

AI Assisted Coding

Assignment – 3.2

Name: K.Deekshith

Batch: 21

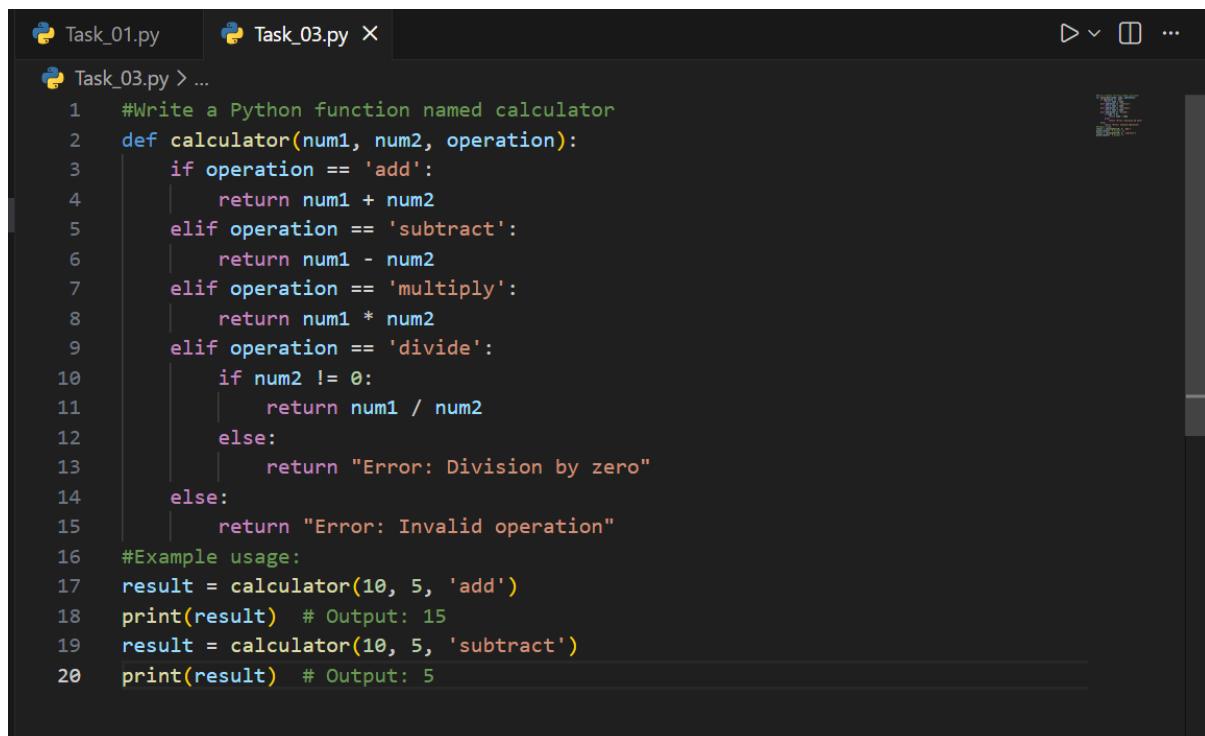
HtNo: 2303A51414

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.

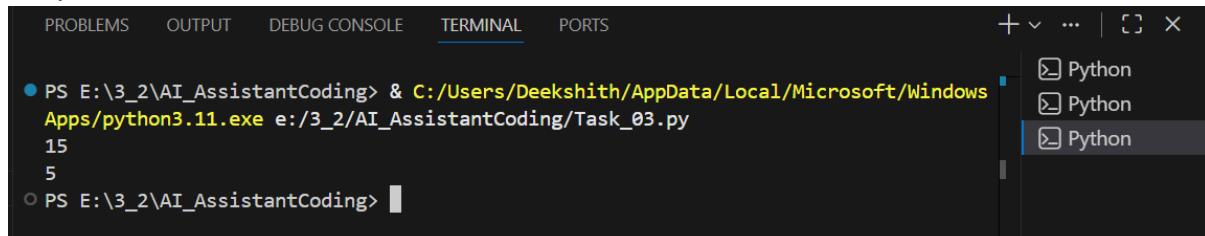
Stage 1:

Code:



```
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:



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS + ... | X
● 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>
```

Python
Python
Python

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
```

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
50

PS E:\3_2\AI_AssistantCoding> []

Ln 49, Col 1 Spaces: 4 UTF-8 CRLF { } Python 3.11.9 (Micro)

Python
Python
Python

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
85
```

Output:

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
E:\3\AI AssistantCoding> python.exe E:/3/AI AssistantCoding>
15
5
○ PS E:\3\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:

The screenshot shows a code editor with a Python script named `Task_03.py`. The code defines a function `sort_student_marks` that returns a sorted list of marks. An example usage is shown with a list [88, 92, 79, 85, 95] and the output being [79, 85, 88, 92, 95]. Below the code editor is a terminal window showing the command `python Task_03.py` and the resulting output: [79, 85, 88, 92, 95].

```
Click to add a breakpoint
87
88
89 #Write a Python program to sort student marks.
90 def sort_student_marks(marks):
91     return sorted(marks)
92 #Example usage:
93 marks = [88, 92, 79, 85, 95]
94 sorted_marks = sort_student_marks(marks)
95 print(sorted_marks) # Output: [79, 85, 88, 92, 95]

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
5
[79, 85, 88, 92, 95]
PS E:\3_2\AI_AssistantCoding>
○ PS E:\3_2\AI_AssistantCoding>
○ PS E:\3_2\AI_AssistantCoding>
```

Stage 2:

Code and Output:

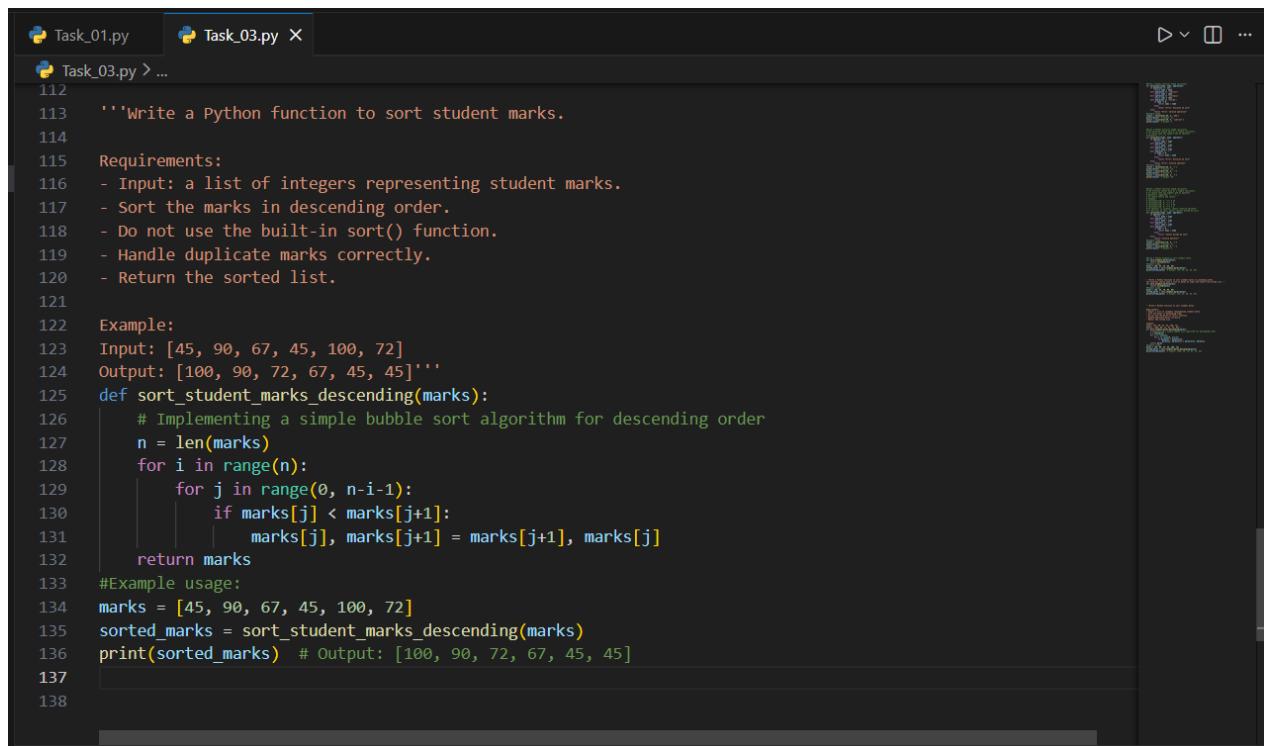
The screenshot shows the same Python script `Task_03.py` with an improved docstring. The docstring now specifies that the function takes a list of marks as input and returns a sorted list. The example usage and output remain the same as in Stage 1.

```
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 #Example usage:
105 marks = [88, 92] (function) def sort_student_marks(marks: Any) -> list
106 sorted_marks = sort_student_marks(marks)
107 print(sorted_marks) # Output: [79, 85, 88, 92]

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
[79, 85, 88, 92, 95]
[79, 85, 88, 92, 95]
○ PS E:\3_2\AI_AssistantCoding>
```

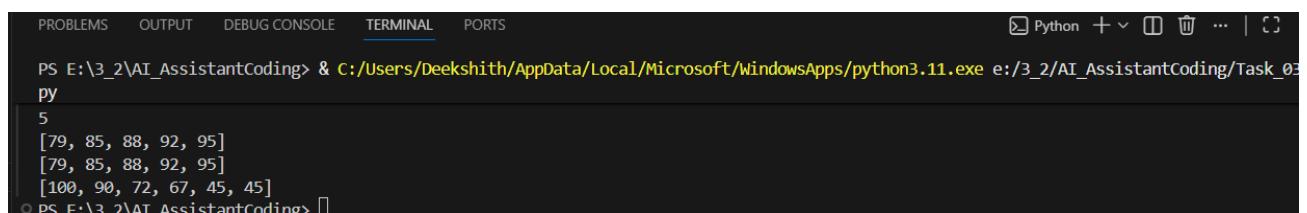
Stage 3:

Code:



```
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
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

- 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.

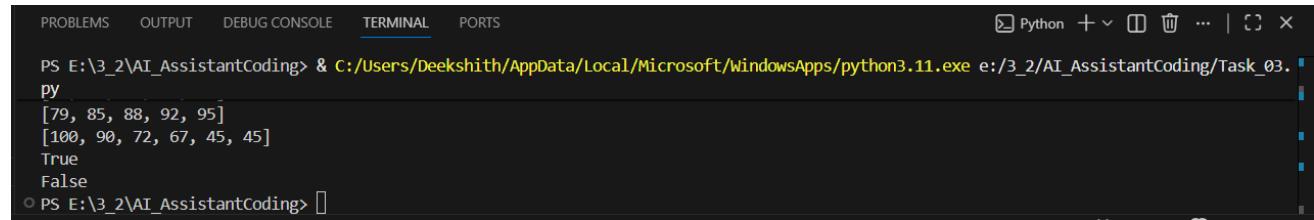
Stage 1:

Code:



```
140
141
142 #Write a Python function to check whether a number is prime.
143 def is_prime(num):
144     if num <= 1:
145         return False
146     for i in range(2, int(num**0.5) + 1):
147         if num % i == 0:
148             return False
149     return True
150 #Example usage:
151 result = is_prime(11)
152 print(result) # Output: True
153 result = is_prime(4)
154 print(result) # Output: False
155
```

Output:



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS Python + ⌂ ⌂ ... | ☰ ×
PS E:\3_2\AI_AssistantCoding> & C:/Users/Deekshith/AppData/Local/Microsoft/WindowsApps/python3.11.exe e:/3_2/AI_AssistantCoding/Task_93.py
[79, 85, 88, 92, 95]
[100, 90, 72, 67, 45, 45]
True
False
○ PS E:\3_2\AI_AssistantCoding> []
```

Stage 2:

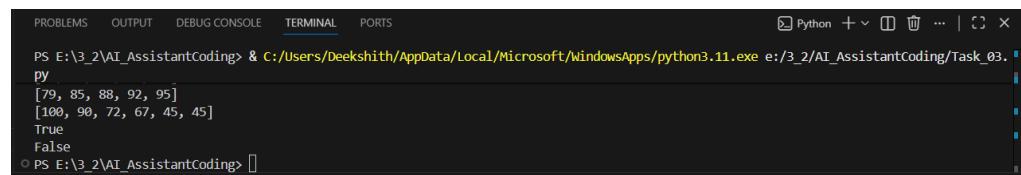
Code:

```

140
141
142 #Write a Python function to check whether a number is prime.
143 def is_prime(num):
144     if num <= 1:
145         return False
146     for i in range(2, int(num**0.5) + 1):
147         if num % i == 0:
148             return False
149     return True
150 #Example usage:
151 result = is_prime(11)
152 print(result) # Output: True
153 result = is_prime(4)
154 print(result) # Output: False
155

```

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
[79, 85, 88, 92, 95]
[100, 90, 72, 67, 45, 45]
True
False
○ PS E:\3_2\AI_AssistantCoding>

```

Stage 3:

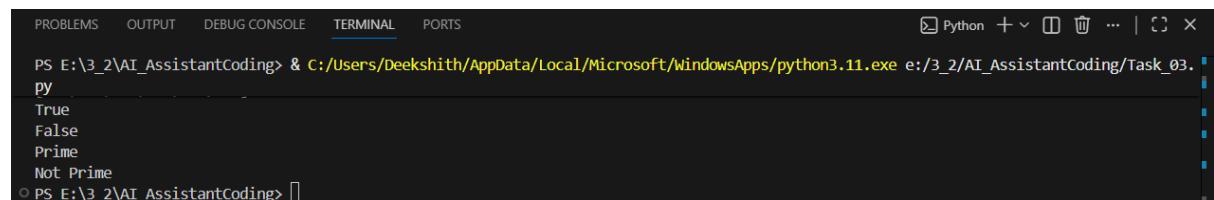
Code:

```

157 """Write a Python function to check whether a number is prime.
158 Examples:
159 Input: 2 → Output: Prime
160 Input: 3 → Output: Prime
161 Input: 4 → Output: Not Prime
162 Input: 9 → Output: Not Prime
163 Input: 1 → Output: Not Prime
164 Input: 0 → Output: Not Prime
165 Input: -7 → Output: Not Prime
166 Input: 13 → Output: Prime
167 The function should return "Prime" or "Not Prime"."""
168 def is_prime(num):
169     if num <= 1:
170         return "Not Prime"
171     for i in range(2, int(num**0.5) + 1):
172         if num % i == 0:
173             return "Not Prime"
174     return "Prime"
175 #Example usage:
176 result = is_prime(2)
177 print(result) # Output: Prime
178 result = is_prime(4)
179 print(result) # Output: Not Prime
180
181

```

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
True
False
Prime
Not Prime
○ PS E:\3_2\AI_AssistantCoding>

```

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

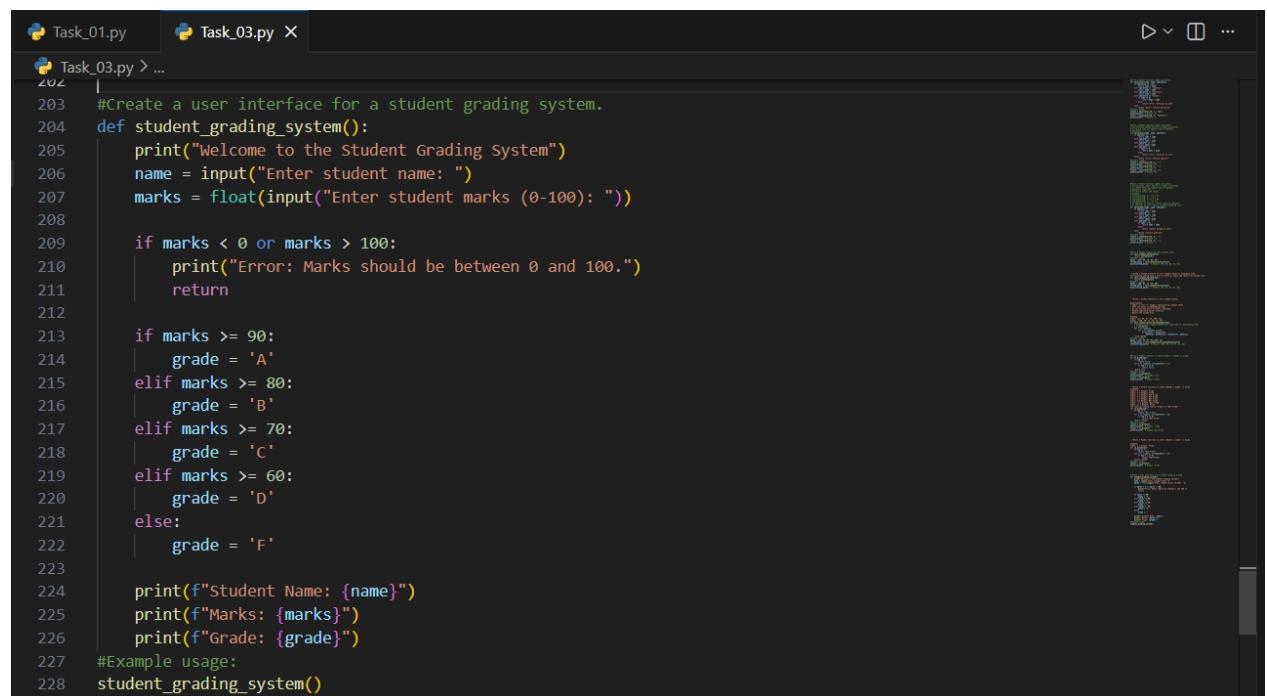
- Prompt-Guided UI Design for Student Grading System: Create a user interface for a student grading system that calculates total marks, percentage, and grade based on user input.

Expected Output-4

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

Stage 1:

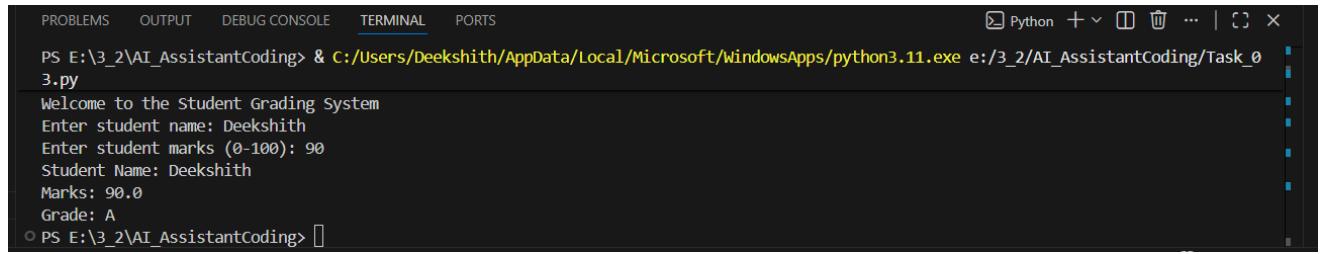
Code:



The screenshot shows a code editor window with two tabs: "Task_01.py" and "Task_03.py X". The "Task_03.py" tab is active and contains the following Python code:

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

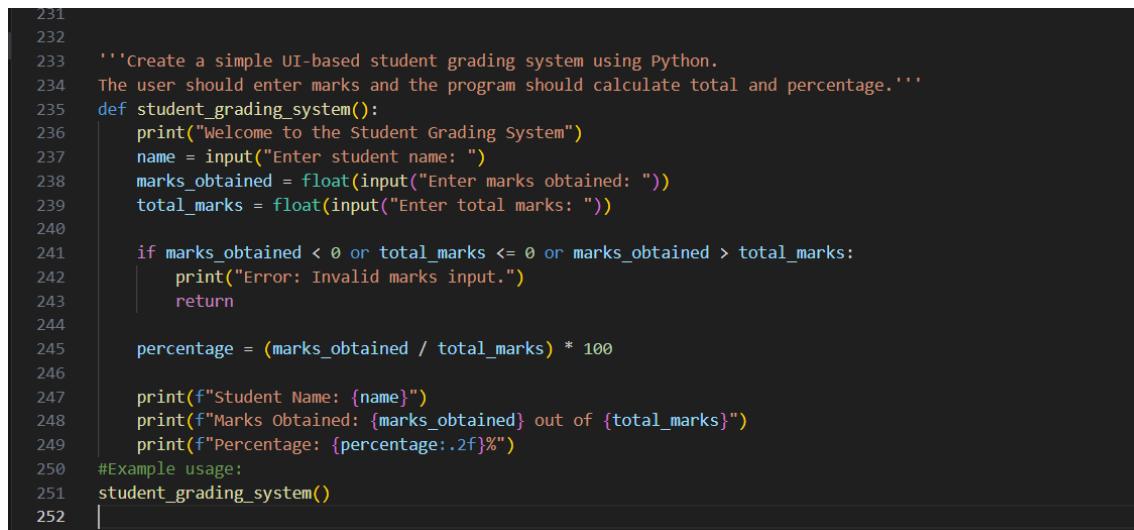
Output:



```
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
Welcome to the Student Grading System
Enter student name: Deekshith
Enter student marks (0-100): 90
Student Name: Deekshith
Marks: 90.0
Grade: A
○ PS E:\3_2\AI_AssistantCoding>
```

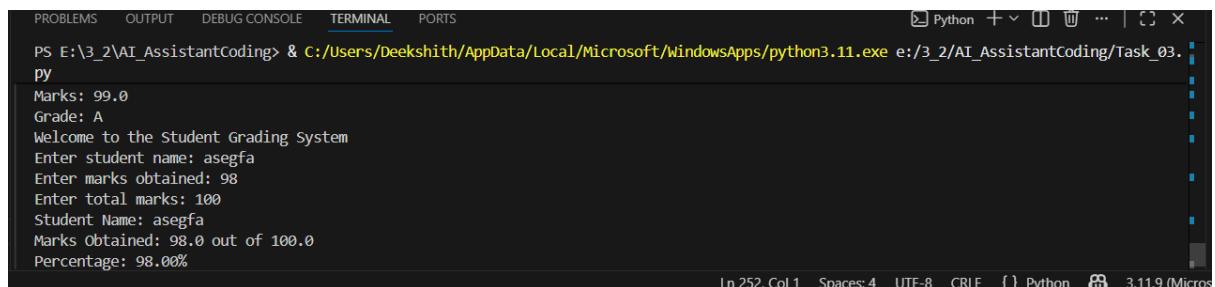
Stage 2:

Code:



```
231
232
233     '''Create a simple UI-based student grading system using Python.
234     The user should enter marks and the program should calculate total and percentage.'''
235 def student_grading_system():
236     print("Welcome to the Student Grading System")
237     name = input("Enter student name: ")
238     marks_obtained = float(input("Enter marks obtained: "))
239     total_marks = float(input("Enter total marks: "))
240
241     if marks_obtained < 0 or total_marks <= 0 or marks_obtained > total_marks:
242         print("Error: Invalid marks input.")
243         return
244
245     percentage = (marks_obtained / total_marks) * 100
246
247     print(f"Student Name: {name}")
248     print(f"Marks Obtained: {marks_obtained} out of {total_marks}")
249     print(f"Percentage: {percentage:.2f}%")
250 #Example usage:
251 student_grading_system()
252 |
```

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
Marks: 99.0
Grade: A
Welcome to the Student Grading System
Enter student name: asegf
Enter marks obtained: 98
Enter total marks: 100
Student Name: asegf
Marks Obtained: 98.0 out of 100.0
Percentage: 98.00%
```

Stage 3:

Code:



The screenshot shows a Python IDE interface with two tabs open: 'Task_01.py' and 'Task_03.py'. The 'Task_03.py' tab is active and displays the following Python code:

```
Task_01.py Task_03.py <...>

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



```
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:

```

Student Name: Deekshith
Marks for Subject 1: 98
Marks for Subject 2: 85
Marks for Subject 3: 80
Marks for Subject 4: 92
Marks for Subject 5: 79

Calculate Grade

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
288         if percentage >= 90:
289             grade = "A+"
290         elif percentage >= 75:
291             grade = "A"
292         elif percentage >= 60:
293             grade = "B"
294         else:
295             grade = "C"
296
297         messagebox.showinfo("Result", f"Student Name: {name}\nTotal Marks: {total}\nPercentage: {percentage:.2f}%\nGrade: {grade}")
298     except Exception as e:
299         messagebox.showerror("Error", str(e))
300
301

```

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

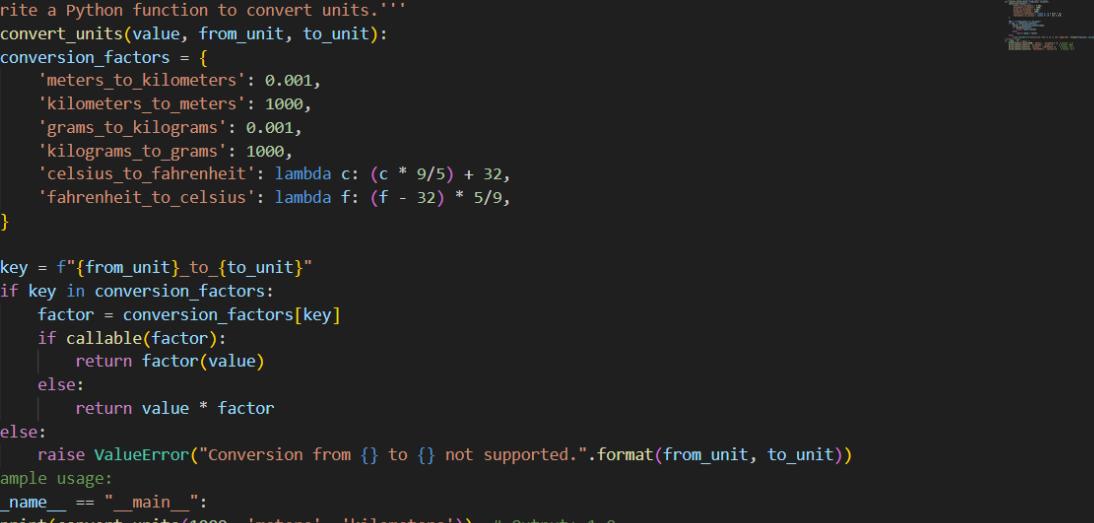
- Analyzing Prompt Specificity in Unit Conversion Functions: Improving a Unit Conversion Function (Kilometers to Miles and Miles to Kilometers) Using Clear Instructions.

Expected Output-5

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

Stage 1:

Code:



```
TempRun.py > ...
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 # Example usage:
22 if __name__ == "__main__":
23     print(convert_units(1000, 'meters', 'kilometers')) # Output: 1.0
24     print(convert_units(5, 'kilograms', 'grams')) # Output: 5000
25     print(convert_units(0, 'celsius', 'fahrenheit')) # Output: 32.0
26     print(convert_units(32, 'fahrenheit', 'celsius')) # Output: 0.0
```

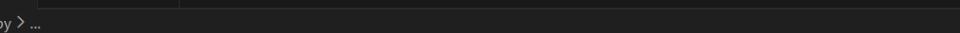
Output:

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS  Python + ⌂ ⌂ ⌂ | ⌂ X

Prime
PS E:\3_2\AI_AssistantCoding> ^C
PS E:\3_2\AI_AssistantCoding> & C:/Users/Deekshit/AppData/Local/Microsoft/WindowsApps/python3.11.exe e:\3_2\AI_AssistantCoding\TempRun.py
1.0
5000
32.0
0.0
PS E:\3_2\AI_AssistantCoding>
```

Stage 2:

Code:



```
TempRun.py > ...
1  ''' Write a Python function to convert kilometers to miles.'''
2  def convert_kilometers_to_miles(kilometers):
3      return kilometers * 0.621371
4  #Example usage:
5  miles = convert_kilometers_to_miles(5)
6  print(miles) # Output: 3.106855
```

Output:

```
PS E:\3_2\AI_AssistantCoding> & C:/Users/Deekshith/AppData/Local/Microsoft/WindowsApps/python3.11.exe e:/3_2/AI_AssistantCoding/TmpRun.py  
3.106855  
PS E:\3_2\AI_AssistantCoding>
```

Stage 3:

Code:

```
TempRun.py X Task_03.py
TempRun.py > ...
1  '''Create a Python function for unit conversion.
2
3  Requirements:
4  - Support both:
5      1. Kilometers to Miles
6      2. Miles to Kilometers
7  - Use correct formulas:
8      miles = km * 0.621371
9      km = miles / 0.621371
10 - The function should take two inputs: value and conversion type.
11 - Validate input (must be a number and not negative).
12 - Return the result rounded to 3 decimal places.
13
14 Example:
15 convert(10, "km_to_miles") → 6.214
16 convert(5, "miles_to_km") → 8.047'''
17 def convert(value, conversion_type):
18     # Validate input
19     if not isinstance(value, (int, float)):
20         raise ValueError("Input value must be a number.")
21     if value < 0:
22         raise ValueError("Input value must not be negative.")
23
24     # Perform conversion based on type
25     if conversion_type == "km_to_miles":
26         result = value * 0.621371
27     elif conversion_type == "miles_to_km":
28         result = value / 0.621371
29     else:
30         raise ValueError("Invalid conversion type. Use 'km_to_miles' or 'miles_to_km'.")
31
32     # Return the result rounded to 3 decimal places
33     return round(result, 3)
34 # Example usage:
35 print(convert(10, "km_to_miles")) # Output: 6.214
36 print(convert(5, "miles_to_km")) # Output: 8.047
```

Output:

Final 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.