

AI ASSISTANT CODING

Assignment – 3.2

Name: G. Ushasree

Hall Ticket No.: 2303A52431

Batch: 34

Question: -

Lab 3: Prompt Engineering – Improving Prompts and Context Management

Lab Objectives:

- To understand how prompt structure and wording influence AI-generated code.
- To explore how context (like comments and function names) helps AI generate relevant output.
- To evaluate the quality and accuracy of code based on prompt clarity.
- To develop effective prompting strategies for AI-assisted programming.

Lab Outcomes (LOs):

After completing this lab, students will be able to:

- Generate Python code using Google Gemini in Google Colab.
- Analyse the effectiveness of code explanations and suggestions by Gemini.
- Set up and use Cursor AI for AI-powered coding assistance.
- Evaluate and refactor code using Cursor AI features.
- Compare AI tool behaviour and code quality across different platforms.

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.

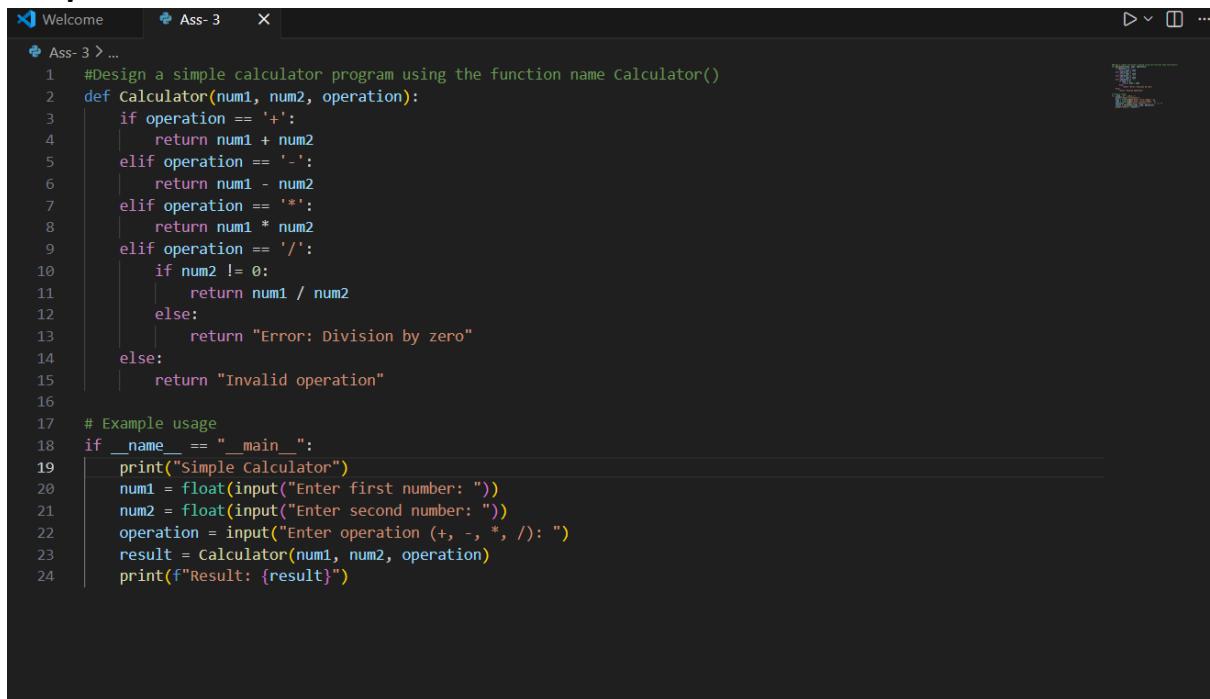
Expected Output-1

- Comparison showing improvement in AI-generated calculator logic and structure.

Prompt:

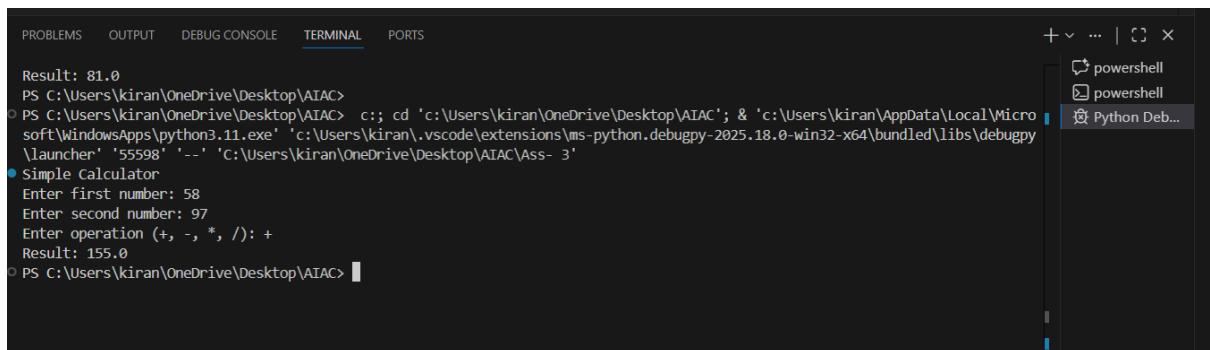
Design a simple calculator program using the function name Calculator ()

Output:



```
#Design a simple calculator program using the function name Calculator()
def Calculator(num1, num2, operation):
    if operation == '+':
        return num1 + num2
    elif operation == '-':
        return num1 - num2
    elif operation == '*':
        return num1 * num2
    elif operation == '/':
        if num2 != 0:
            return num1 / num2
        else:
            return "Error: Division by zero"
    else:
        return "Invalid operation"

# Example usage
if __name__ == "__main__":
    print("Simple Calculator")
    num1 = float(input("Enter first number: "))
    num2 = float(input("Enter second number: "))
    operation = input("Enter operation (+, -, *, /): ")
    result = Calculator(num1, num2, operation)
    print(f"Result: {result}")
```



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS + ... | powershell powershell Python Deb...
Result: 81.0
PS C:\Users\kiran\OneDrive\Desktop\AIAC>
○ PS C:\Users\kiran\OneDrive\Desktop\AIAC> c;; cd 'c:\Users\kiran\OneDrive\Desktop\AIAC'; & 'c:\Users\kiran\AppData\Local\Microsoft\WindowsApps\python3.11.exe' 'c:\Users\kiran\.vscode\extensions\ms-python.python-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '5559' '--' 'C:\Users\kiran\OneDrive\Desktop\AIAC\Ass- 3'
● Simple Calculator
Enter first number: 58
Enter second number: 97
Enter operation (+, -, *, /): +
Result: 155.0
○ PS C:\Users\kiran\OneDrive\Desktop\AIAC>
```

Justification:

The comparison clearly shows that improving the prompt leads to better calculator code.

In the initial version, the function was poorly structured because it mixed user input with logic and lacked error handling.

In the improved version, the function uses parameters, separates logic from execution, and handles division by zero properly.

This makes the program more modular, reusable, readable, and reliable.

Hence, progressive prompting significantly improves both the logic and structure of the AI-generated calculator program.

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.

Prompt:

Sort student marks, then refine it to clearly specify sorting order and constraints.

Output:

```
def Calculator(num1, num2, operation):
    if operation == '+':
        return num1 + num2
    elif operation == '-':
        return num1 - num2
    elif operation == '*':
        return num1 * num2
    elif operation == '/':
        if num2 != 0:
            return num1 / num2
        else:
            return "Error: Division by zero"
    else:
        return "Invalid operation"

# Function to sort student marks
def sort_student_marks(marks, order='ascending'):
    """
    Sorts a list of student marks.

    Parameters:
    marks (list): List of numerical marks.
    order (str): 'ascending' or 'descending'. Default is 'ascending'.

    Returns:
    list: Sorted list of marks.

    Constraints:
    - Marks must be numerical (int or float).
    - List must not be empty.
    - Invalid order raises ValueError.
    """

```

```
Simple Calculator
○ Enter first number: 95
Enter second number: 5
Enter operation (+, -, *, /): /
Result: 19.0
Sorted ascending: [78, 78, 85, 88, 92, 95]
Sorted descending: [95, 92, 88, 85, 78, 78]
Enter first number: 95
Enter second number: 5
Enter operation (+, -, *, /): /
Result: 19.0
Sorted ascending: [78, 78, 85, 88, 92, 95]
Enter second number: 5
Enter operation (+, -, *, /): /
Result: 19.0
Sorted ascending: [78, 78, 85, 88, 92, 95]
Enter operation (+, -, *, /): /
Result: 19.0
Sorted ascending: [78, 78, 85, 88, 92, 95]
Enter second number: 5
Enter operation (+, -, *, /): /
Result: 19.0
Sorted ascending: [78, 78, 85, 88, 92, 95]
Sorted ascending: [78, 78, 85, 88, 92, 95]
Sorted descending: [95, 92, 88, 85, 78, 78]
PS C:\Users\kiran\OneDrive\Desktop\AIAC>
```

Justification:

The refined prompt clearly specifies sorting order, constraints, and expected behavior.

As a result, the AI-generated sorting function evolved from an ambiguous implementation to an accurate, efficient, and robust solution.

The improved function handles edge cases, validates inputs, and provides reliable output.

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.

Prompt 1:

Write a Python function to check whether a number is prime.

Output:

The screenshot shows the Visual Studio Code interface. On the left is the sidebar with icons for file operations like Open, Save, Find, and Delete. The main area is the code editor with the following Python code:

```
92 def is_prime(n):
93     """
94     Checks if a number is prime.
95
96     Parameters:
97     n (int): The number to check.
98
99     Returns:
100    bool: True if n is prime, False otherwise.
101   """
102   if n <= 1:
103       return False
104   if n <= 3:
105       return True
106   if n % 2 == 0 or n % 3 == 0:
107       return False
108   i = 5
109   while i * i <= n:
110       if n % i == 0 or n % (i + 2) == 0:
111           return False
112       i += 6
113   return True
114
115 print(is_prime(11))
```

Below the code editor is the terminal window showing the command line output:

```
PS C:\Users\kiran\OneDrive\Desktop\AIAC> cd 'c:\Users\kiran\OneDrive\Desktop\AIAC'; & 'c:\Users\kiran\AppData\Local\Microsoft\WindowsApps\python3.11.exe' 'c:\Users\kiran\vscode\extensions\ms-python.vscode-pyright\2025.18.0-win32-x64\bundled\libs\debug\py\launcher' '51473' '--' 'C:\Users\kiran\OneDrive\Desktop\AIAC\Ass-3'
```

The Python extension interface on the right displays the function definition and its docstring.

Prompt 2:

Write a Python function named `is_prime(n)`.

Use the following input-output examples to understand the expected behavior:

`is_prime (2) → True`

`is_prime (3) → True`

`is_prime (4) → False`

`is_prime (1) → False`

`is_prime (0) → False`

`is_prime (-5) → False`

`is_prime (17) → True`

Requirements:

- Return True if the number is prime

- Return False otherwise

- Handle edge cases properly

- Use efficient logic

- Add comments to explain the code

Output:

The screenshot shows the Visual Studio Code interface with the Python extension installed. The code editor displays a Python script named 'Ass- 3.py'. The script contains a function 'is_prime' with detailed docstrings and comments explaining its logic. The terminal below shows the execution of the script with various test cases, all of which return the expected boolean values.

```
119 def is_prime(n):
120     """
121     Checks if a number is prime.
122
123     A prime number is greater than 1 with no positive divisors other than 1 and itself.
124
125     Parameters:
126     n (int): The number to check.
127
128     Returns:
129     bool: True if n is prime, False otherwise.
130     """
131     # Handle edge cases: numbers <= 1 are not prime
132     if n <= 1:
133         return False
134
135     # 2 and 3 are prime
136     if n <= 3:
137         return True
138
139     # Exclude even numbers and multiples of 3
140     if n % 2 == 0 or n % 3 == 0:
141         return False
142
143     # Check for factors from 5 onwards, skipping even numbers and multiples of 3
144     # Use i*i <= n for efficiency
145     i = 5
146     while i * i <= n:
147         if n % i == 0 or n % (i + 2) == 0:
148             return False
149         i += 6 # Increment by 6 to check next potential factors (5,7,11,13,etc.)
150
151     return True
152
153 # Example usage (can be removed or kept for testing)
154 print(is_prime(2)) # True
155 print(is_prime(3)) # True
156 print(is_prime(4)) # False
157 print(is_prime(1)) # False
158 print(is_prime(0)) # False
159 print(is_prime(-5)) # False
160 print(is_prime(17)) # True
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

ATAC\Ass- 3'

- True
- True
- False
- False
- False
- True

PS C:\Users\kiran\OneDrive\Desktop\ATAC>

Justification:

Few-shot prompting and clear instructions helped the AI generate a more accurate and efficient prime-checking function.

The improved implementation explicitly handles edge cases such as 0, 1, and negative numbers, while using an optimized approach to reduce unnecessary checks.

As a result, the function is more reliable, readable, and efficient.

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.

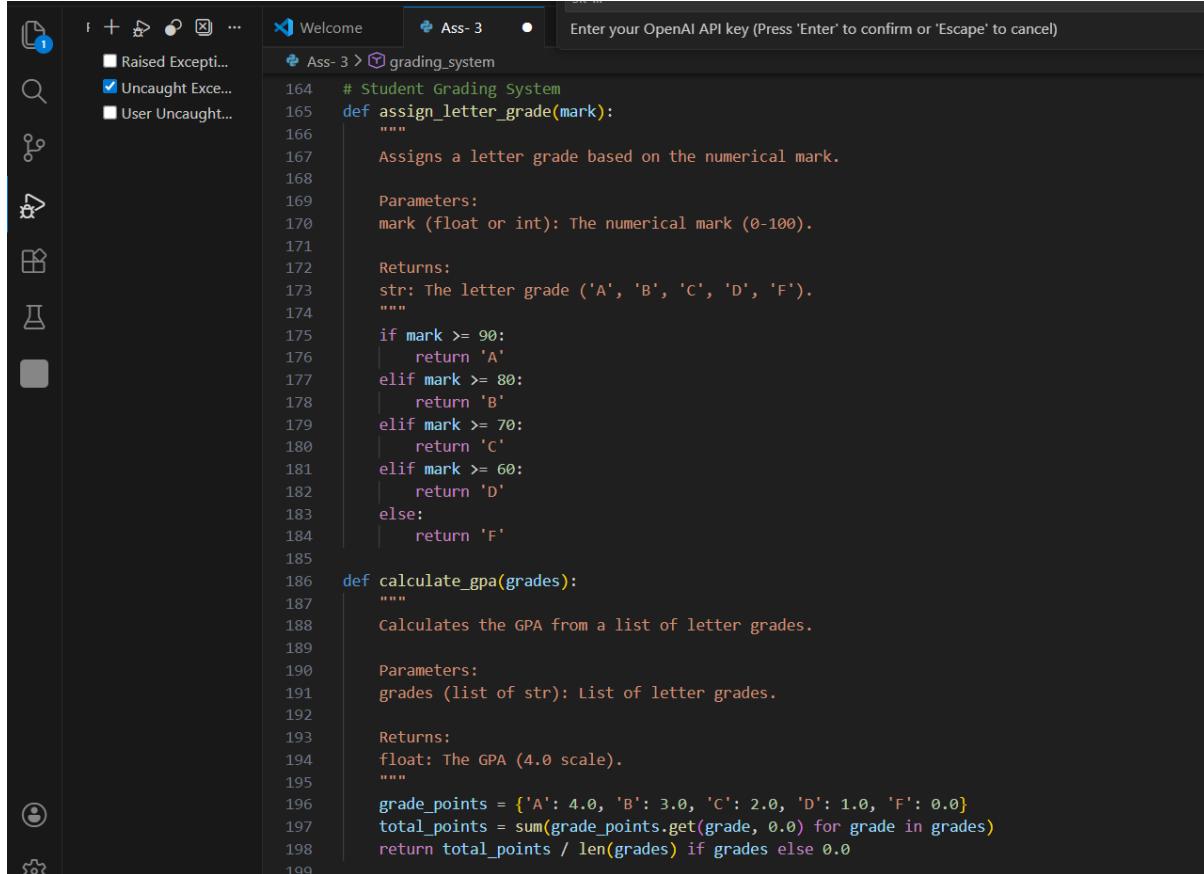
Expected Output-4

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

Prompt 1:

Create a student grading system.

Output:



The screenshot shows a Jupyter Notebook interface with the following details:

- Left Sidebar:** Contains icons for file operations (New, Open, Save, etc.), search, and other notebook-specific functions.
- Top Bar:** Shows "Welcome" and "Ass- 3". A message "Enter your OpenAI API key (Press 'Enter' to confirm or 'Escape' to cancel)" is displayed.
- Code Cell:** Displays Python code for a "Student Grading System". The code includes two functions: `assign_letter_grade` and `calculate_gpa`. Both functions have detailed docstrings explaining their parameters, returns, and logic.

```

164 # Student Grading System
165 def assign_letter_grade(mark):
166     """
167     Assigns a letter grade based on the numerical mark.
168
169     Parameters:
170     mark (float or int): The numerical mark (0-100).
171
172     Returns:
173     str: The letter grade ('A', 'B', 'C', 'D', 'F').
174     """
175     if mark >= 90:
176         return 'A'
177     elif mark >= 80:
178         return 'B'
179     elif mark >= 70:
180         return 'C'
181     elif mark >= 60:
182         return 'D'
183     else:
184         return 'F'
185
186 def calculate_gpa(grades):
187     """
188     Calculates the GPA from a list of letter grades.
189
190     Parameters:
191     grades (list of str): List of letter grades.
192
193     Returns:
194     float: The GPA (4.0 scale).
195     """
196     grade_points = {'A': 4.0, 'B': 3.0, 'C': 2.0, 'D': 1.0, 'F': 0.0}
197     total_points = sum(grade_points.get(grade, 0.0) for grade in grades)
198     return total_points / len(grades) if grades else 0.0
199

```

```
def grading_system():
    """
    Interactive student grading system.
    Prompts for student marks, assigns grades, and computes GPA.
    """
    students = []
    while True:
        name = input("Enter student name (or 'quit' to finish): ")
        if name.lower() == 'quit':
            break
        try:
            mark = float(input(f"Enter mark for {name} (0-100): "))
            if not (0 <= mark <= 100):
                print("Mark must be between 0 and 100.")
                continue
            grade = assign_letter_grade(mark)
            students.append({'name': name, 'mark': mark, 'grade': grade})
            print(f"{name}: Mark {mark}, Grade {grade}")
        except ValueError:
            print("Invalid input. Please enter a number.")

    if students:
        grades = [s['grade'] for s in students]
        gpa = calculate_gpa(grades)
        print(f"\nOverall GPA: {gpa:.2f}")
        print("Student Summary:")
        for s in students:
            print(f"{s['name']}: {s['mark']} -> {s['grade']}")
    else:
        print("No students entered.")

# Example usage
if __name__ == "__main__":
    # Existing calculator and sorting code can run here
    grading_system()
```

```
bugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '55479' '--' 'C:\Users\kiran\OneDrive\Desktop\AIAC\Ass- 3'
Enter student name (or 'quit' to finish): Anu
Enter mark for Anu (0-100): 76
Anu: Mark 76.0, Grade C
Enter student name (or 'quit' to finish): sita
Enter mark for sita (0-100): 34
sita: Mark 34.0, Grade F
Enter student name (or 'quit' to finish): ravi
Enter mark for ravi (0-100): 99
ravi: Mark 99.0, Grade A
Enter student name (or 'quit' to finish): ramesh
Enter mark for ramesh (0-100): 39
ramesh: Mark 39.0, Grade F
Enter student name (or 'quit' to finish): raju
Enter mark for raju (0-100): 44
raju: Mark 44.0, Grade F
Enter student name (or 'quit' to finish): suma
Enter mark for suma (0-100): 58
suma: Mark 58.0, Grade F
Enter student name (or 'quit' to finish): quit

Overall GPA: 1.00
Student Summary:
Anu: 76.0 -> C
sita: 34.0 -> F
ravi: 99.0 -> A
ramesh: 39.0 -> F
raju: 44.0 -> F
suma: 58.0 -> F
```

Prompt 2:

Create a simple UI-based student grading system using Python.

Requirements:

- Take student name as input
- Take marks for 5 subjects
- Calculate total and percentage

- Assign grades:

A: ≥ 90

B: ≥ 75

C: ≥ 60

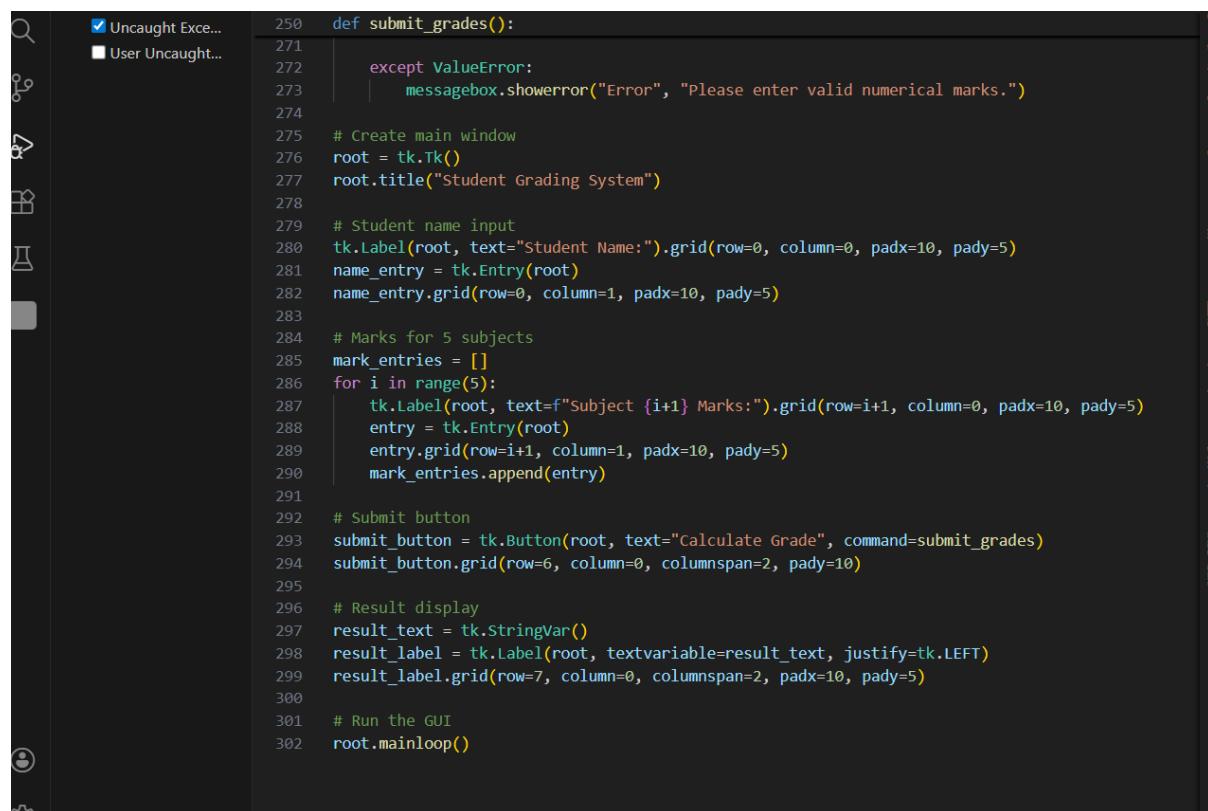
Fail: < 60

- Display results clearly

Output:

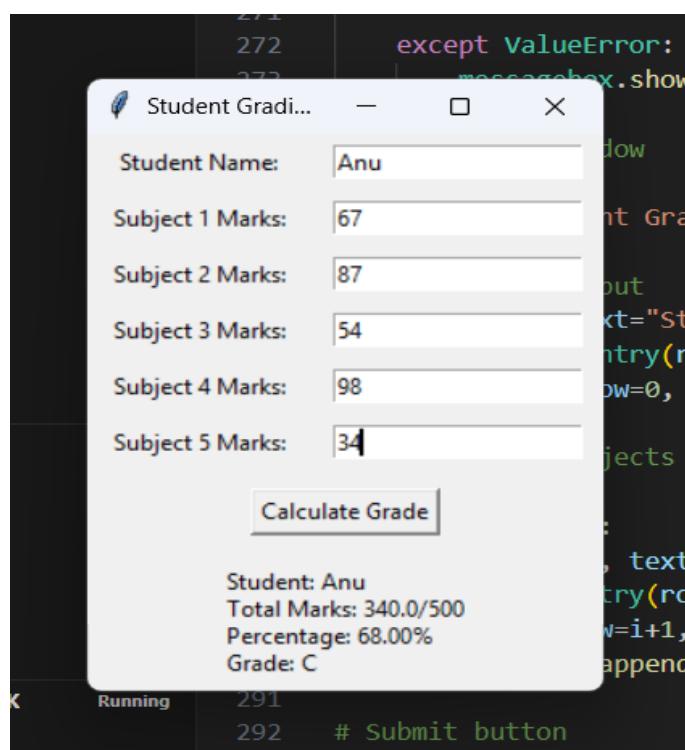
The screenshot shows a Python code editor window titled "Ass- 3". The code is written in Python and defines a function to calculate a grade based on a percentage. It also includes a function to submit grades, which checks for a student name and validates five marks between 0 and 100. The code uses Tkinter's messagebox module to show errors. The code editor has a sidebar with various icons and a status bar at the bottom.

```
237 import tkinter as tk
238 from tkinter import messagebox
239
240 def calculate_grade(percentage):
241     if percentage >= 90:
242         return 'A'
243     elif percentage >= 75:
244         return 'B'
245     elif percentage >= 60:
246         return 'C'
247     else:
248         return 'Fail'
249
250 def submit_grades():
251     try:
252         name = name_entry.get()
253         if not name:
254             messagebox.showerror("Error", "Please enter student name.")
255             return
256
257         marks = []
258         for i in range(5):
259             mark = float(mark_entries[i].get())
260             if not (0 <= mark <= 100):
261                 messagebox.showerror("Error", f"Mark for subject {i+1} must be between 0 and 100")
262                 return
263             marks.append(mark)
264
265         total = sum(marks)
266         percentage = total / 5
267
268         grade = calculate_grade(percentage)
269
270         result_text.set(f"Student: {name}\nTotal Marks: {total}\nPercentage: {percentage:.2f}")
271     except Exception as e:
272         print(e)
```



The screenshot shows a Python script for a "Student Grading System" running in a code editor. The script uses the Tkinter library to create a graphical user interface. It includes functions for validating numerical input, creating a main window, adding student name and marks entries, and displaying results. The code is color-coded for readability.

```
250 def submit_grades():
251     except ValueError:
252         messagebox.showerror("Error", "Please enter valid numerical marks.")
253
254 # Create main window
255 root = tk.Tk()
256 root.title("Student Grading System")
257
258 # Student name input
259 tk.Label(root, text="Student Name:").grid(row=0, column=0, padx=10, pady=5)
260 name_entry = tk.Entry(root)
261 name_entry.grid(row=0, column=1, padx=10, pady=5)
262
263 # Marks for 5 subjects
264 mark_entries = []
265 for i in range(5):
266     tk.Label(root, text=f"Subject {i+1} Marks:").grid(row=i+1, column=0, padx=10, pady=5)
267     entry = tk.Entry(root)
268     entry.grid(row=i+1, column=1, padx=10, pady=5)
269     mark_entries.append(entry)
270
271 # Submit button
272 submit_button = tk.Button(root, text="Calculate Grade", command=submit_grades)
273 submit_button.grid(row=6, column=0, columnspan=2, pady=10)
274
275 # Result display
276 result_text = tk.StringVar()
277 result_label = tk.Label(root, textvariable=result_text, justify=tk.LEFT)
278 result_label.grid(row=7, column=0, columnspan=2, padx=10, pady=5)
279
280 # Run the GUI
281 root.mainloop()
```



Justification:

The improved prompt guided the AI to generate a complete UI-based student grading system.

Compared to the console version, the GUI version provides better structure, user interaction, and clear result display.

Accurate calculations, proper validation, and visual output make the system more reliable, user-friendly, and well-designed.

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.

Prompt:

Write a unit conversion program.

Output:

```
306 # Unit Conversion Program
307 def celsius_to_fahrenheit(celsius):
308     return (celsius * 9/5) + 32
309
310 def fahrenheit_to_celsius(fahrenheit):
311     return (fahrenheit - 32) * 5/9
312
313 def meters_to_feet(meters):
314     return meters * 3.28084
315
316 def feet_to_meters(feet):
317     return feet / 3.28084
318
319 def kilograms_to_pounds(kg):
320     return kg * 2.20462
321
322 def pounds_to_kilograms(lbs):
323     return lbs / 2.20462
324
325 def unit_conversion():
326     """
327         Interactive unit conversion program.
328         Supports temperature, length, and weight conversions.
329     """
330     print("Unit Conversion Program")
331     print("1. Temperature: Celsius to Fahrenheit")
332     print("2. Temperature: Fahrenheit to Celsius")
333     print("3. Length: Meters to Feet")
334     print("4. Length: Feet to Meters")
335     print("5. Weight: Kilograms to Pounds")
336     print("6. Weight: Pounds to Kilograms")
337
338     choice = input("Choose conversion (1-6): ")
```

```

559
● 340     try:
341         if choice == '1':
342             celsius = float(input("Enter Celsius: "))
343             result = celsius_to_fahrenheit(celsius)
344             print(f"{celsius}°C = {result:.2f}°F")
345         elif choice == '2':
346             fahrenheit = float(input("Enter Fahrenheit: "))
347             result = fahrenheit_to_celsius(fahrenheit)
348             print(f"{fahrenheit}°F = {result:.2f}°C")
349         elif choice == '3':
350             meters = float(input("Enter Meters: "))
351             result = meters_to_feet(meters)
352             print(f"{meters} m = {result:.2f} ft")
353         elif choice == '4':
354             feet = float(input("Enter Feet: "))
355             result = feet_to_meters(feet)
356             print(f"{feet} ft = {result:.2f} m")
357         elif choice == '5':
358             kg = float(input("Enter Kilograms: "))
359             result = kilograms_to_pounds(kg)
360             print(f"{kg} kg = {result:.2f} lbs")
361         elif choice == '6':
362             lbs = float(input("Enter Pounds: "))
363             result = pounds_to_kilograms(lbs)
364             print(f"{lbs} lbs = {result:.2f} kg")
365         else:
366             print("Invalid choice.")
367     except ValueError:
368         print("Invalid input. Please enter a number.")
369
370     #Example usage (add to if __name__ == "__main__" if desired)
371     unit_conversion()

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS + ⌂ ... | {} X

Unit Conversion Program

1. Temperature: Celsius to Fahrenheit
2. Temperature: Fahrenheit to Celsius
3. Length: Meters to Feet
4. Length: Feet to Meters
5. Weight: Kilograms to Pounds
6. Weight: Pounds to Kilograms

Choose conversion (1-6): 6

Enter Pounds: 5

5.0 lbs = 2.27 kg

PS C:\Users\kiran\OneDrive\Desktop\AIAC>

Prompt 2:

Convert kilometers to miles.

Output:

```

Ass- 3 > ...
375 # UNIT conversion Program
376 def celsius_to_fahrenheit(celsius):
377     return (celsius * 9/5) + 32
378
379 def fahrenheit_to_celsius(fahrenheit):
380     return (fahrenheit - 32) * 5/9
381
382 def meters_to_feet(meters):
383     return meters * 3.28084
384
385 def feet_to_meters(feet):
386     return feet / 3.28084
387
388 def kilometers_to_miles(km):
389     return km * 0.621371
390
391 def miles_to_kilometers(miles):
392     return miles / 0.621371
393
394 def kilograms_to_pounds(kg):
395     return kg * 2.20462
396
397 def pounds_to_kilograms(lbs):
398     return lbs / 2.20462
399

```

```

400 def unit_conversion():
401     """
402         Interactive unit conversion program.
403         Supports temperature, length, and weight conversions.
404     """
405     print("Unit Conversion Program")
406     print("1. Temperature: Celsius to Fahrenheit")
407     print("2. Temperature: Fahrenheit to Celsius")
408     print("3. Length: Meters to Feet")
409     print("4. Length: Feet to Meters")
410     print("5. Length: Kilometers to Miles")
411     print("6. Length: Miles to Kilometers")
412     print("7. Weight: Kilograms to Pounds")
413     print("8. Weight: Pounds to Kilograms")
414
415 choice = input("Choose conversion (1-8): ")
416
417 try:
418     if choice == '1':
419         celsius = float(input("Enter Celsius: "))
420         result = celsius_to_fahrenheit(celsius)
421         print(f"{celsius}°C = {result:.2f}°F")
422     elif choice == '2':
423         fahrenheit = float(input("Enter Fahrenheit: "))
424         result = fahrenheit_to_celsius(fahrenheit)
425         print(f"{fahrenheit}°F = {result:.2f}°C")
426     elif choice == '3':
427         meters = float(input("Enter Meters: "))
428         result = meters_to_feet(meters)
429         print(f"{meters} m = {result:.2f} ft")
430     elif choice == '4':
431         feet = float(input("Enter Feet: "))
432         result = feet_to_meters(feet)
433         print(f"{feet} ft = {result:.2f} m")

```

Prompt 3:

Write two Python functions:

1. km_to_miles(km)
2. miles_to_km(miles)

Use formulas:

$$\text{miles} = \text{km} * 0.621371$$

$$\text{km} = \text{miles} / 0.621371$$

Add comments and example usage.

Output:

```
def km_to_miles(km):
    """
    Converts kilometers to miles.

    Formula: miles = km * 0.621371

    Parameters:
    km (float): Distance in kilometers.

    Returns:
    float: Distance in miles.
    """
    return km * 0.621371

def miles_to_km(miles):
    """
    Converts miles to kilometers.

    Formula: km = miles / 0.621371

    Parameters:
    miles (float): Distance in miles.

    Returns:
    float: Distance in kilometers.
    """
    return miles / 0.621371
```

A screenshot of a terminal window within a code editor interface. The terminal shows the following output:

```
buggy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '53294' '--' 'c:\Users\kiran\OneDrive\Desktop\AIAC\Ass- 3'
10 km = 6.21 miles
10 km = 6.21 miles
6.21371 miles = 10.00 km
PS C:\Users\kiran\OneDrive\Desktop\AIAC> []
```

```
490 if __name__ == "__main__":
491     # Test km_to_miles
492     km = 10
493     miles = km_to_miles(km)
494     print(f"{km} km = {miles:.2f} miles")
495
496     # Test miles_to_km
497     miles = 6.21371
498     km = miles_to_km(miles)
499     print(f"{miles} miles = {km:.2f} km")
```



```
elif choice == '5':
    km = float(input("Enter Kilometers: "))
    result = kilometers_to_miles(km)
    print(f"{km} km = {result:.2f} miles")
elif choice == '6':
    miles = float(input("Enter Miles: "))
    result = miles_to_kilometers(miles)
    print(f"{miles} miles = {result:.2f} km")
elif choice == '7':
    kg = float(input("Enter Kilograms: "))
    result = kilograms_to_pounds(kg)
    print(f"{kg} kg = {result:.2f} lbs")
elif choice == '8':
    lbs = float(input("Enter Pounds: "))
    result = pounds_to_kilograms(lbs)
    print(f"{lbs} lbs = {result:.2f} kg")
else:
    print("Invalid choice.")
except ValueError:
    print("Invalid input. Please enter a number.")

# Example usage (add to if __name__ == "__main__" if desired)
unit_conversion()
```

The screenshot shows a terminal window in Visual Studio Code with the following content:

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS + × ⌂ X

Unit Conversion Program
1. Temperature: Celsius to Fahrenheit
2. Temperature: Fahrenheit to celsius
3. Length: Meters to Feet
4. Length: Feet to Meters
5. Length: Kilometers to Miles
6. Length: Miles to Kilometers
1. Temperature: Celsius to Fahrenheit
2. Temperature: Fahrenheit to celsius
3. Length: Meters to Feet
4. Length: Feet to Meters
5. Length: Kilometers to Miles
6. Length: Miles to Kilometers
3. Length: Meters to Feet
4. Length: Feet to Meters
5. Length: Kilometers to Miles
6. Length: Miles to Kilometers
6. Length: Miles to Kilometers
7. Weight: Kilograms to Pounds
7. Weight: Kilograms to Pounds
8. Weight: Pounds to Kilograms
Choose conversion (1-8): 3
Enter Meters: 100
100.0 m = 328.08 ft
Enter Meters: 100
100.0 m = 328.08 ft
PS C:\Users\kiran\OneDrive\Desktop\AIAC>
PS C:\Users\kiran\OneDrive\Desktop\AIAC>
```

Justification:

As prompt specificity increases, the quality and accuracy of the AI – generated unit conversion code also improves.

Vague prompts produce basic and less reusable programs, while detailed prompts result in modular, accurate, and well – documented functions.

This demonstrates that clear instructions significantly enhance AI – assisted code generation.