-01-2026.

Stage3:

The screenshot shows the AI assisted coding interface. The code editor displays a Python script named day42.py with the following content:

```
113 '''Write a Python function to check whether a number is prime.
114 Examples:
115 Input: 2 -> Output: Prime
116 Input: 3 -> Output: Prime
117 Input: 4 -> Output: Not Prime
118 Input: 9 -> Output: Not Prime
119 Input: 1 -> Output: Not Prime
120 Input: 0 -> Output: Not Prime
121 Input: -7 -> Output: Not Prime
122 Input: 13 -> Output: Prime
123 The function should return "Prime" or "Not Prime".'''
124
125 def is_prime(num):
126     if num <= 1:
127         return "Not Prime"
128     for i in range(2, int(num**0.5) + 1):
129         if num % i == 0:
130             return "Not Prime"
131     return "Prime"
132
133 #Example usage:
134 result = is_prime(2)
135 print(result) # Output: Prime
136 result = is_prime(4)
137 print(result) # Output: Not Prime
```

The terminal below shows the execution of the script:

```
PS C:\Users\kommu\OneDrive\Desktop\ai assisted coding> python day42.py
Prime
Not Prime
PS C:\Users\kommu\OneDrive\Desktop\ai assisted coding>
```

The status bar at the bottom indicates the system is at 14:32 on 18-01-2026.

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:

The screenshot shows the Microsoft Visual Studio Code interface. The left sidebar displays a file tree with files like 'Ai assisted 1.docx', 'Ai assisted 2.docx', 'day3.py', 'day4.py', 'day42.py', 'priya.py', and 'sort_student_marks.py'. The main editor area contains the following Python code:

```
# Create a user interface for a student grading system.
def student_grading_system():
    print("Welcome to the Student Grading System")
    name = input("Enter student name: ")
    marks = float(input("Enter student marks (0-100): "))
    if marks < 0 or marks > 100:
        print("Error: Marks should be between 0 and 100.")
        return
    if marks >= 90:
        grade = 'A'
    elif marks >= 80:
        grade = 'B'
    elif marks >= 70:
        grade = 'C'
    elif marks >= 60:
        grade = 'D'
    else:
        grade = 'E'
    print(f"Student Name: {name}")
    print(f"Marks: {marks}")
    print(f"Grade: {grade}")

# Example usage:
student_grading_system()
```

The terminal below the editor shows the execution of the script and its output:

```
Welcome to the Student Grading System
Enter student name: madhupriya
Enter student marks (0-100): 88
Student Name: madhupriya
Marks: 88.0
Grade: B
PS C:\Users\kommu\OneDrive\Desktop\ai assisted coding>
```

The status bar at the bottom indicates the environment is Python 3.13.9 64-bit (Microsoft Store) and the date is 18-01-2026.

Stage2:

The screenshot shows the Microsoft Visual Studio Code interface, similar to Stage 1. The left sidebar displays a file tree with files like 'Ai assisted 1.docx', 'Ai assisted 2.docx', 'day3.py', 'day4.py', 'day42.py', 'priya.py', and 'sort_student_marks.py'. The main editor area contains the following Python code, which includes comments explaining the purpose and logic:

```
'''Create a simple UI-based student grading system using Python.
The user should enter marks and the program should calculate total and percentage.'''
def student_grading_system():
    print("Welcome to the Student Grading System")
    name = input("Enter student name: ")
    marks_obtained = float(input("Enter marks obtained: "))
    total_marks = float(input("Enter total marks: "))
    if marks_obtained < 0 or total_marks < 0 or marks_obtained > total_marks:
        print("Error: Invalid marks input.")
        return
    percentage = (marks_obtained / total_marks) * 100
    print(f"Student Name: {name}")
    print(f"Marks Obtained: {marks_obtained} out of {total_marks}")
    print(f"Percentage: {percentage:.2f}%")

# Example usage:
student_grading_system()
```

The terminal below the editor shows the execution of the script and its output:

```
Enter student name: kommu madhupriya
Enter marks obtained: 100
Enter total marks: 100
Student Name: kommu madhupriya
Marks Obtained: 100.0 out of 100.0
Percentage: 100.00%
PS C:\Users\kommu\OneDrive\Desktop\ai assisted coding>
```

The status bar at the bottom indicates the environment is Python 3.13.9 64-bit (Microsoft Store) and the date is 18-01-2026.

Stage3:

```
File Edit Selection View Go Run Terminal Help Agents Editor ai assisted coding
AI ASSISTED CODING
Ai assistant 1.docx
Ai assistant 2.docx
day3.py
day4.py
day42.py
priya.py
sort_student_marks.py

day42.py ...
186     '''Create a Python Tkinter-based user interface for a Student Grading System.
187     Requirements:
188     - UI should accept student name and marks of 5 subjects.
189     - Each subject is out of 100.
190     - Calculate:
191         1. Total marks
192         2. Percentage
193         3. Grade
194     Grade rules:
195     - >= 90% -> A+
196     - >= 75% -> A
197     - >= 60% -> B
198     - >= 40% -> C
199     - < 40% -> Fail
200     - Display total, percentage, and grade clearly in the UI.
201     - Show an error message if any input is invalid.'''
202 import tkinter as tk
203 from tkinter import messagebox
204 def calculate_grade():
205     try:
206         name = entry_name.get()
207         marks = [float(entry.get()) for entry in entries_marks]
208         if any(mark < 0 or mark > 100 for mark in marks):
209             raise ValueError("Marks should be between 0 and 100.")
210         total = sum(marks)
211         percentage = (total / 500) * 100
212         if percentage >= 90:
213             grade = 'A+'
214         elif percentage >= 75:
215             grade = 'A'
216         elif percentage >= 60:
217             grade = 'B'
218         elif percentage >= 40:
219             grade = 'C'
220         else:
221             grade = 'Fail'
222     except ValueError as e:
223         messagebox.showerror("Input Error", str(e))
224     result_text = f"Student Name: {name}\nTotal Marks: {total}\nPercentage: {percentage:.2f}%\nGrade: {grade}"
225     messagebox.showinfo("Result", result_text)
226
227 # Create the main window
228 root = tk.Tk()
229 root.title("Student Grading System")
230 # Student Name
231 tk.Label(root, text="Student Name:").grid(row=0, column=0)
232 entry_name = tk.Entry(root)
233 entry_name.grid(row=0, column=1)
234 # Marks for 5 subjects
235 entries_marks = []
236 for i in range(5):
237     tk.Label(root, text=f"Marks for Subject {i+1}:").grid(row=i+1, column=0)
238     entry = tk.Entry(root)
239     entry.grid(row=i+1, column=1)
240     entries_marks.append(entry)
241 # Calculate Button
242 btn_calculate = tk.Button(root, text="Calculate Grade", command=calculate_grade)
243 btn_calculate.grid(row=6, column=0, columnspan=2)
244 root.mainloop()

> OUTLINE
> TIMELINE
ai assisted coding 30°C Sunny
```

```
File Edit Selection View Go Run Terminal Help Agents Editor ai assisted coding
AI ASSISTED CODING
Ai assistant 1.docx
Ai assistant 2.docx
day3.py
day4.py
day42.py
priya.py
sort_student_marks.py

day42.py ...
284     def calculate_grade():
285         grade = 'A'
286         elif percentage >= 60:
287             grade = 'B'
288         elif percentage >= 40:
289             grade = 'C'
290         else:
291             grade = 'Fail'
292         result_text = f"Student Name: {name}\nTotal Marks: {total}\nPercentage: {percentage:.2f}%\nGrade: {grade}"
293         messagebox.showinfo("Result", result_text)
294     except ValueError as e:
295         messagebox.showerror("Input Error", str(e))
296
297 # Create the main window
298 root = tk.Tk()
299 root.title("Student Grading System")
300 # Student Name
301 tk.Label(root, text="Student Name:").grid(row=0, column=0)
302 entry_name = tk.Entry(root)
303 entry_name.grid(row=0, column=1)
304 # Marks for 5 subjects
305 entries_marks = []
306 for i in range(5):
307     tk.Label(root, text=f"Marks for Subject {i+1}:").grid(row=i+1, column=0)
308     entry = tk.Entry(root)
309     entry.grid(row=i+1, column=1)
310     entries_marks.append(entry)
311 # Calculate Button
312 btn_calculate = tk.Button(root, text="Calculate Grade", command=calculate_grade)
313 btn_calculate.grid(row=6, column=0, columnspan=2)
314 root.mainloop()

> OUTLINE
> TIMELINE
ai assisted coding 30°C Sunny
```

Output:

```
Student Name: kommu madhupriya
Total Marks: 574.0
Percentage: 74.60%
Grade: B

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()
```

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:

The screenshot shows a Microsoft Visual Studio Code (VS Code) window. The top menu bar includes File, Edit, Selection, View, Go, Run, Terminal, Help, Agents, and Editor. The title bar says "ai assisted coding". The left sidebar has sections for OUTLINE and TIMELINE, with a weather icon showing 30°C and Sunny. The main editor area contains Python code for unit conversion, specifically meters to kilometers and vice versa. The terminal below shows the code being run and outputting values like 1.0, 5000, and 32.0. The status bar at the bottom indicates Python 3.13.9 64-bit (Microsoft Store), ENG IN, and the date 18-01-2026.

```
246     '''Write a Python function to convert units.'''  
247     def convert_units(value, from_unit, to_unit):  
248         conversion_factors = {  
249             'meters_to_kilometers': 0.001,  
250             'kilometers_to_meters': 1000,  
251             'grams_to_kilograms': 0.001,  
252             'kilograms_to_grams': 1000,  
253             'celsius_to_fahrenheit': lambda c: (c * 9/5) + 32,  
254             'fahrenheit_to_celsius': lambda f: (f - 32) * 5/9,  
255         }  
256         key = f'{from_unit}.to.{to_unit}'  
257         if key in conversion_factors:  
258             factor = conversion_factors[key]  
259             if callable(factor):  
260                 return factor(value)  
261             else:  
262                 return value * factor  
263         else:  
264             raise ValueError("Conversion from {} to {} not supported.".format(from_unit, to_unit))  
265     # Example usage:  
266     if __name__ == "__main__":  
267         print(convert_units(1000, 'meters', 'kilometers')) # Output: 1.0  
268         print(convert_units(5, 'kilograms', 'grams')) # Output: 5000  
269         print(convert_units(0, 'celsius', 'fahrenheit')) # Output: 32.0
```

Terminal output:

```
> & C:/Users/kommu/AppData/Local/Microsoft/WindowsApps/python3.13.exe "c:/Users/kommu/OneDrive/Desktop/p/ai assisted coding/day42.py"  
1.0  
5000  
32.0  
PS C:/Users/kommu/OneDrive/Desktop/ai assisted coding>
```

Bottom status bar: Current Tab, Ln 245, Col 1, Spaces: 4, UTF-8, Python, 3.13.9 64-bit (Microsoft Store), ENG IN, 14:43, 18-01-2026

Stage2:

The screenshot shows a Microsoft Visual Studio Code (VS Code) window. The top menu bar includes File, Edit, Selection, View, Go, Run, Terminal, Help, Agents, and Editor. The title bar says "ai assisted coding". The left sidebar has sections for OUTLINE and TIMELINE, with a weather icon showing 30°C and Sunny. The main editor area contains Python code for converting kilometers to miles. The terminal below shows the code being run and outputting 3.106855. The status bar at the bottom indicates Python 3.13.9 64-bit (Microsoft Store), ENG IN, and the date 18-01-2026.

```
271     """ Write a Python function to convert kilometers to miles."""  
272     def convert_kilometers_to_miles(kilometers):  
273         return kilometers * 0.621371  
274     #Example usage:  
275     miles = convert_kilometers_to_miles(5)  
276     print(miles) # Output: 3.106855
```

Terminal output:

```
PS C:/Users/kommu/OneDrive/Desktop/ai assisted coding> & C:/Users/kommu/AppData/Local/Microsoft/WindowsApps/python3.13.exe "c:/Users/kommu/OneDrive/Desktop/p/ai assisted coding/day42.py"  
3.106855  
PS C:/Users/kommu/OneDrive/Desktop/ai assisted coding>
```

Bottom status bar: Current Tab, Ln 274, Col 33, Spaces: 4, UTF-8, CR/LF, Python, 3.13.9 64-bit (Microsoft Store), ENG IN, 14:44, 18-01-2026

Stage3:

The screenshot shows the AI assisted coding interface with the following details:

- File Menu:** File, Edit, Selection, View, Go, Run, Terminal, Help.
- Agents Tab:** Agents, Editor, ai assisted coding.
- Code Area:** A code editor window titled "day42.py".

```
279     '''Create a Python function for unit conversion.
280     Requirements:
281     - Support both:
282         1. Kilometers to Miles
283         2. Miles to Kilometers
284     - Use correct formulas:
285         miles = km * 0.621371
286         km = miles / 0.621371
287     - The function should take two inputs: value and conversion type.
288     - Validate input (must be a number and not negative).
289     - Return the result rounded to 3 decimal places.
290     Example:
291     convert(10, "km_to_miles") -> 6.214
292     convert(5, "miles_to_km") -> 8.047'''
293     def convert(value, conversion_type):
294         # Validate input
295         if not isinstance(value, (int, float)):
296             raise ValueError("Input value must be a number.")
297         if value < 0:
298             raise ValueError("Input value must not be negative.")
299
300         # Perform conversion based on type
301         if conversion_type == "km_to_miles":
302             result = value * 0.621371
303         elif conversion_type == "miles_to_km":
304             result = value / 0.621371
305         else:
306             raise ValueError("Invalid conversion type. Use 'km_to_miles' or 'miles_to_km'.")
307
308         # Return the result rounded to 3 decimal places
309         return round(result, 3)
310     # Example usage:
311     print(convert(10, "km_to_miles")) # Output: 6.214
312     print(convert(5, "miles_to_km")) # Output: 8.047
```
- Right Panel:** Review Next File, New Agent, Agents, Student marks sorting ..., 35m, Code explanation for da..., 3d, Archived.
- Bottom Status Bar:** Common Tabs, Ln 278, Col 1, Spaces: 4, UTF-8, CR/LF, Python, 3.13.9 64-bit (Microsoft Store), ENG IN, 14:46, 18-01-2026.

Output:

The screenshot shows the AI assisted coding interface with the following details:

- File Menu:** File, Edit, Selection, View, Go, Run, Terminal, Help.
- Agents Tab:** Agents, Editor, ai assisted coding.
- Code Area:** A code editor window titled "day42.py".

```
280     - Validate input (must be a number and not negative).
281     - Return the result rounded to 3 decimal places.
282     Example:
283     convert(10, "km_to_miles") -> 6.214
284     convert(5, "miles_to_km") -> 8.047'''
285     def convert(value, conversion_type):
286         # Validate input
287         if not isinstance(value, (int, float)):
288             raise ValueError("Input value must be a number.")
289         if value < 0:
290             raise ValueError("Input value must not be negative.")
291
292         # Perform conversion based on type
293         if conversion_type == "km_to_miles":
294             result = value * 0.621371
295         elif conversion_type == "miles_to_km":
296             result = value / 0.621371
297         else:
298             raise ValueError("Invalid conversion type. Use 'km_to_miles' or 'miles_to_km'.")
299
300         # Return the result rounded to 3 decimal places
301         return round(result, 3)
302     # Example usage:
303     print(convert(10, "km_to_miles")) # Output: 6.214
304     print(convert(5, "miles_to_km")) # Output: 8.047
```
- Terminal Tab:** Problems, Output, Debug Console, Terminal, Ports. The terminal shows the command: PS C:\Users\kommu\OneDrive\Desktop\ai assisted coding> python day42.py and the output: 6.214, 8.047.
- Right Panel:** Review Next File, New Agent, Agents, Student marks sorting ..., 36m, Code explanation for da..., 3d, Archived.
- Bottom Status Bar:** Common Tabs, Ln 278, Col 1, Spaces: 4, UTF-8, CR/LF, Python, 3.13.9 64-bit (Microsoft Store), ENG IN, 14:47, 18-01-2026.

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.