

AI Assisted Coding

Assignment – 3.2

Name:N.VARSHITH

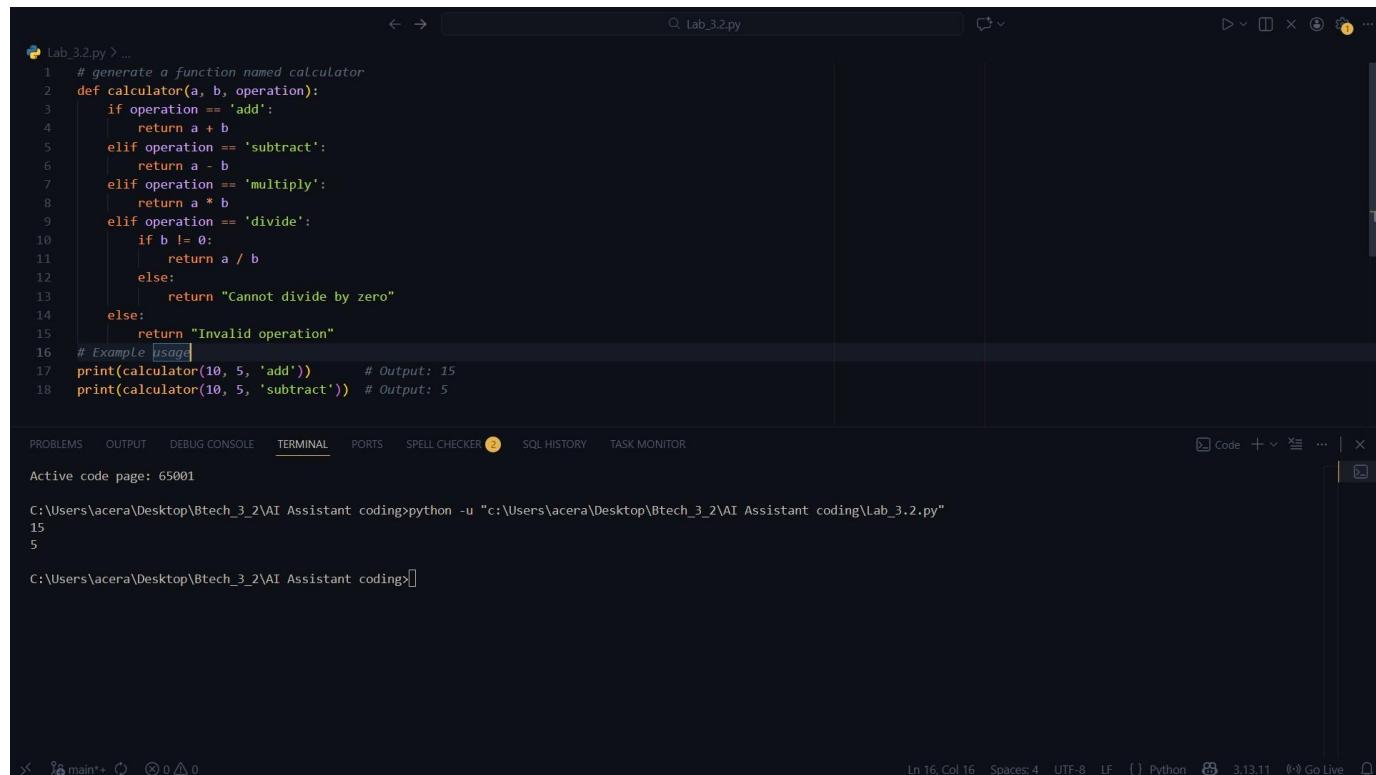
Batch: 21

HtNo: 2303A51441

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:



The screenshot shows a code editor window with the file 'Lab_3.2.py' open. The code defines a function 'calculator' that takes two arguments 'a' and 'b', and a third argument 'operation'. The function uses conditional statements to perform addition, subtraction, multiplication, or division based on the operation. It includes comments for division by zero and invalid operations. Two example usage print statements are provided at the bottom. The code editor interface includes tabs for PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL, PORTS, SPELL CHECKER, SQL HISTORY, and TASK MONITOR. The TERMINAL tab is active, showing the command 'python -u "c:\Users\acera\Desktop\Btech_3_2\AI Assistant coding\Lab_3.2.py"' and its output: '15' and '5'. The status bar at the bottom indicates the code has 16 lines, 16 columns, 4 spaces, and is in UTF-8 encoding, using Python 3.13.11, with Go Live and other icons.

```
1 # generate a function named calculator
2 def calculator(a, b, operation):
3     if operation == 'add':
4         return a + b
5     elif operation == 'subtract':
6         return a - b
7     elif operation == 'multiply':
8         return a * b
9     elif operation == 'divide':
10        if b != 0:
11            return a / b
12        else:
13            return "Cannot divide by zero"
14    else:
15        return "Invalid operation"
16 # Example usage
17 print(calculator(10, 5, 'add'))      # Output: 15
18 print(calculator(10, 5, 'subtract')) # Output: 5
```

Stage 2:

Code:

```

Lab_3.2.py > calculator
20  # write a function named calculator , that takes three parameters: two numbers and an operation (add, subtract, multiply, divide) and returns the result of the operation
21  on the two numbers.
22  def calculator(a, b, operation):
23      if operation == 'add':
24          return a + b
25      elif operation == 'subtract':
26          return a - b
27      elif operation == 'multiply':
28          return a * b
29      elif operation == 'divide':
30          if b != 0:
31              return a / b
32          else:
33              return "Cannot divide by zero"
34      else:
35          return "Invalid operation"
36  # Example usage
37  print(calculator(10, 5, 'add'))      # Output: 15
38  print(calculator(10, 5, 'subtract'))  # Output: 5
39  print(calculator(10, 5, 'multiply')) # Output: 50
40  print(calculator(10, 5, 'divide'))   # Output: 2.0
41  print(calculator(10, 0, 'divide'))   # Output: Cannot divide by zero

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS SPELL CHECKER SQL HISTORY TASK MONITOR

```

C:\Users\acer\Desktop\Btech_3_2\AI Assistant coding>python -u "c:\Users\acer\Desktop\Btech_3_2\AI Assistant coding\tempCodeRunnerFile.py"
15
5
50
2.0
Cannot divide by zero
C:\Users\acer\Desktop\Btech_3_2\AI Assistant coding>

```

Ln 23, Col 21 Spaces: 4 UTF-8 LF { } Python 3.13.11 Go Live

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:

```
Lab_3.2.py > ...
42
43 # write a python code to sort student marks
44 def sort_marks(marks):
45     return sorted(marks)
46 # Example usage
47 student_marks = [85, 92, 78, 90, 88]
48 sorted_marks = sort_marks(student_marks)
49 print("Sorted student marks:", sorted_marks)
50

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS SPELL CHECKER SQL HISTORY TASK MONITOR

5
5
50
2.0
Cannot divide by zero

C:\Users\acera\Desktop\Btech_3_2\AI Assistant coding>python -u "c:\Users\acera\Desktop\Btech_3_2\AI Assistant coding\tempCodeRunnerFile.py"
Sorted student marks: [78, 85, 88, 90, 92]

C:\Users\acera\Desktop\Btech_3_2\AI Assistant coding>

Ln 50, Col 1  Spaces: 4  LF  { } Python  3.13.11  Go Live
```

```
Lab_3.2.py > ...
51 # write a python code to sort student marks in ascending order the function to take List of inputs from the user and return the sorted List
52 def sort_marks(marks):
53     return sorted(marks)
54 # Taking input from the user
55 user_input = input("Enter student marks separated by commas: ")
56 try:
57     marks_list = [int(mark.strip()) for mark in user_input.split(',')]
58     sorted_marks = sort_marks(marks_list)
59     print("Sorted student marks:", sorted_marks)
60 except ValueError:
61     print("Invalid input. Please enter integers separated by commas.")
62

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS SPELL CHECKER SQL HISTORY TASK MONITOR

C:\Users\acera\Desktop\Btech_3_2\AI Assistant coding>python -u "c:\Users\acera\Desktop\Btech_3_2\AI Assistant coding\tempCodeRunnerFile.py"
Enter student marks separated by commas: 45,36,20,98,83
Sorted student marks: [20, 36, 45, 83, 98]

C:\Users\acera\Desktop\Btech_3_2\AI Assistant coding>

Ln 55, Col 19  Spaces: 4  LF  { } Python  3.13.11  Go Live
```

Stage 3:

Code:

The screenshot shows a code editor window with a dark theme. At the top, there's a tab labeled "Lab_3.2.py > ...". Below the tabs, the code is displayed:

```
63 # write a python code to sort student marks, implementing bubble sort algorithm to sort in descending order
64 def bubble_sort_descending(marks):
65     n = len(marks)
66     for i in range(n):
67         for j in range(0, n - i - 1):
68             if marks[j] < marks[j + 1]:
69                 marks[j], marks[j + 1] = marks[j + 1], marks[j]
70     return marks
71 # Example usage
72 student_marks = [85, 92, 78, 90, 88]
73 sorted_marks = bubble_sort_descending(student_marks)
74 print("Sorted student marks in descending order:", sorted_marks)
```

Below the code, the terminal output is shown:

```
C:\Users\acera\Desktop\Btech_3_2\AI Assistant coding>python -u "c:\Users\acera\Desktop\Btech_3_2\AI Assistant coding\tempCodeRunnerFile.py"
Sorted student marks in descending order: [92, 90, 88, 85, 78]

C:\Users\acera\Desktop\Btech_3_2\AI Assistant coding>
```

At the bottom, there are several status indicators: "main*+ 0 0 0", "Ln 73, Col 53", "Spaces:4", "UTF-8", "LF", "Python", "3.13.11", and "Go Live".

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:

The screenshot shows a code editor window with a dark theme. The file being edited is `Lab_3.2.py`. The code defines a function `is_prime` that checks if a number is prime. It includes an example usage where it prints whether the number 29 is prime or not. The terminal tab at the bottom shows the command `python -u "c:\Users\acera\Desktop\Btech_3_2\AI Assistant coding\tempCodeRunnerFile.py"` and the output "29 is a prime number.".

```
Lab_3.2.py > ...
15
76 # write a function to check if a number is prime or not
77 def is_prime(n):
78     if n <= 1:
79         return False
80     for i in range(2, int(n**0.5) + 1):
81         if n % i == 0:
82             return False
83     return True
84 # Example usage
85 number = 29
86 if is_prime(number):
87     print(f"{number} is a prime number.")
88 else:
89     print(f"{number} is not a prime number.")

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS SPELL CHECKER SQL HISTORY TASK MONITOR
C:\Users\acera\Desktop\Btech_3_2\AI Assistant coding>python -u "c:\Users\acera\Desktop\Btech_3_2\AI Assistant coding\tempCodeRunnerFile.py"
29 is a prime number.

C:\Users\acera\Desktop\Btech_3_2\AI Assistant coding>[

Ln 85, Col 12  Spaces: 4  UTF-8  LF  Python  3.13.11  Go Live  
```

Stage 2:

Code:

The screenshot shows a code editor window with a dark theme, identical to the previous one but with a different file name. The file is `main.py`. The code is the same as in the first screenshot, defining the `is_prime` function and demonstrating its use with the number 11 and 4.

```
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:

The screenshot shows a terminal window with a dark theme. The command `python -u "c:\Users\acera\Desktop\Btech_3_2\AI Assistant coding\tempCodeRunnerFile.py"` is run, and the output is "True" followed by "False".

```
C:\Users\acera\Desktop\Btech_3_2\AI Assistant coding>python -u "c:\Users\acera\Desktop\Btech_3_2\AI Assistant coding\tempCodeRunnerFile.py"
True
False
```

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:

```
C:\Users\acera\Desktop\Btech_3_2\AI Assistant coding>python -u "c:\Users\acera\Desktop\Btech_3_2\AI Assistant coding\tempCodeRunnerFile.py"
prime
not prime
prime
not prime
prime
not prime
not prime
not prime
not prime
```

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
209

```

Output:

```

C:\Users\acera\Desktop\Btech_3_2\AI Assistant coding>python -u "c:\Users\acera\Desktop\Btech_3_2\AI Assistant coding\tempCodeRunnerFile.py"
Welcome to the Student Grading System
Enter student name: Sharan
Enter student marks (0-100): 91
Student Name: Sharan
Marks: 91
Grade: A

C:\Users\acera\Desktop\Btech_3_2\AI Assistant 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()
```

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 btn_calculate = tk.Button(root, text="Submit", command=submit).grid(row=6, columnspan=2)
284 root.mainloop()
285

```

Output:

```

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

```

```

 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 # create a python tkinter user interface for a student grading system
 232
 233 # requirements:
 234 # ui should accept student name and marks of 5 subjects
 235 # each subject marks should be out of 100
 236 # on submit it should calculate total marks,percentage and grade
 237
 238 # grade rules:
 239 # 90-100 : A
 240 # 80-89 : B
 241 # 70-79 : C
 242 # 60-69 : D
 243 # below 60 : F
 244
 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'

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS SQL HISTORY TASK MONITOR

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

```

Output:

```

C:\Users\acera\Desktop\Btech_3_2\AI Assistant coding>python -u "c:\Users\acera\Desktop\Btech_3_2\AI Assistant coding\tempCodeRunnerFile.py"
1.0
5000
212.0
100.0
Error: Invalid conversion

```

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:

```
C:\Users\acera\Desktop\Btech_3_2\AI Assistant coding>python -u "c:\Users\acera\Desktop\Btech_3_2\AI Assistant coding\tempCodeRunnerFile.py"
3.106855
```

Stage 3:

Code:

```

320
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:

```
C:\Users\acera\Desktop\Btech_3_2\AI Assistant coding>python -u "c:\Users\acera\Desktop\Btech_3_2\AI Assistant coding\tempCodeRunnerFile.py"
6.21371
10.0
Invalid Conversion
```

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.