

AI ASSISTANT CODING

Assignment – 3.2

Name: K.ShivaCharan

Hall Ticket No.: 2303A52160

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.

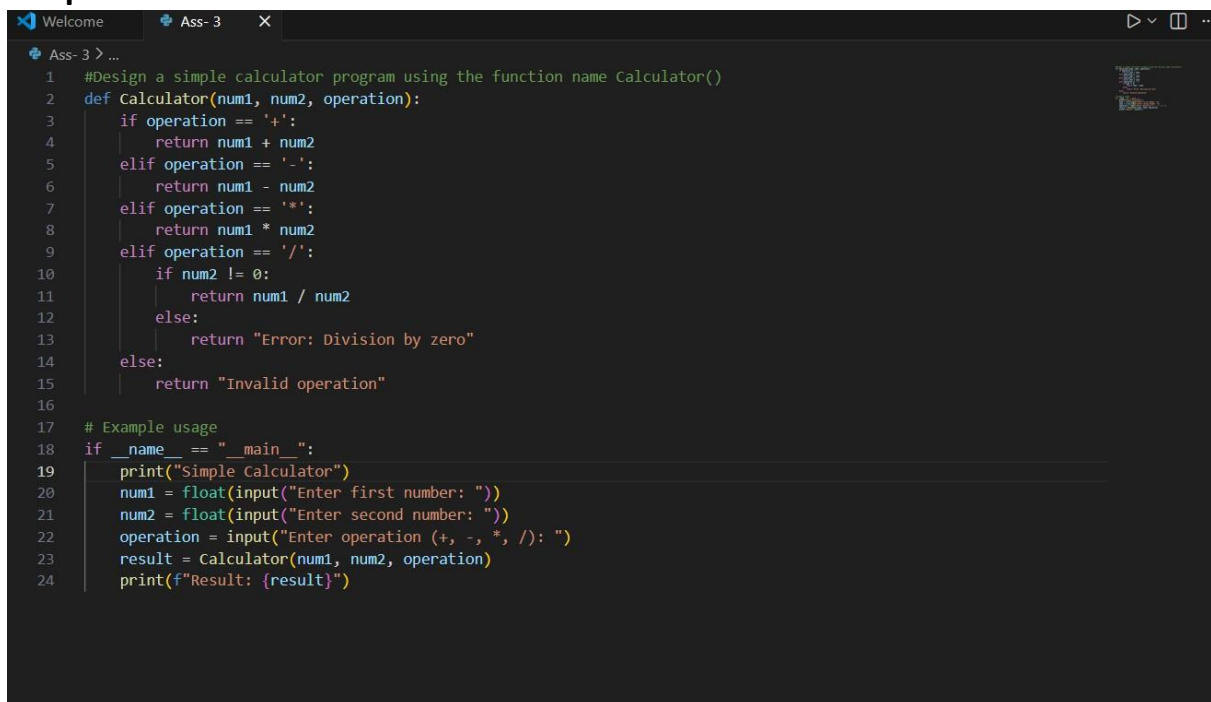
Expected Output-1

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

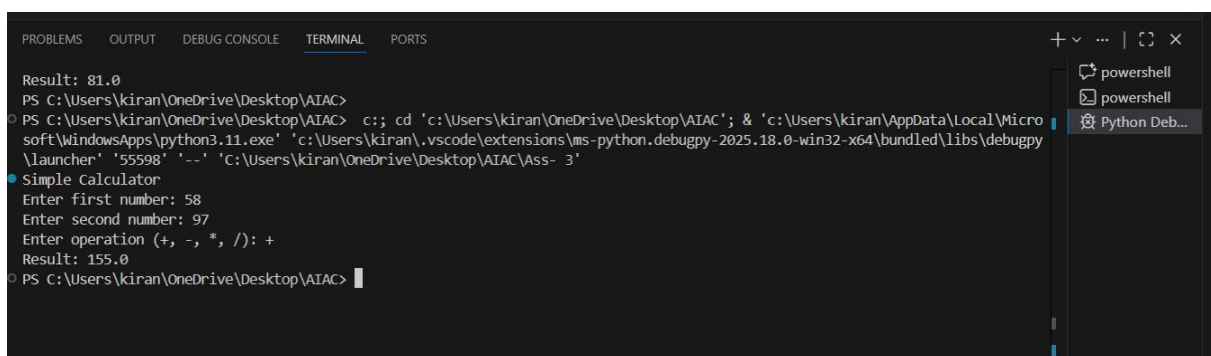
Prompt:

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

Output:



```
1 #Design a simple calculator program using the function name Calculator()
2 def Calculator(num1, num2, operation):
3     if operation == '+':
4         return num1 + num2
5     elif operation == '-':
6         return num1 - num2
7     elif operation == '*':
8         return num1 * num2
9     elif operation == '/':
10        if num2 != 0:
11            return num1 / num2
12        else:
13            return "Error: Division by zero"
14    else:
15        return "Invalid operation"
16
17 # Example usage
18 if __name__ == "__main__":
19     print("Simple Calculator")
20     num1 = float(input("Enter first number: "))
21     num2 = float(input("Enter second number: "))
22     operation = input("Enter operation (+, -, *, /): ")
23     result = Calculator(num1, num2, operation)
24     print(f"Result: {result}")
```



```
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.debugpy-2025.18.0-win32-x64\bundle\libs\debugpy\launcher' '55598' '--' '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.
    """
```

The screenshot shows a code editor with the following Python code:

```
46 def sort_student_marks(marks, order='ascending'):
47     """
48     Sorts a list of student marks.
49
50     Parameters:
51     marks (list): List of numerical marks.
52     order (str): 'ascending' or 'descending'. Default is 'ascending'.
53
54     Returns:
55     list: Sorted list of marks.
56
57     Constraints:
58     - Marks must be numerical (int or float).
59     - List must not be empty.
60     - Invalid order raises ValueError.
61     """
62     if not marks:
63         raise ValueError("Marks list cannot be empty.")
64     if not all(isinstance(mark, (int, float)) for mark in marks):
65         raise ValueError("All marks must be numerical.")
66     if order not in ['ascending', 'descending']:
67         raise ValueError("Order must be 'ascending' or 'descending'.")
68
69     sorted_marks = sorted(marks)
70     if order == 'descending':
71         sorted_marks.reverse()
72     return sorted_marks
73
74 # Example usage
75 if __name__ == "__main__":
76     print("Simple Calculator")
77     num1 = float(input("Enter first number: "))
78     num2 = float(input("Enter second number: "))
79     operation = input("Enter operation (+, -, *, /): ")
80     result = Calculator(num1, num2, operation)
81     print(f"Result: {result}")
82
83 # Sample marks for sorting
84 marks = [85, 92, 78, 88, 95, 78] # Includes duplicates
85 try:
86     sorted_asc = sort_student_marks(marks, 'ascending')
87     print(f"Sorted ascending: {sorted_asc}")
88     sorted_desc = sort_student_marks(marks, 'descending')
89     print(f"Sorted descending: {sorted_desc}")
90 except ValueError as e:
91     print(f"Error: {e}")
```

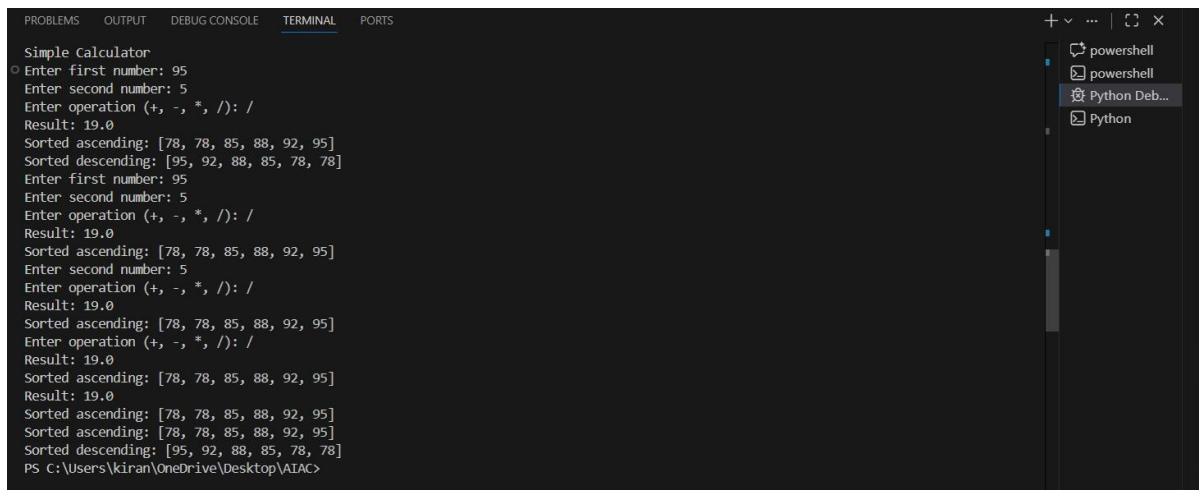
The chat window on the right contains the following refined prompt:

```
<- SIMPLE CALCULATOR PROGRAM DESIGN WITH...
raise ValueError("All marks m
if order not in ['ascending', 'de
raise ValueError("Order must l

sorted_marks = sorted(marks)
if order == 'descending':
    sorted_marks.reverse()
return sorted_marks

# Example usage
if __name__ == "__main__":
    print("Simple Calculator")
    num1 = float(input("Enter first n
    num2 = float(input("Enter second
    operation = input("Enter operatio
    result = Calculator(num1, num2, o
    print(f"Result: {result}")

# Sample marks for sorting
marks = [85, 92, 78, 88, 95, 78]
try:
    sorted_asc = sort_student_mari
    print(f"Sorted ascending: {sor
    sorted_desc = sort_student_mai
    print(f"Sorted descending: {sc
except ValueError as e:
    print(f"Error: {e}")
```



```
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]
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.

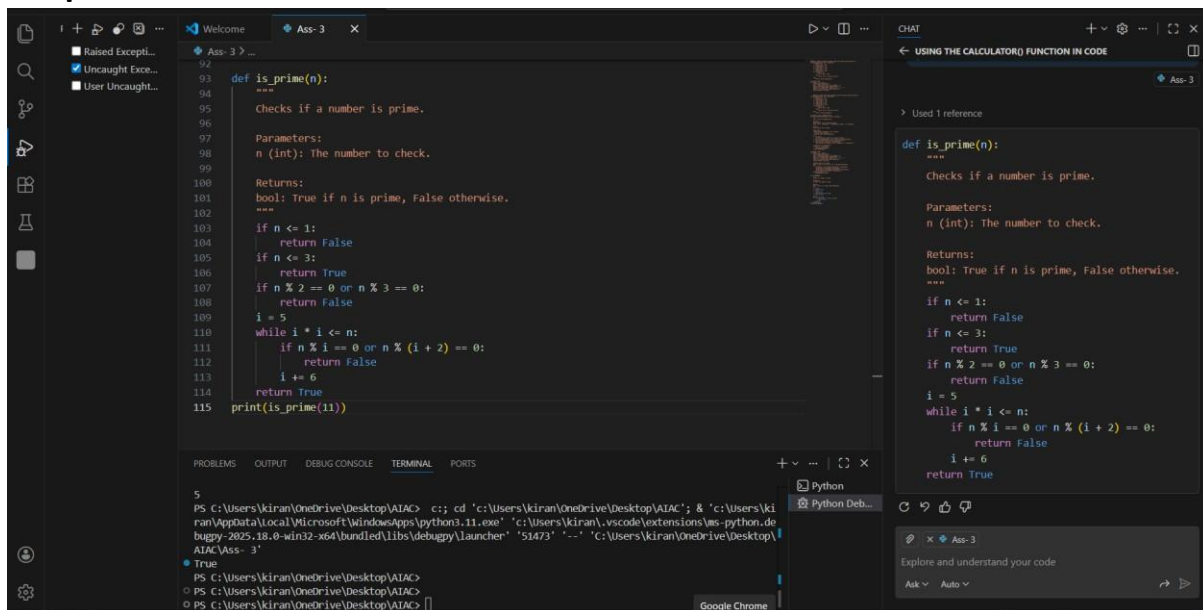
Expected Output-3

- 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 a VS Code editor with a Python file named 'Ass-3'. The code defines a function `is_prime(n)` that checks if a number is prime. The function uses a loop to check divisibility from 2 to \sqrt{n} . The terminal output shows the function being called with `is_prime(11)`, which returns `True`.

```
def is_prime(n):
    """
    Checks if a number is prime.

    Parameters:
    n (int): The number to check.

    Returns:
    bool: True if n is prime, False otherwise.
    """
    if n <= 1:
        return False
    if n <= 3:
        return True
    if n % 2 == 0 or n % 3 == 0:
        return False
    i = 5
    while i * i <= n:
        if n % i == 0 or n % (i + 2) == 0:
            return False
        i += 6
    return True
print(is_prime(11))
```

Terminal Output:

```
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.debugpy-2025.18.0-win32-x64\bundle\libs\debugpy\launcher' '51473' '-.' 'c:\Users\Kiran\OneDrive\Desktop\AIAC\Ass-3'
```

Output: True

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

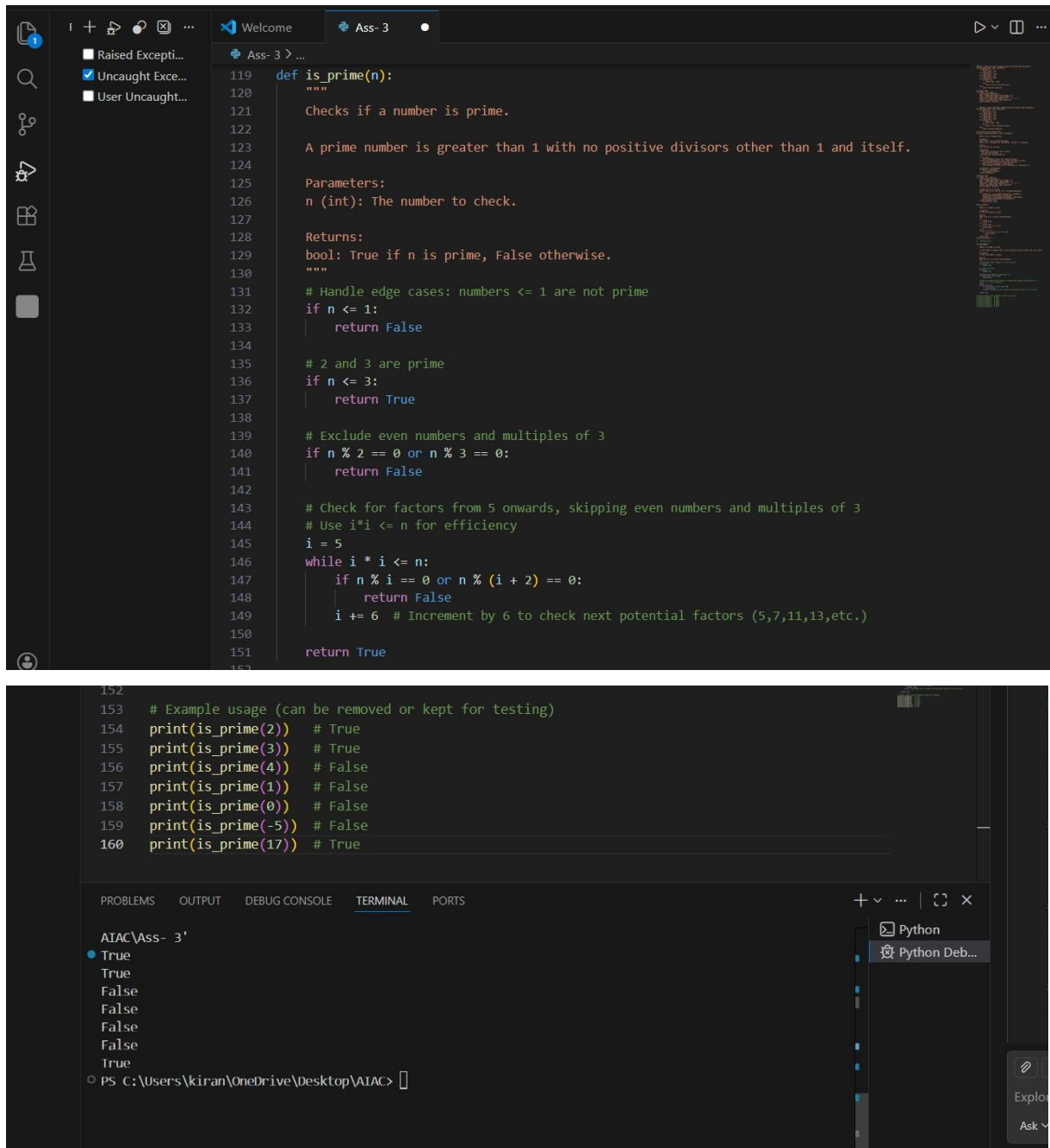
`is_prime(17)` → True

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:



```
def is_prime(n):
    """
    Checks if a number is prime.

    A prime number is greater than 1 with no positive divisors other than 1 and itself.

    Parameters:
    n (int): The number to check.

    Returns:
    bool: True if n is prime, False otherwise.
    """
    # Handle edge cases: numbers <= 1 are not prime
    if n <= 1:
        return False

    # 2 and 3 are prime
    if n <= 3:
        return True

    # Exclude even numbers and multiples of 3
    if n % 2 == 0 or n % 3 == 0:
        return False

    # Check for factors from 5 onwards, skipping even numbers and multiples of 3
    # Use i*i <= n for efficiency
    i = 5
    while i * i <= n:
        if n % i == 0 or n % (i + 2) == 0:
            return False
        i += 6 # Increment by 6 to check next potential factors (5,7,11,13,etc.)

    return True
```

```
# Example usage (can be removed or kept for testing)
print(is_prime(2)) # True
print(is_prime(3)) # True
print(is_prime(4)) # False
print(is_prime(1)) # False
print(is_prime(0)) # False
print(is_prime(-5)) # False
print(is_prime(17)) # True
```

PROBLEMS OUTPUT DEBUG CONSOLE **TERMINAL** PORTS

AIAC\Ass- 3'

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

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

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.

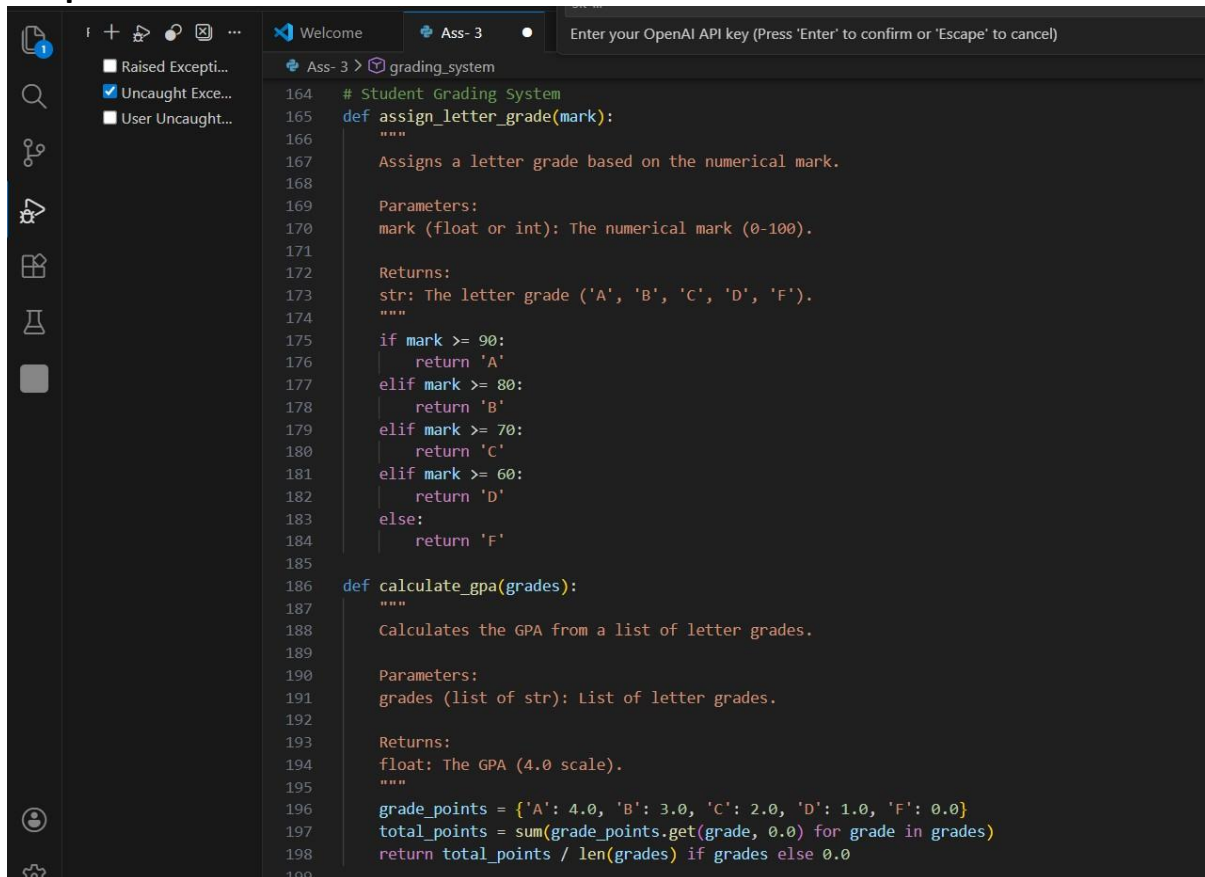
Expected Output-4

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

Prompt 1:

Create a student grading system.

Output:



The screenshot shows a code editor with a dark theme. On the left is a sidebar with icons for Explorer, Search, Source Control, Run and Debug, Extensions, Testing, and Settings. The main editor area displays Python code for a student grading system. The code includes two functions: `assign_letter_grade` and `calculate_gpa`. The `assign_letter_grade` function takes a numerical mark and returns a letter grade (A, B, C, D, F) based on a set of thresholds. The `calculate_gpa` function takes a list of letter grades and returns the GPA on a 4.0 scale. The code is well-commented and includes docstrings for both functions.

```
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
```



```
Ass- 3 > ...
200 def grading_system():
201     """
202     Interactive student grading system.
203     Prompts for student marks, assigns grades, and computes GPA.
204     """
205     students = []
206     while True:
207         name = input("Enter student name (or 'quit' to finish): ")
208         if name.lower() == 'quit':
209             break
210         try:
211             mark = float(input(f"Enter mark for {name} (0-100): "))
212             if not (0 <= mark <= 100):
213                 print("Mark must be between 0 and 100.")
214                 continue
215             grade = assign_letter_grade(mark)
216             students.append({'name': name, 'mark': mark, 'grade': grade})
217             print(f"{name}: Mark {mark}, Grade {grade}")
218         except ValueError:
219             print("Invalid input. Please enter a number.")
220
221     if students:
222         grades = [s['grade'] for s in students]
223         gpa = calculate_gpa(grades)
224         print(f"\nOverall GPA: {gpa:.2f}")
225         print("Student Summary:")
226         for s in students:
227             print(f"{s['name']}: {s['mark']} -> {s['grade']}")
228     else:
229         print("No students entered.")
230
231 # Example usage
232 if __name__ == "__main__":
233     # Existing calculator and sorting code can run here
234     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): rajuu
Enter mark for rajuu (0-100): 44
rajuu: 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
rajuu: 44.0 -> F
suma: 58.0 -> F
PS C:\Users\kiran\OneDrive\Desktop\AIAC>
```

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

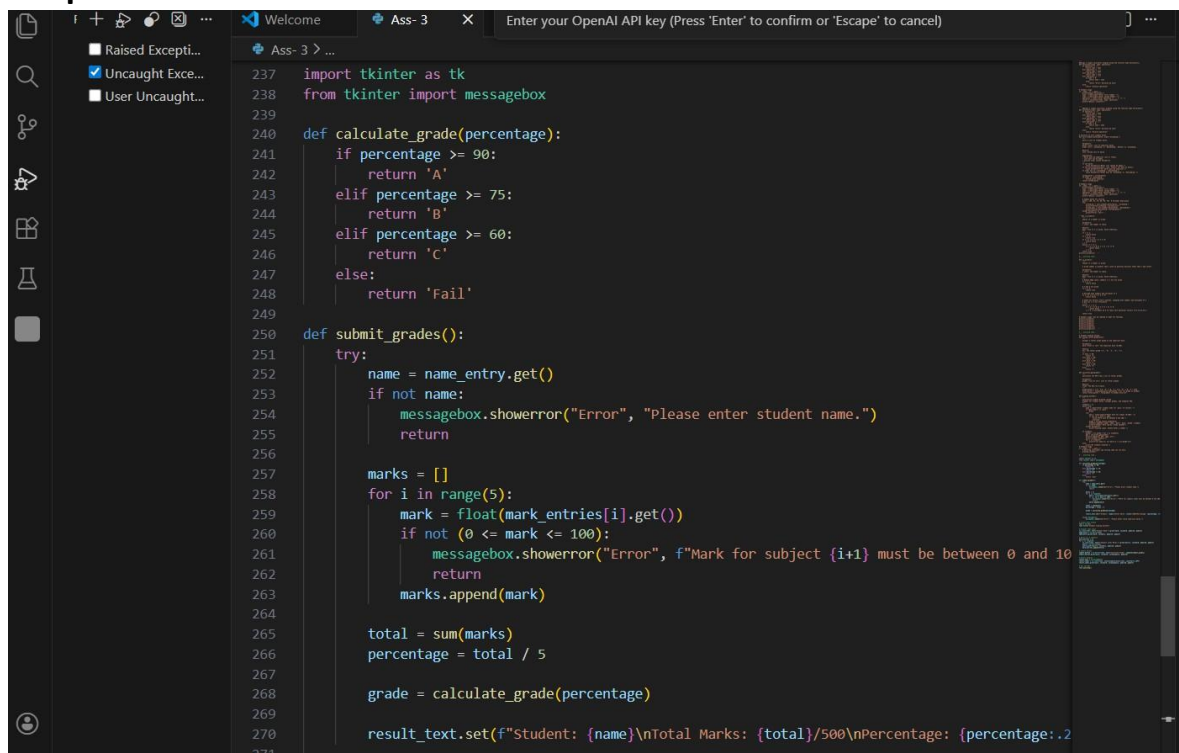
B: ≥ 75

C: ≥ 60

Fail: < 60

- Display results clearly

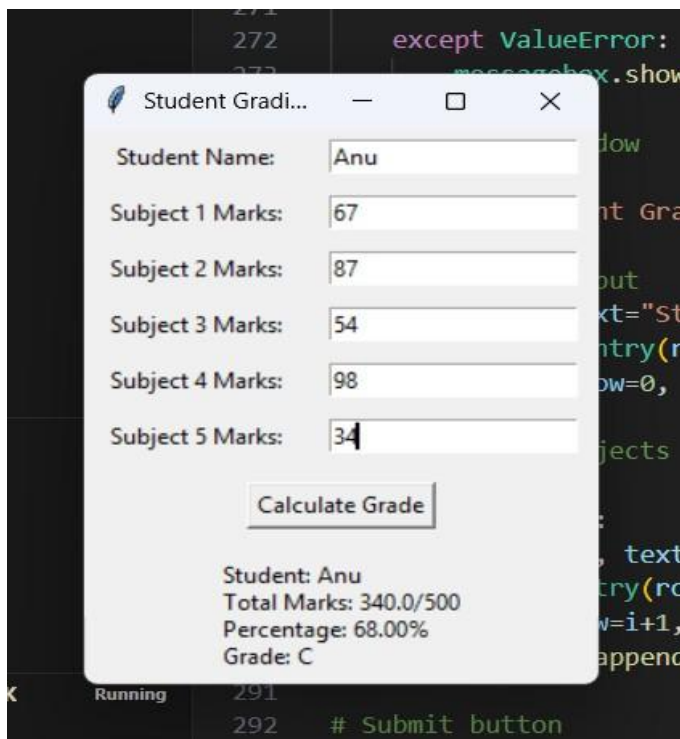
Output:



The screenshot shows a code editor with a dark theme. The left sidebar contains icons for Explorer, Search, Source Control, Run and Debug, and Extensions. The main editor area displays Python code for a grade calculation application. The code includes imports for tkinter and messagebox, a function to calculate grades based on percentages, and a function to submit grades and calculate the total percentage. The code is as follows:

```
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}/500\nPercentage: {percentage:.2%}")
271
```

```
250 def submit_grades():
271     except ValueError:
272         messagebox.showerror("Error", "Please enter valid numerical marks.")
273
274 # Create main window
275 root = tk.Tk()
276 root.title("Student Grading System")
277
278 # Student name input
279 tk.Label(root, text="Student Name:").grid(row=0, column=0, padx=10, pady=5)
280 name_entry = tk.Entry(root)
281 name_entry.grid(row=0, column=1, padx=10, pady=5)
282
283 # Marks for 5 subjects
284 mark_entries = []
285 for i in range(5):
286     tk.Label(root, text=f"Subject {i+1} Marks:").grid(row=i+1, column=0, padx=10, pady=5)
287     entry = tk.Entry(root)
288     entry.grid(row=i+1, column=1, padx=10, pady=5)
289     mark_entries.append(entry)
290
291 # Submit button
292 submit_button = tk.Button(root, text="Calculate Grade", command=submit_grades)
293 submit_button.grid(row=6, column=0, columnspan=2, pady=10)
294
295 # Result display
296 result_text = tk.StringVar()
297 result_label = tk.Label(root, textvariable=result_text, justify=tk.LEFT)
298 result_label.grid(row=7, column=0, columnspan=2, padx=10, pady=5)
299
300 # Run the GUI
301 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): ")
```

```
339
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
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
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
```

```

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")

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```

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

```

Python
Python Deb...
Python Deb...

```

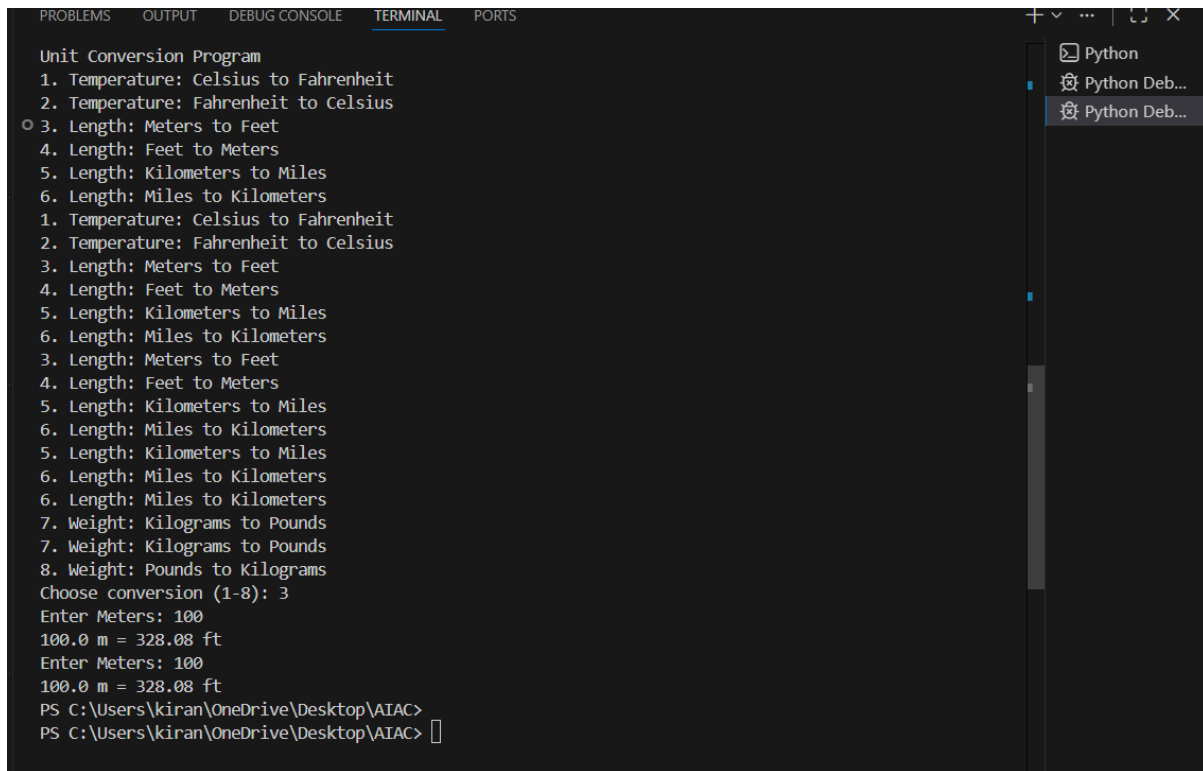
elif choice == '5':
    km = float(input("Enter Kilometers: "))
    result = kilometers_to_miles(km)
    print(f"{km} km = {result:.2f} miles")
elif choice == '6':
    miles = float(input("Enter Miles: "))
    result = miles_to_kilometers(miles)
    print(f"{miles} miles = {result:.2f} km")
elif choice == '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()

```

```
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
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.