

AI Assisted Coding

Assignment – 3.2

Name: A. JEEVAN SAI

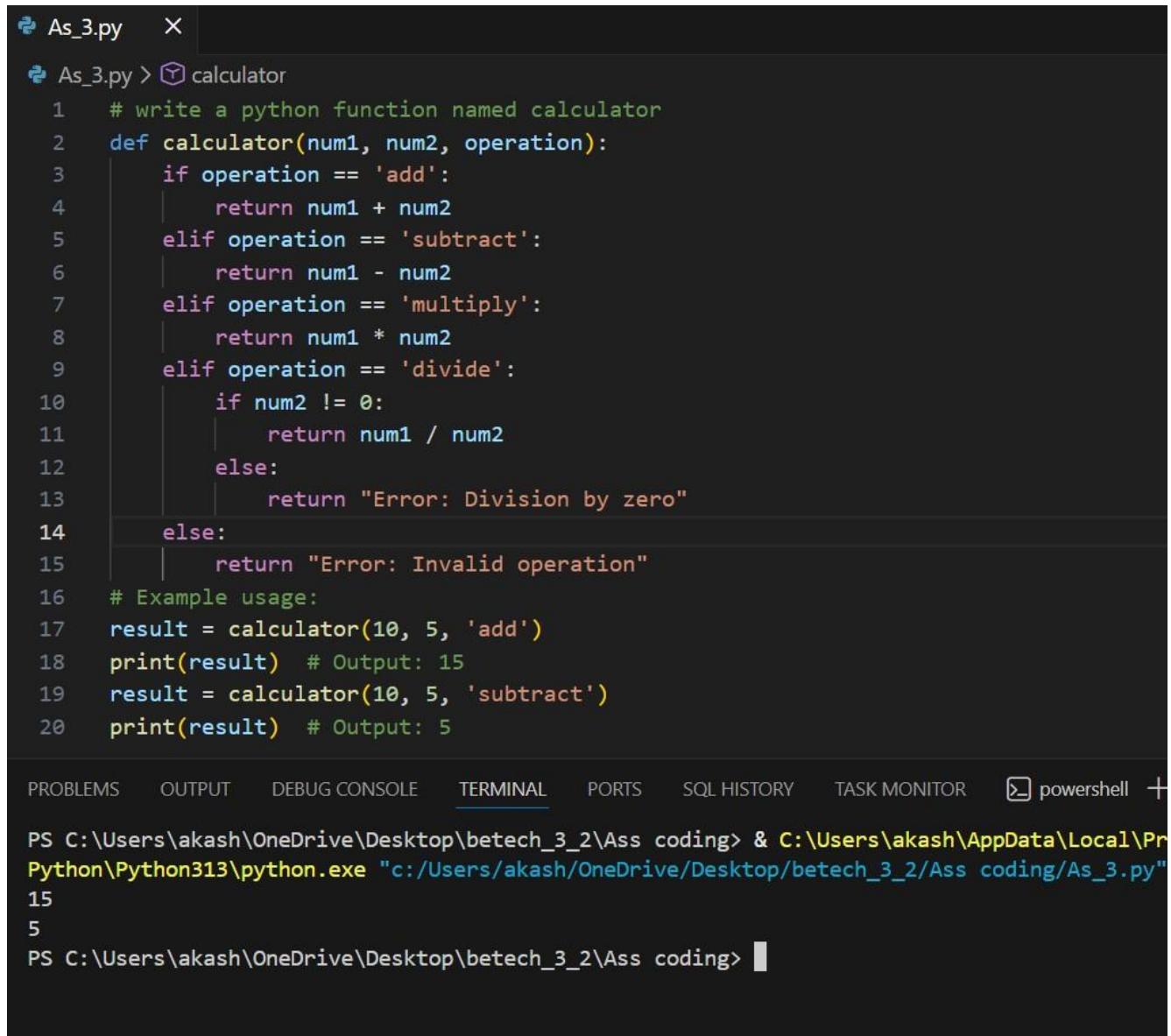
Batch: 21

HtNo: 2303A51420

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:



```
As_3.py  X
As_3.py > calculator
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
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL SQL HISTORY TASK MONITOR powershell +

```
PS C:\Users\akash\OneDrive\Desktop\betech_3_2\Ass coding> & C:\Users\akash\AppData\Local\Programs\Python\Python313\python.exe "c:/Users/akash/OneDrive/Desktop/betech_3_2/Ass coding/As_3.py"
15
5
PS C:\Users\akash\OneDrive\Desktop\betech_3_2\Ass coding>
```

Output:

A screenshot of a terminal window from a development environment like VS Code. The window has tabs at the top: PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL (which is underlined), PORTS, SQL HISTORY, TASK MONITOR, and powershell. The main area shows a command-line session:

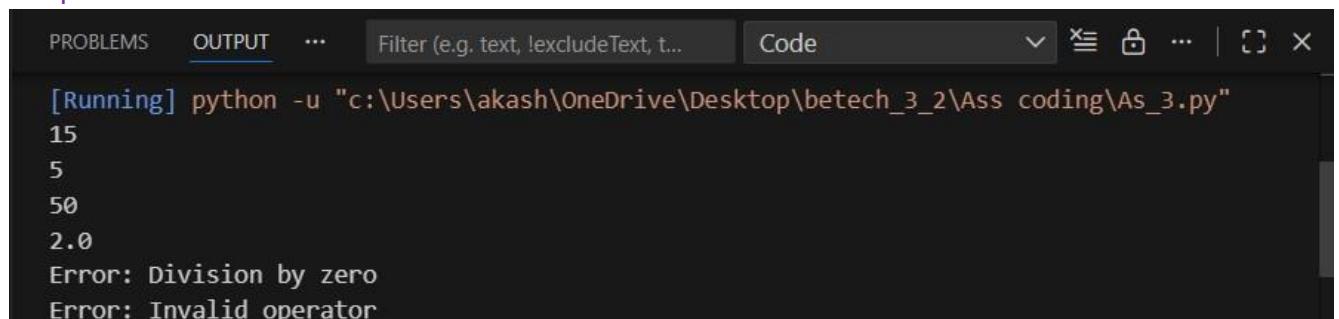
```
PS C:\Users\akash\OneDrive\Desktop\betech_3_2\Ass coding> & C:\Users\akash\AppData\Local\Python\Python313\python.exe "c:/Users/akash/OneDrive/Desktop/betech_3_2/Ass coding/As_3.py"
15
5
PS C:\Users\akash\OneDrive\Desktop\betech_3_2\Ass coding>
```

Stage 2:

Code:

```
Click to add a breakpoint
23 # write a python function named calculator
24 # the function should work as a simple calculator
25 # it should take two numbers and an operator
26 # the operator can be +, -, *, /
27 def calculator(num1, num2, operator):
28     if operator == '+':
29         return num1 + num2
30     elif operator == '-':
31         return num1 - num2
32     elif operator == '*':
33         return num1 * num2
34     elif operator == '/':
35         if num2 != 0:
36             return num1 / num2
37         else:
38             return "Error: Division by zero"
39     else:
40         return "Error: Invalid operator"
41 # Example usage:
42 result = calculator(10, 5, '+')
43 print(result) # Output: 15
44 result = calculator(10, 5, '-')
45 print(result) # Output: 5
46 result = calculator(10, 5, '*')
47 print(result) # Output: 50
48 result = calculator(10, 5, '/')
49 print(result) # Output: 2.0
50 result = calculator(10, 0, '/')
51 print(result) # Output: Error: Division by zero
52 result = calculator(10, 5, '%')
53 print(result) # Output: Error: Invalid operator
54
```

Output:



The screenshot shows a code editor interface with a dark theme. At the top, there are tabs for PROBLEMS, OUTPUT (which is underlined), ..., Filter (e.g. text, !excludeText, t...), Code, and a set of icons. The main area displays the following text output:

```
[Running] python -u "c:\Users\akash\OneDrive\Desktop\betech_3_2\Ass coding\As_3.py"
15
5
50
2.0
Error: Division by zero
Error: Invalid operator
```

Stage 3:

Code:

```
57 # write a python function named calculator
58 # the function should work as a simple calculator
59 # it should take two numbers and an operator
60 # the operator can be +, -, *, /
61 # it should return the result of the operation
62 # Example usage:
63 # calculator(10, 5, '+') should return 15
64 # calculator(10, 5, '-') should return 5
65 # calculator(10, 5, '*') should return 50
66 # calculator(10, 5, '/') should return 2.0
67 def calculator(num1, num2, operator):
68     if operator == '+':
69         return num1 + num2
70     elif operator == '-':
71         return num1 - num2
72     elif operator == '*':
73         return num1 * num2
74     elif operator == '/':
75         if num2 != 0:
76             return num1 / num2
77         else:
78             return "Error: Division by zero"
79     else:
80         return "Error: Invalid operator"
81 # Example usage:
82 result = calculator(10, 5, '+')
83 print(result) # Output: 15
84 result = calculator(10, 5, '-')
85 print(result) # Output: 5
86 result = calculator(10, 5, '*')
87 print(result) # Output: 50
88 result = calculator(10, 5, '/')
89 print(result) # Output: 2.0
90 result = calculator(10, 0, '/')
91 print(result) # Output: Error: Division by zero
92 result = calculator(10, 5, '%')
93 print(result) # Output: Error: Invalid operator
94
```

Output:

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS ... powershell + × ⌂ ⚡ ...  
Local\Programs\Python\Python313\python.exe "c:/Users/akash/OneDrive/Desktop/betech_3_2/Ass coding/As_3.py"  
15  
5  
50  
2.0
```

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 :

```
94  
95 # write a python program to sort student marks  
96 def sort_student_marks(marks):  
97     return sorted(marks)  
98 # Example usage:  
99 student_marks = [88, 92, 79, 85, 95]  
100 sorted_marks = sort_student_marks(student_marks)  
101 print(sorted_marks) # Output: [79, 85, 88, 92, 95]  
102  
103
```

Output :

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS ... powershell + v ⌂ ⌂ ...

```
Local\Programs\Python\Python313\python.exe "c:/Users/akash/OneDrive/Desktop/betech_3_2/Ass coding/As_3.py"
[79, 85, 88, 92, 95]
PS C:\Users\akash\OneDrive\Desktop\betech_3_2\Ass coding> & C:\Users\akash\AppData\
```

Stage 2:

Code :

```
103
104     # write a python functions to sort student marks in ascending order.
105     # the function should take a list of marks as input and return the sorted list.
106     def sort_student_marks(marks):
107         return sorted(marks)
108
109     # Example usage:
110     student_marks = [88, 92, 79, 85, 95]
111     sorted_marks = sort_student_marks(student_marks)
112     print(sorted_marks) # output: [79, 85, 88, 92, 95]
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
```

Out put :

```
Local\Programs\Python\Python313\python.exe "c:/Users/akash/OneDrive/Desktop/betech_3_2/Ass coding/As_3.py"
[79, 85, 88, 92, 95]
PS C:\Users\akash\OneDrive\Desktop\betech_3_2\Ass coding>
```

Stage 3:

Code:

```
114
115     # write a python functions to sort student marks.
116     # implement a simple bubble sort algorithm to sort the marks in descending order.
117     def sort_student_marks_descending(marks):
118         n = len(marks)
119         for i in range(n):
120             for j in range(0, n-i-1):
121                 if marks[j] < marks[j+1]:
122                     marks[j], marks[j+1] = marks[j+1], marks[j]
123
124     return marks
125
126     # Example usage:
127     student_marks = [88, 92, 79, 85, 95]
128     sorted_marks = sort_student_marks_descending(student_marks)
129     print(sorted_marks) # Output: [95, 92, 88, 85, 79]
130
131
```

Output:

```
Local\Programs\Python\Python313\python.exe "c:/Users/akash/OneDrive/Desktop/betech_3_2/Ass coding/As_3.py"
[95, 92, 88, 85, 79]
PS C:\Users\akash\OneDrive\Desktop\betech_3_2\Ass coding> █
```

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:

```
Local\Programs\Python\Python313\python.exe "c:/Users/akash/OneDrive/Desktop/betech_3_2/Ass coding/As_3.py"
True
False
PS C:\Users\akash\OneDrive\Desktop\betech_3_2\Ass coding> █
```

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:

```
Local\Programs\Python\Python313\python.exe "c:/Users/akash/OneDrive/Desktop/betech_3_2/Ass coding/As_3.py"
True
False
PS C:\Users\akash\OneDrive\Desktop\betech_3_2\Ass coding> █
```

Stage 3:

Code:

```
144
145 # write a python function to check whether a number is prime.
146 # examples:
147 # input : 2 output : prime
148 # input : 3 output : prime
149 # input : 4 output : not prime
150 # input : 5 output : prime
151 # input : 6 output : not prime
152 # input : 7 output : prime
153 # input : 8 output : not prime
154 def is_prime(num):
155     if num <= 1:
156         return "not prime"
157     for i in range(2, int(num**0.5) + 1):
158         if num % i == 0:
159             return "not prime"
160     return "prime"
161 # Example usage:
162 result = is_prime(11)
163 print(result) # Output: prime
164 result = is_prime(4)
165 print(result) # Output: not prime
166 result = is_prime(2)
167 print(result) # Output: prime
168 result = is_prime(9)
169 print(result) # Output: not prime
170 result = is_prime(13)
171 print(result) # Output: prime
172 result = is_prime(1)
173 print(result) # Output: not prime
174 result = is_prime(0)
175 print(result) # Output: not prime
176 result = is_prime(-5)
177 print(result) # Output: not prime
178
```

Output:

```
n\Python313\python.exe "c:/Users/akash/OneDrive/Desktop/betech_3_2/Ass coding/As_3.py"
prime
not prime
prime
not prime
prime
not prime
not prime
not prime
not prime
PS C:\Users\akash\OneDrive\Desktop\betech_3_2\Ass coding> █
```

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:

```

179 # create a user interface for a student grading system
180 def student_grading_system():
181     def calculate_grade(marks):
182         if marks >= 90:
183             return 'A'
184         elif marks >= 80:
185             return 'B'
186         elif marks >= 70:
187             return 'C'
188         elif marks >= 60:
189             return 'D'
190         else:
191             return 'F'
192
193     print("Welcome to the Student Grading System")
194     name = input("Enter student name: ")
195     try:
196         marks = float(input("Enter student marks (0-100): "))
197         if 0 <= marks <= 100:
198             grade = calculate_grade(marks)
199             print(f"Student Name: {name}")
200             print(f"Marks: {marks}")
201             print(f"Grade: {grade}")
202         else:
203             print("Error: Marks should be between 0 and 100.")
204     except ValueError:
205         print("Error: Invalid input. Please enter numeric marks.")
206     # Example usage:
207     student_grading_system()
208

```

Output:

```

\Programs\Python\Python313\python.exe "c:/Users/akash/OneDrive/Desktop/betech_3_2/Ass coding/As_3.py"
Welcome to the Student Grading System
Enter student name: Akash
Enter student marks (0-100): 91
Student Name: Akash
Marks: 91.0
Grade: A
PS C:\Users\akash\OneDrive\Desktop\betech_3_2\Ass coding>

```

Stage 2:

Code:

```

# create a simple user interface for a 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: ")
    try:
        marks_obtained = float(input("Enter marks obtained: "))
        total_marks = float(input("Enter total marks: "))
        if 0 <= marks_obtained <= total_marks:
            percentage = (marks_obtained / total_marks) * 100
            print(f"Student Name: {name}")
            print(f"Marks Obtained: {marks_obtained}")
            print(f"Total Marks: {total_marks}")
            print(f"Percentage: {percentage:.2f}%")
        else:
            print("Error: Marks obtained should be between 0 and total marks.")
    except ValueError:
        print("Error: Invalid input. Please enter numeric values for marks.")
    # Example usage:
student_grading_system()

```

Output:

```

PS C:\Users\akash\OneDrive\Desktop\betech_3_2\Ass coding> & C:\Users\akash\AppData\Local\Programs\Python313\python.exe "c:/Users/akash/OneDrive/Desktop/betech_3_2/Ass coding/As_3.py"
Welcome to the Student Grading System
Enter student name: Akash
Enter marks obtained: 98
Enter total marks: 100
Student Name: Akash
Marks Obtained: 98.0
Total Marks: 100.0
Percentage: 98.00%
PS C:\Users\akash\OneDrive\Desktop\betech_3_2\Ass coding> █

```

Stage 3:

Code:

```

# create a python tkinter user interface for a student grading system

# requirements:
# ui should accept student name and marks of 5 subjects
# each subject marks should be out of 100
# on submit it should calculate total marks,percentage and grade

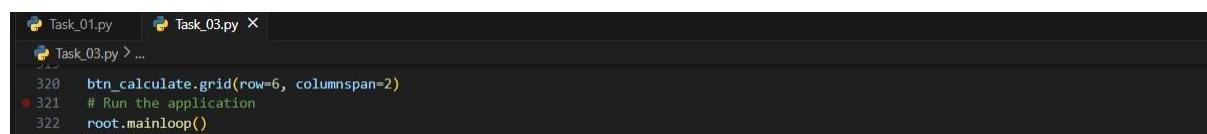
# grade rules:
# 90-100 : A
# 80-89  : B
# 70-79  : C
# 60-69  : D
# below 60 : F

```

```

245 import tkinter as tk
246 from tkinter import messagebox
247 def calculate_grade(percentage):
248     if percentage >= 90:
249         return 'A'
250     elif percentage >= 80:
251         return 'B'
252     elif percentage >= 70:
253         return 'C'
254     elif percentage >= 60:
255         return 'D'
256     else:
257         return 'F'
258 def submit():
259     try:
260         name = entry_name.get()
261         marks = [float(entry.get()) for entry in entries_marks]
262         if any(mark < 0 or mark > 100 for mark in marks):
263             messagebox.showerror("Error", "Marks should be between 0 and 100.")
264             return
265         total_marks = sum(marks)
266         percentage = (total_marks / 500) * 100
267         grade = calculate_grade(percentage)
268         result = f"Student Name: {name}\nTotal Marks: {total_marks}\nPercentage: {percentage:.2f}%\nGrade: {grade}"
269         messagebox.showinfo("Result", result)
270     except ValueError:
271         messagebox.showerror("Error", "Invalid input. Please enter numeric values for marks.")
272 root = tk.Tk()
273 root.title("Student Grading System")
274 tk.Label(root, text="Student Name:").grid(row=0, column=0)
275 entry_name = tk.Entry(root)
276 entry_name.grid(row=0, column=1)
277 entries_marks = []
278 for i in range(5):
279     tk.Label(root, text=f"Subject {i+1} Marks:").grid(row=i+1, column=0)
280     entry = tk.Entry(root)
281     entry.grid(row=i+1, column=1)
282     entries_marks.append(entry)
283 tk.Button(root, text="Submit", command=submit).grid(row=6, columnspan=2)
284 root.mainloop()
285

```



```

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

```

Output:

The screenshot shows a Python application window with two main components: a code editor and a graphical user interface (GUI) window.

Code Editor:

```
As_3.py
As_3.py > ...
223     print(f"Percentage: {percentage:.2f}%"")
224 else:
225     print("Error: Marks obtained should be between 0 and total marks.")
226 except ValueError:
227     print("Error: Invalid input. Please enter numeric values for marks.")
228 # Example usage:
229 student_grading_system()
230
231
232 # create a python tkinter user interface for a student grading system
233
234 # requirements:
235 # ui should accept student name and marks of 5 subjects
236 # each subject marks should be out of 100
237 # on submit it should calculate total marks,percentage and grade
238
239 # grade rules:
240 # 90-100 : A
241 # 80-89 : B
242 # 70-79 : C
243 # 60-69 : D
244 # below 60 : F
245 import tkinter as tk
246 from tkinter import messagebox
247 def calculate_grade(percentage):
248     if percentage >= 90:
249         return 'A'
250     elif percentage >= 80:
251         return 'B'
252     elif percentage >= 70:
253         return 'C'
254     else:
255         return 'F'.
```

Graphical User Interface (GUI) Window:

A window titled "Student..." displays a form with five text input fields for "Student Name" (Akash), "Subject 1 Marks" (98), "Subject 2 Marks" (90), "Subject 3 Marks" (84), "Subject 4 Marks" (78), and "Subject 5 Marks" (70). A "Submit" button is located at the bottom right of the form.

A second window titled "Result" displays the calculated results: "Student Name: Akash", "Total Marks: 420.0", "Percentage: 84.00%", and "Grade: B". An "OK" button is at the bottom right of this window.

Terminal:

```
PS C:\Users\akash\OneDrive\Desktop\betech_3_2\Ass coding> & C:\Users\akash\AppData\Local\Programs\Python\Python313\python.exe "c:/Users/akash/OneDrive/Desktop/betech_3_2/Ass coding/As
```

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:

```
285 # write a python function to convert units
286 def convert_units(value, from_unit, to_unit):
287     conversion_factors = {
288         'meters_to_kilometers': 0.001,
289         'kilometers_to_meters': 1000,
290         'grams_to_kilograms': 0.001,
291         'kilograms_to_grams': 1000,
292         'celsius_to_fahrenheit': lambda c: (c * 9/5) + 32,
293         'fahrenheit_to_celsius': lambda f: (f - 32) * 5/9
294     }
295     key = f"{from_unit}_to_{to_unit}"
296     if key in conversion_factors:
297         factor = conversion_factors[key]
298         if callable(factor):
299             return factor(value)
300         else:
301             return value * factor
302     else:
303         return "Error: Invalid conversion"
304 # Example usage:
305 result = convert_units(1000, 'meters', 'kilometers')
306 print(result) # Output: 1.0
307 result = convert_units(5, 'kilograms', 'grams')
308 print(result) # Output: 5000
309 result = convert_units(100, 'celsius', 'fahrenheit')
310 print(result) # Output: 212.0
311 result = convert_units(212, 'fahrenheit', 'celsius')
312 print(result) # Output: 100.0
313 result = convert_units(10, 'meters', 'grams')
314 print(result) # Output: Error: Invalid conversion
315
316
317
```

Output:

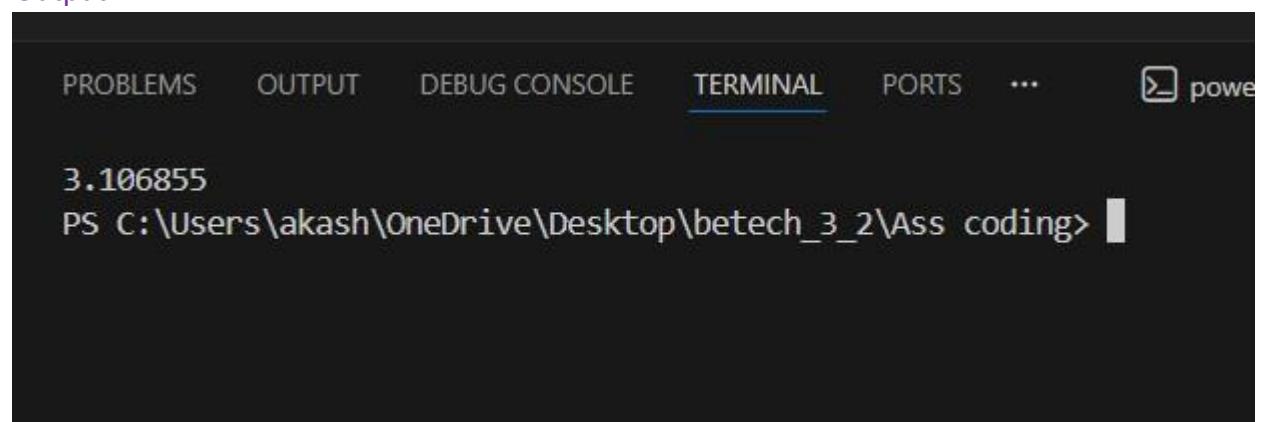
```
Error: Invalid conversion
PS C:\Users\akash\OneDrive\Desktop\betech_3_2\Ass coding> & C:\U
1.0
5000
212.0
100.0
Error: Invalid conversion
PS C:\Users\akash\OneDrive\Desktop\betech_3_2\Ass coding> █
```

Stage 2:

Code:

```
317 # write a python function to convert kilometers to miles
318 def kilometers_to_miles(km):
319     miles = km * 0.621371
320     return miles
321 # Example usage:
322 result = kilometers_to_miles(10)
323 print(result) # Output: 6.21371
324 result = kilometers_to_miles(5)
325 print(result) # Output: 3.106855
326
327
328
329
```

Output:



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS ... power
3.106855
PS C:\Users\akash\OneDrive\Desktop\betech_3_2\Ass coding> █
```

Stage 3:

Code:

```
326
327 # create a python function for unit conversion
328 # the function should convert kilometers to miles and miles to kilometers
329 def convert_distance(value, from_unit, to_unit):
330     if from_unit == 'kilometers' and to_unit == 'miles':
331         return value * 0.621371
332     elif from_unit == 'miles' and to_unit == 'kilometers':
333         return value / 0.621371
334     else:
335         return "Error: Invalid conversion"
336 # Example usage:
337 result = convert_distance(10, 'kilometers', 'miles')
338 print(result) # Output: 6.21371
339 result = convert_distance(6.21371, 'miles', 'kilometers')
340 print(result) # Output: 10.0
341 result = convert_distance(5, 'kilometers', 'grams')
342 print(result) # Output: Error: Invalid conversion
343
344
345
```

Output:

```
ms\Python\Python313\python.exe "c:/Users/akash/OneDrive/Desktop/betech_3_2\Ass coding\As_3.py"
6.21371
10.0
Error: Invalid conversion
PS C:\Users\akash\OneDrive\Desktop\betech_3_2\Ass coding>
```

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.