

# AI Assisted Coding

## Assignment – 3.2

Name: K.Laxman

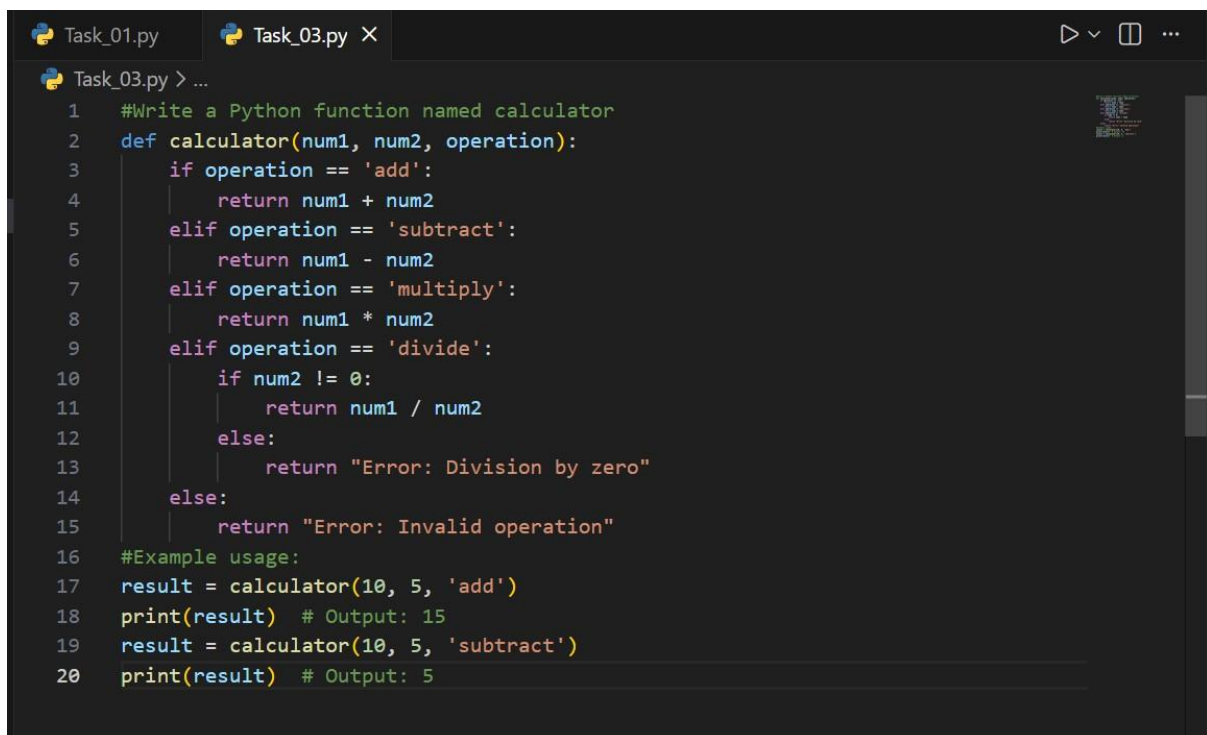
Batch: 21

HtNo: 2303A51428

**Question 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.

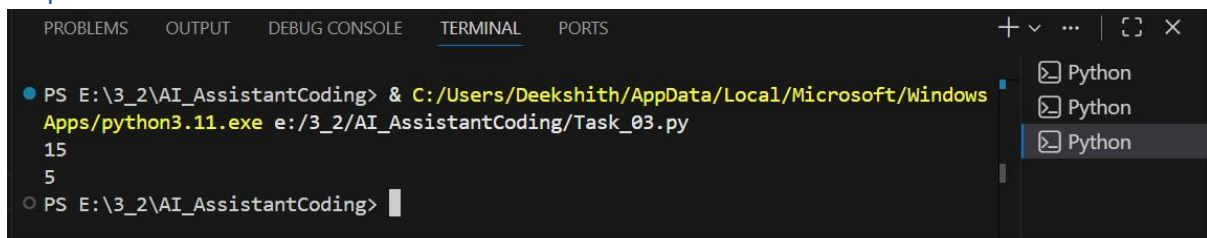
Stage 1:

Code:

A screenshot of a code editor with two tabs: 'Task\_01.py' and 'Task\_03.py'. The 'Task\_03.py' tab is active, showing a Python script. The script defines a function 'calculator' that takes three arguments: 'num1', 'num2', and 'operation'. It uses conditional logic to perform addition, subtraction, multiplication, and division, with error handling for division by zero and invalid operations. Below the function definition, there are two example usage lines: 'result = calculator(10, 5, 'add')' and 'result = calculator(10, 5, 'subtract')', each followed by a print statement to display the result.

```
Task_03.py > ...
1  #Write a Python function named calculator
2  def calculator(num1, num2, operation):
3      if operation == 'add':
4          return num1 + num2
5      elif operation == 'subtract':
6          return num1 - num2
7      elif operation == 'multiply':
8          return num1 * num2
9      elif operation == 'divide':
10         if num2 != 0:
11             return num1 / num2
12         else:
13             return "Error: Division by zero"
14     else:
15         return "Error: Invalid operation"
16 #Example usage:
17 result = calculator(10, 5, 'add')
18 print(result) # Output: 15
19 result = calculator(10, 5, 'subtract')
20 print(result) # Output: 5
```

Output:

A screenshot of a terminal window with tabs for 'PROBLEMS', 'OUTPUT', 'DEBUG CONSOLE', 'TERMINAL', and 'PORTS'. The 'TERMINAL' tab is active, showing a PowerShell prompt. The user has run the command 'C:/Users/Deekshith/AppData/Local/Microsoft/Windows Apps/python3.11.exe e:/3\_2/AI\_AssistantCoding/Task\_03.py'. The output shows the results of the two example calculations: '15' and '5'.

```
PS E:\3_2\AI_AssistantCoding> & C:/Users/Deekshith/AppData/Local/Microsoft/Windows
Apps/python3.11.exe e:/3_2/AI_AssistantCoding/Task_03.py
15
5
PS E:\3_2\AI_AssistantCoding>
```

Stage 2:

Code:

```

23
24 #Write a Python function named calculator.
25 # The function should work as a simple calculator.
26 # It should take two numbers and an operator.
27 # Operators: +, -, *, /
28 def calculator(num1, num2, operator):
29     if operator == '+':
30         return num1 + num2
31     elif operator == '-':
32         return num1 - num2
33     elif operator == '*':
34         return num1 * num2
35     elif operator == '/':
36         if num2 != 0:
37             return num1 / num2
38         else:
39             return "Error: Division by zero"
40     else:
41         return "Error: Invalid operator"
42 #Example usage:
43 result = calculator(10, 5, '+')
44 print(result) # Output: 15
45 result = calculator(10, 5, '-')
46 print(result) # Output: 5
47 result = calculator(10, 5, '*')
48 print(result) # Output: 50
49

```

Output:

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS E:\3_2\AI_AssistantCoding> & C:/Users/Deekshith/AppData/Local/Microsoft/windowsApps/python3.11.exe e:/3_2/AI_Assis
tantCoding/Task_03.py
15
5
50
PS E:\3_2\AI_AssistantCoding>

```

Stage 3:

Code:

```
Task_01.py Task_03.py X
Task_03.py > ...
54 #Write a Python function named calculator.
55 # The function should work as a simple calculator.
56 # It should take two numbers and an operator.
57 # Operators supported: +, -, *, /
58 # It should return the result.
59 # Example:
60 # calculator(10, 5, "+") → 15
61 # calculator(10, 5, "-") → 5
62 # calculator(10, 5, "**") → 50
63 # calculator(10, 5, "/") → 2
64 # If operator is invalid, return "Invalid operator"
65 # If division by zero, return "Cannot divide by zero"
66 def calculator(num1, num2, operator):
67     if operator == '+':
68         return num1 + num2
69     elif operator == '-':
70         return num1 - num2
71     elif operator == '*':
72         return num1 * num2
73     elif operator == '/':
74         if num2 != 0:
75             return num1 / num2
76         else:
77             return "Cannot divide by zero"
78     else:
79         return "Invalid operator"
80 #Example usage:
81 result = calculator(10, 5, '+')
82 print(result) # Output: 15
83 result = calculator(10, 5, '-')
84 print(result) # Output: 5
oc
```

Output:

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS E:\3_2\AI_AssistantCoding> C:\Users\DEEKSHITH\AppData\Local\Microsoft\WindowsApps\python3.11.exe E:\3_2\AI_AssistantCoding\Task_03.py
15
5
PS E:\3_2\AI_AssistantCoding>
```

### Final 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:

### Code and Output:

[illegible]

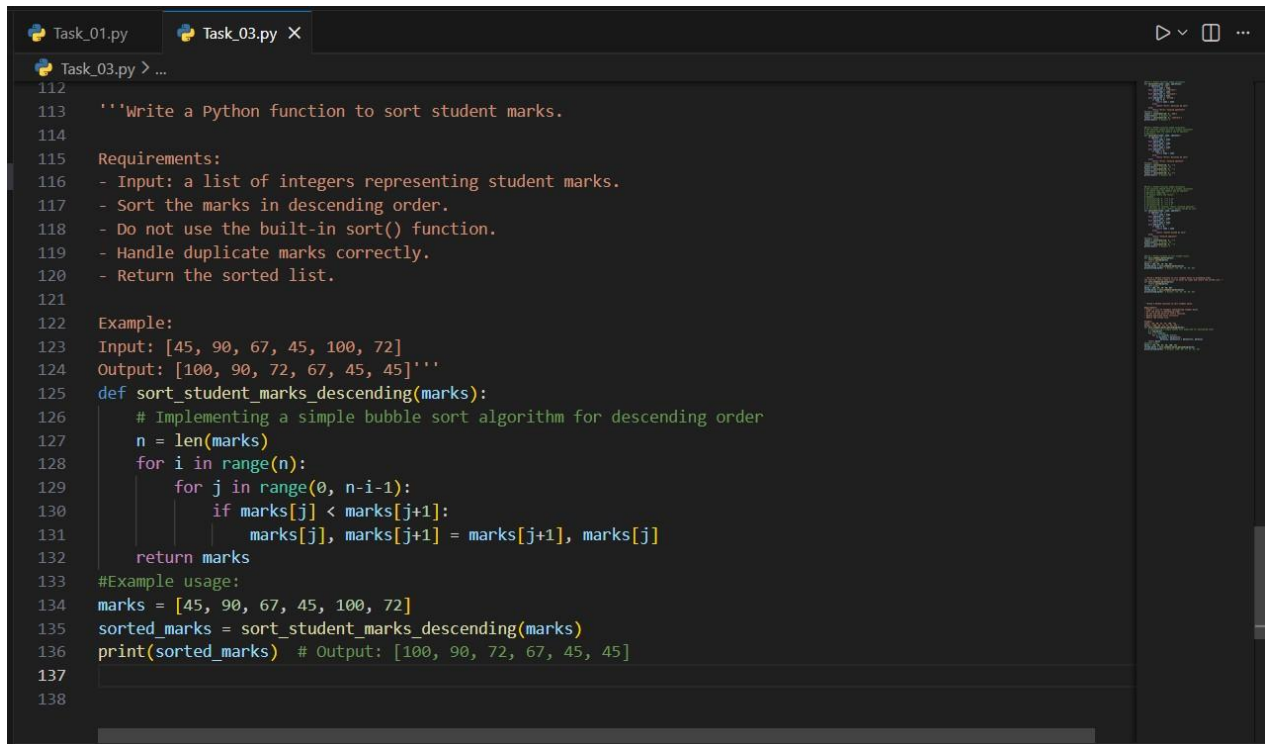
### Stage 2:

### Code and Output:

```
98
99
100 < '''Write a Python function to sort student marks in ascending order.
101     The function should take a list of marks as input and return the sorted list.'''
102 < def sort_student_marks(marks):
103     < return sorted(marks)
104
105     #Example usage:
106     marks = [88, 92] (function) def sort_student_marks(marks: Any) -> list
107     sorted_marks = sort_student_marks(marks)
108     print(sorted_marks) # Output: [79, 85, 88, 92]
```

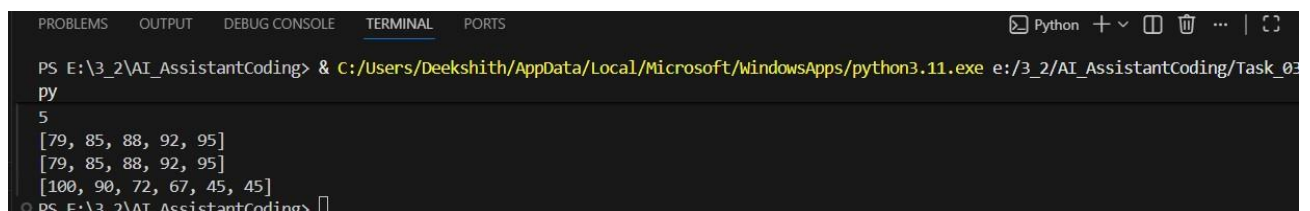
### Stage 3:

#### Code:



```
Task_01.py Task_03.py X
Task_03.py > ...
112
113 '''Write a Python function to sort student marks.
114
115 Requirements:
116 - Input: a list of integers representing student marks.
117 - Sort the marks in descending order.
118 - Do not use the built-in sort() function.
119 - Handle duplicate marks correctly.
120 - Return the sorted list.
121
122 Example:
123 Input: [45, 90, 67, 45, 100, 72]
124 Output: [100, 90, 72, 67, 45, 45]'''
125 def sort_student_marks_descending(marks):
126     # Implementing a simple bubble sort algorithm for descending order
127     n = len(marks)
128     for i in range(n):
129         for j in range(0, n-i-1):
130             if marks[j] < marks[j+1]:
131                 marks[j], marks[j+1] = marks[j+1], marks[j]
132     return marks
133 #Example usage:
134 marks = [45, 90, 67, 45, 100, 72]
135 sorted_marks = sort_student_marks_descending(marks)
136 print(sorted_marks) # Output: [100, 90, 72, 67, 45, 45]
137
138
```

#### Output:



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Python + v [Icons] | [Refresh]
PS E:\3_2\AI_AssistantCoding> & C:/Users/Deekshith/AppData/Local/Microsoft/WindowsApps/python3.11.exe e:/3_2/AI_AssistantCoding/Task_03.py
5
[79, 85, 88, 92, 95]
[79, 85, 88, 92, 95]
[100, 90, 72, 67, 45, 45]
PS E:\3_2\AI_AssistantCoding>
```

### Final 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



Output:

[illegible]

### Stage 3:

Code:

Table of Contents

Chapter 1: Introduction

Chapter 2: Literature Review

Chapter 3: Methodology

Chapter 4: Results

Chapter 5: Discussion

Chapter 6: Conclusion

References

Appendix A

Appendix B

Appendix C

Appendix D

Appendix E

Appendix F

Appendix G

Appendix H

Appendix I

Appendix J

Appendix K

Appendix L

Appendix M

Appendix N

Appendix O

Appendix P

Appendix Q

Appendix R

Appendix S

Appendix T

Appendix U

Appendix V

Appendix W

Appendix X

Appendix Y

Appendix Z

Appendix AA

Appendix AB

Appendix AC

Appendix AD

Appendix AE

Appendix AF

Appendix AG

Appendix AH

Appendix AI

Appendix AJ

Appendix AK

Appendix AL

Appendix AM

Appendix AN

Appendix AO

Appendix AP

Appendix AQ

Appendix AR

Appendix AS

Appendix AT

Appendix AU

Appendix AV

Appendix AW

Appendix AX

Appendix AY

Appendix AZ

Appendix BA

Appendix BB

Appendix BC

Appendix BD

Appendix BE

Appendix BF

Appendix BG

Appendix BH

Appendix BI

Appendix BJ

Appendix BK

Appendix BL

Appendix BM

Appendix BN

Appendix BO

Appendix BP

Appendix BQ

Appendix BR

Appendix BS

Appendix BT

Appendix BU

Appendix BV

Appendix BW

Appendix BX

Appendix BY

Appendix BZ

Appendix CA

Appendix CB

Appendix CC

Appendix CD

Appendix CE

Appendix CF

Appendix CG

Appendix CH

Appendix CI

Appendix CJ

Appendix CK

Appendix CL

Appendix CM

Appendix CN

Appendix CO

Appendix CP

Appendix CQ

Appendix CR

Appendix CS

Appendix CT

Appendix CU

Appendix CV

Appendix CW

Appendix CX

Appendix CY

Appendix CZ

Appendix DA

Appendix DB

Appendix DC

Appendix DD

Appendix DE

Appendix DF

Appendix DG

Appendix DH

Appendix DI

Appendix DJ

Appendix DK

Appendix DL

Appendix DM

Appendix DN

Appendix DO

Appendix DP

Appendix DQ

Appendix DR

Appendix DS

Appendix DT

Appendix DU

Appendix DV

Appendix DW

Appendix DX

Appendix DY

Appendix DZ

Appendix EA

Appendix EB

Appendix EC

Appendix ED

Appendix EE

Appendix EF

Appendix EG

Appendix EH

Appendix EI

Appendix EJ

Appendix EK

Appendix EL

Appendix EM

Appendix EN

Appendix EO

Appendix EP

Appendix EQ

Appendix ER

Appendix ES

Appendix ET

Appendix EU

Appendix EV

Appendix EW

Appendix EX

Appendix EY

Appendix EZ

Appendix FA

Appendix FB

Appendix FC

Appendix FD

Appendix FE

Appendix FF

Appendix FG

Appendix FH

Appendix FI

Appendix FJ

Output:

[illegible]

### Final 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

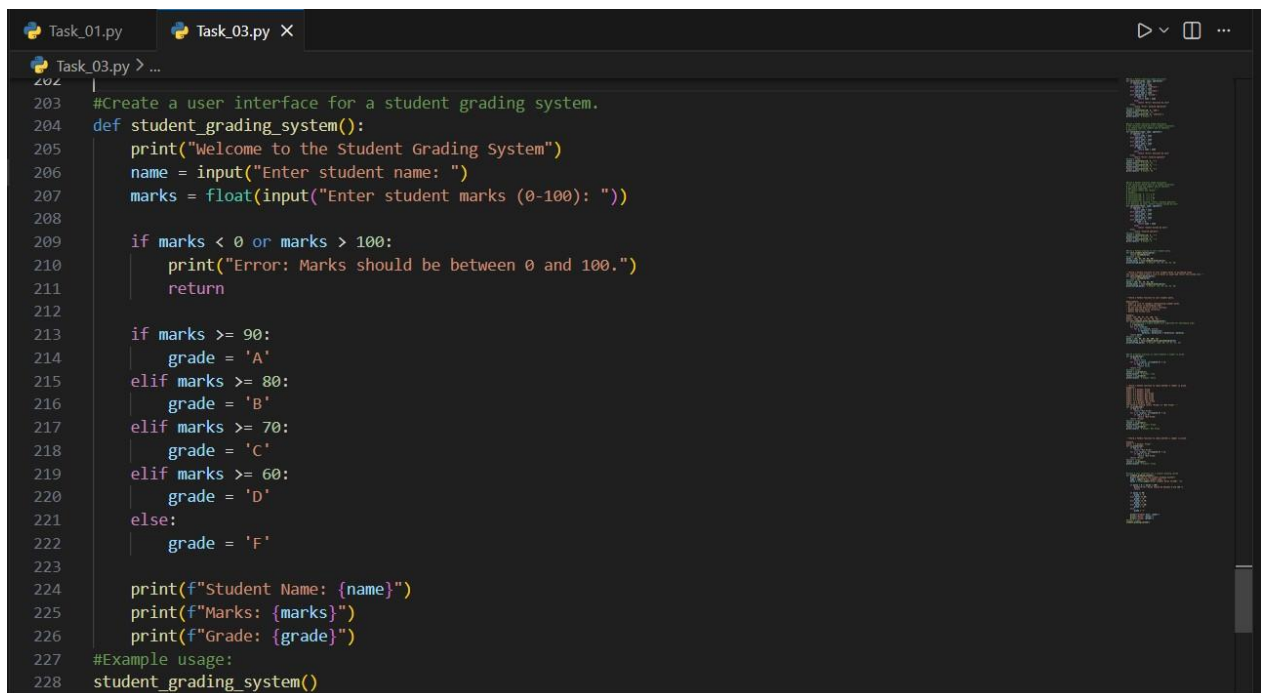
- Prompt-Guided UI Design for Student Grading System: Create a user interface for a student grading system that calculates total marks, percentage, and grade based on user input.

#### Expected Output-4

- Well-structured UI code with accurate calculations and clear output display.

#### Stage 1:

#### Code:

A screenshot of a code editor with two tabs: 'Task\_01.py' and 'Task\_03.py'. The 'Task\_03.py' tab is active, showing Python code for a student grading system. The code includes a function 'student\_grading\_system()' that prompts the user for a name and marks, validates the marks, calculates a grade, and prints the results. The code is as follows:

```
202 |  
203 | #Create a user interface for a student grading system.  
204 | def student_grading_system():  
205 |     print("Welcome to the Student Grading System")  
206 |     name = input("Enter student name: ")  
207 |     marks = float(input("Enter student marks (0-100): "))  
208 |  
209 |     if marks < 0 or marks > 100:  
210 |         print("Error: Marks should be between 0 and 100.")  
211 |         return  
212 |  
213 |     if marks >= 90:  
214 |         grade = 'A'  
215 |     elif marks >= 80:  
216 |         grade = 'B'  
217 |     elif marks >= 70:  
218 |         grade = 'C'  
219 |     elif marks >= 60:  
220 |         grade = 'D'  
221 |     else:  
222 |         grade = 'F'  
223 |  
224 |     print(f"Student Name: {name}")  
225 |     print(f"Marks: {marks}")  
226 |     print(f"Grade: {grade}")  
227 | #Example usage:  
228 | student_grading_system()
```

#### Output:



```

255
256 '''Create a Python Tkinter-based user interface for a Student Grading System.
257
258 Requirements:
259 - UI should accept student name and marks of 5 subjects.
260 - Each subject is out of 100.
261 - Calculate:
262   1. Total marks
263   2. Percentage
264   3. Grade
265
266 Grade rules:
267 - ≥ 90% → A+
268 - ≥ 75% → A
269 - ≥ 60% → B
270 - ≥ 40% → C
271 - < 40% → Fail
272
273 - Display total, percentage, and grade clearly in the UI.
274 - Show an error message if any input is invalid.'''
275 import tkinter as tk
276 from tkinter import messagebox
277 def calculate_grade():
278     try:
279         name = entry_name.get()
280         marks = [float(entry.get()) for entry in entries_marks]
281
282         if any(mark < 0 or mark > 100 for mark in marks):
283             raise ValueError("Marks should be between 0 and 100.")
284
285         total = sum(marks)
286         percentage = (total / 500) * 100
287

```

```

Task_01.py Task_03.py X
Task_03.py > ...
277 def calculate_grade():
278     if percentage >= 90:
279         grade = 'A+'
280     elif percentage >= 75:
281         grade = 'A'
282     elif percentage >= 60:
283         grade = 'B'
284     elif percentage >= 40:
285         grade = 'C'
286     else:
287         grade = 'Fail'
288
289     result_text = f"Student Name: {name}\nTotal Marks: {total}\nPercentage: {percentage:.2f}%\nGrade: {grade}"
290     messagebox.showinfo("Result", result_text)
291 except ValueError as e:
292     messagebox.showerror("Input Error", str(e))
293
294 # Create the main window
295 root = tk.Tk()
296 root.title("Student Grading System")
297 # Student Name
298 tk.Label(root, text="Student Name:").grid(row=0, column=0)
299 entry_name = tk.Entry(root)
300 entry_name.grid(row=0, column=1)
301 # Marks for 5 subjects
302 entries_marks = []
303 for i in range(5):
304     tk.Label(root, text=f"Marks for Subject {i+1}:").grid(row=i+1, column=0)
305     entry = tk.Entry(root)
306     entry.grid(row=i+1, column=1)
307     entries_marks.append(entry)
308 # Calculate Button
309 btn_calculate = tk.Button(root, text="Calculate Grade", command=calculate_grade)
310

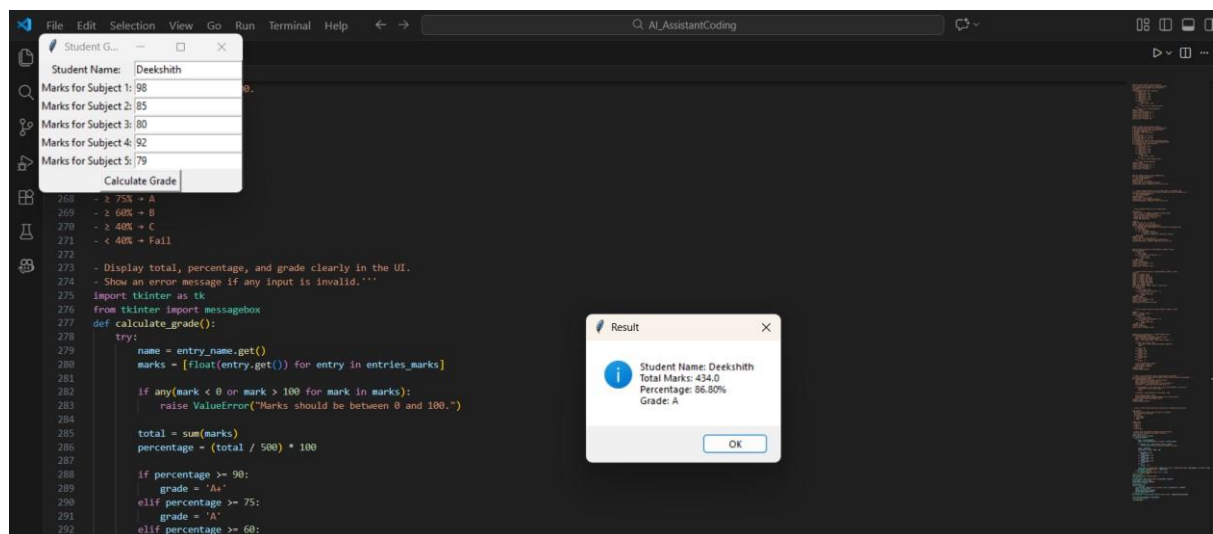
```

```

Task_01.py Task_03.py X
Task_03.py > ...
320 btn_calculate.grid(row=6, columnspan=2)
321 # Run the application
322 root.mainloop()

```

Output:



### Final 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

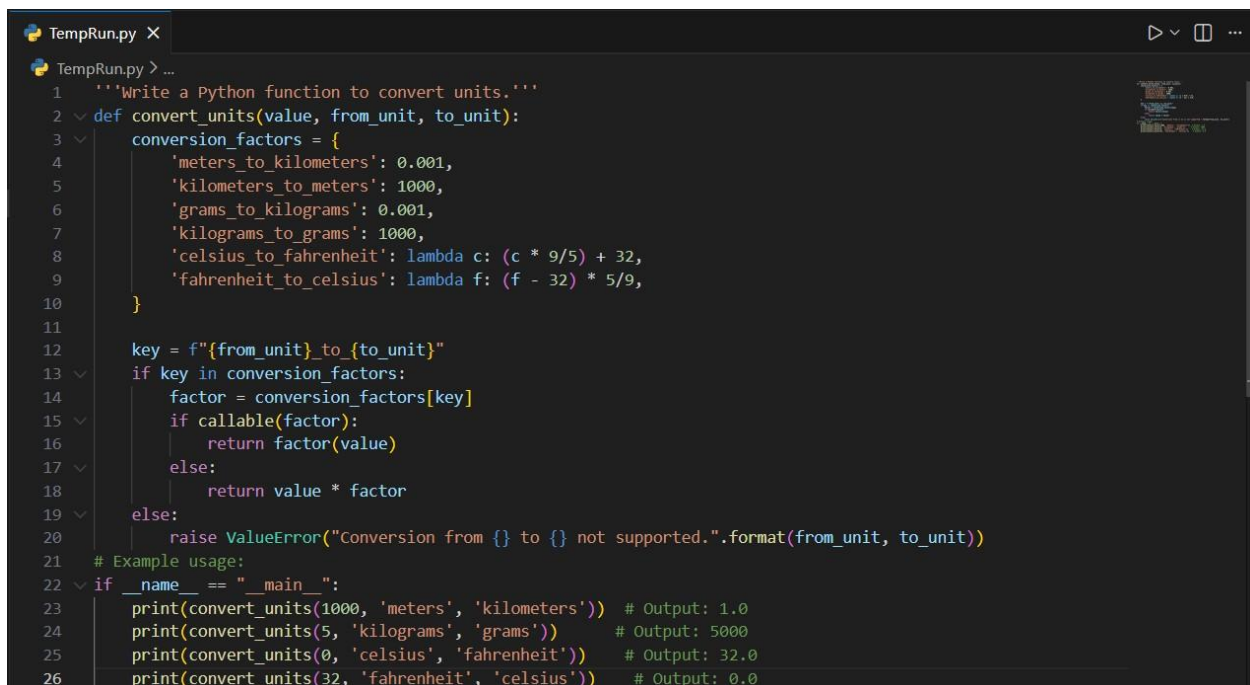
Instructions.

Expected Output-5

- Analysis of code quality and accuracy differences across multiple prompt variations.

### Stage 1:

Code:



```
TempRun.py X
TempRun.py > ...
1  '''Write a Python function to convert units.'''
2  def convert_units(value, from_unit, to_unit):
3      conversion_factors = {
4          'meters_to_kilometers': 0.001,
5          'kilometers_to_meters': 1000,
6          'grams_to_kilograms': 0.001,
7          'kilograms_to_grams': 1000,
8          'celsius_to_fahrenheit': lambda c: (c * 9/5) + 32,
9          'fahrenheit_to_celsius': lambda f: (f - 32) * 5/9,
10     }
11
12     key = f"{from_unit}_to_{to_unit}"
13     if key in conversion_factors:
14         factor = conversion_factors[key]
15         if callable(factor):
16             return factor(value)
17         else:
18             return value * factor
19     else:
20         raise ValueError("Conversion from {} to {} not supported.".format(from_unit, to_unit))
21
22     # Example usage:
23     if __name__ == "__main__":
24         print(convert_units(1000, 'meters', 'kilometers')) # Output: 1.0
25         print(convert_units(5, 'kilograms', 'grams'))     # Output: 5000
26         print(convert_units(0, 'celsius', 'fahrenheit'))  # Output: 32.0
27         print(convert_units(32, 'fahrenheit', 'celsius')) # Output: 0.0
```

Output:



