AI ASSISTANT CODING

Assignment – 3.2

Name: K. Shailaja
Hall Ticket No.: 2303A52155

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.

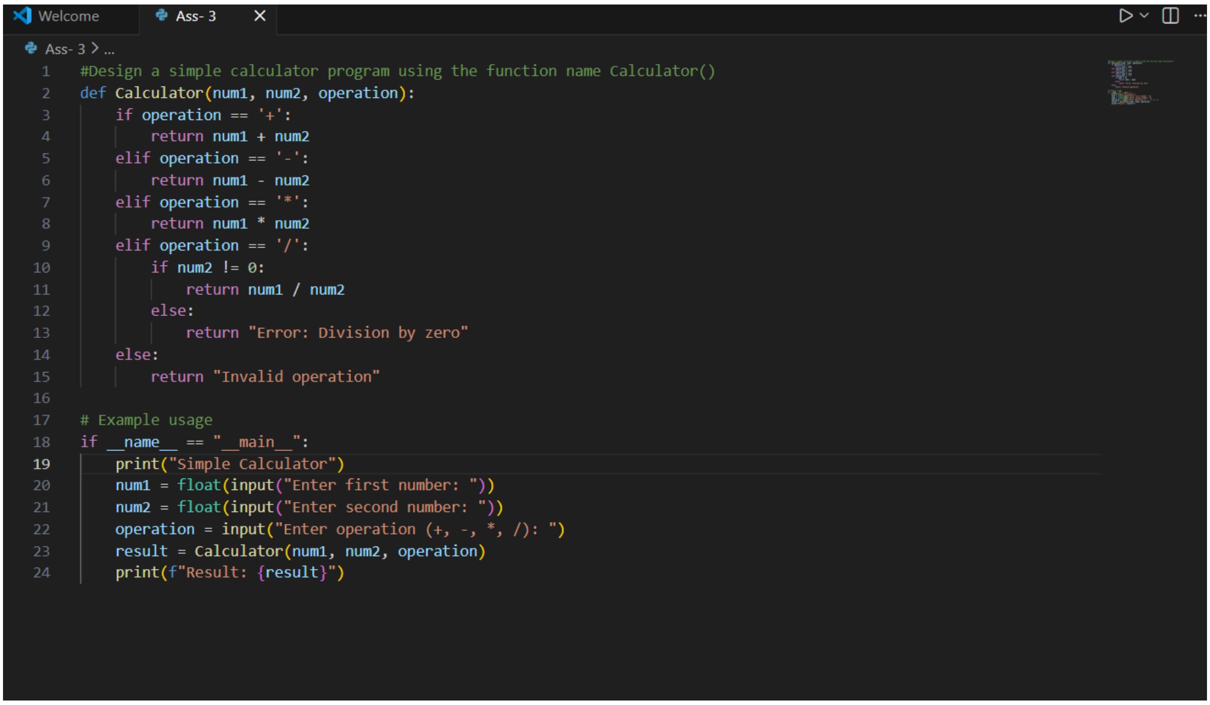
Lab Outcomes (LOs):
After completing this lab, students will be able to:
- Generate Python code using Google Gemini in Google Colab.
- Analyse the effectiveness of code explanations and suggestions by Gemini.
- Set up and use Cursor AI for AI-powered coding assistance.
- Evaluate and refactor code using 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.

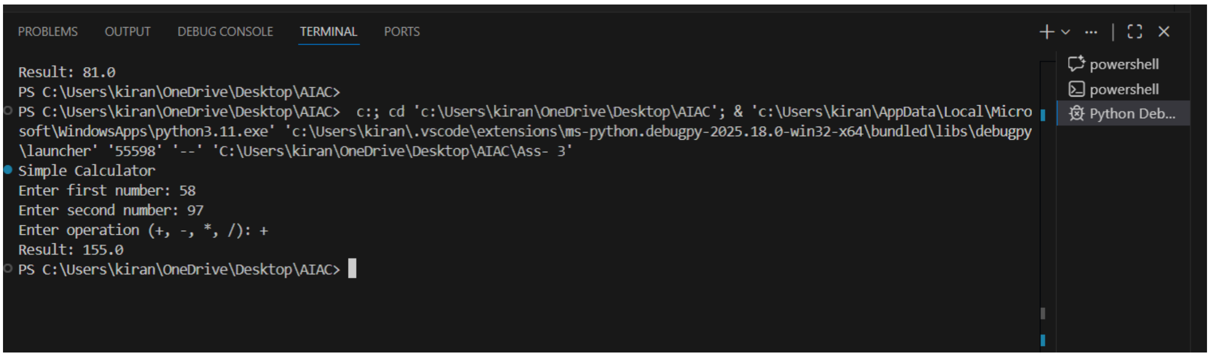
Expected Output-1
- Comparison showing improvement in AI-generated calculator logic and structure.

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

Output:



```python
#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}")
```



```
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\Micro
soft\WindowsApps\python3.11.exe' 'c:\Users\kiran\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\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.

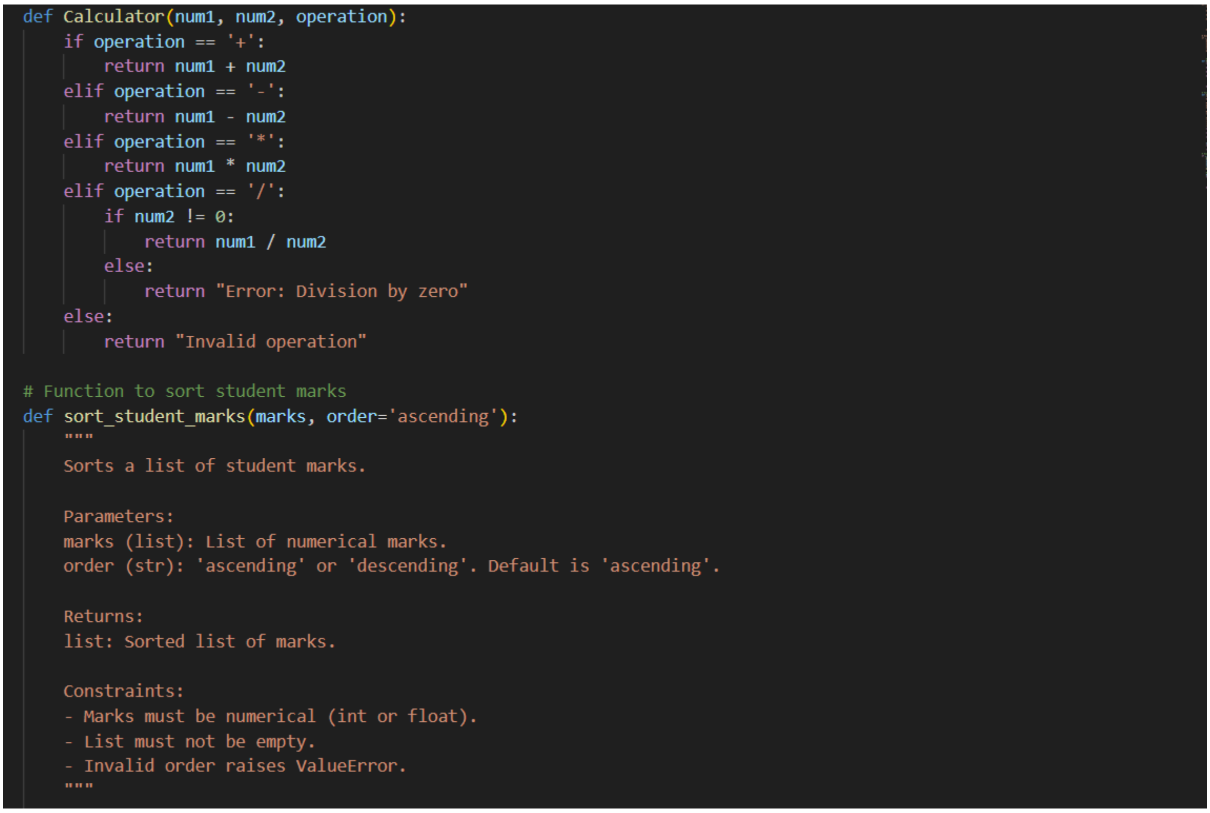
Expected Output-2

- AI-generated sorting function evolves from ambiguous logic to an accurate and efficient implementation.

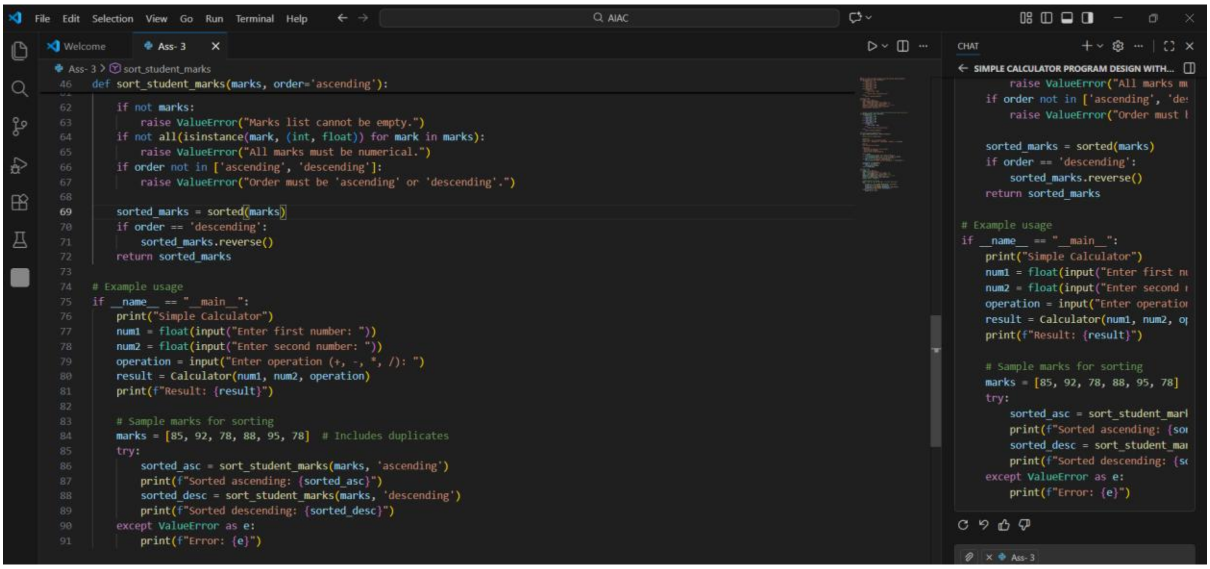
Prompt:

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

Output:



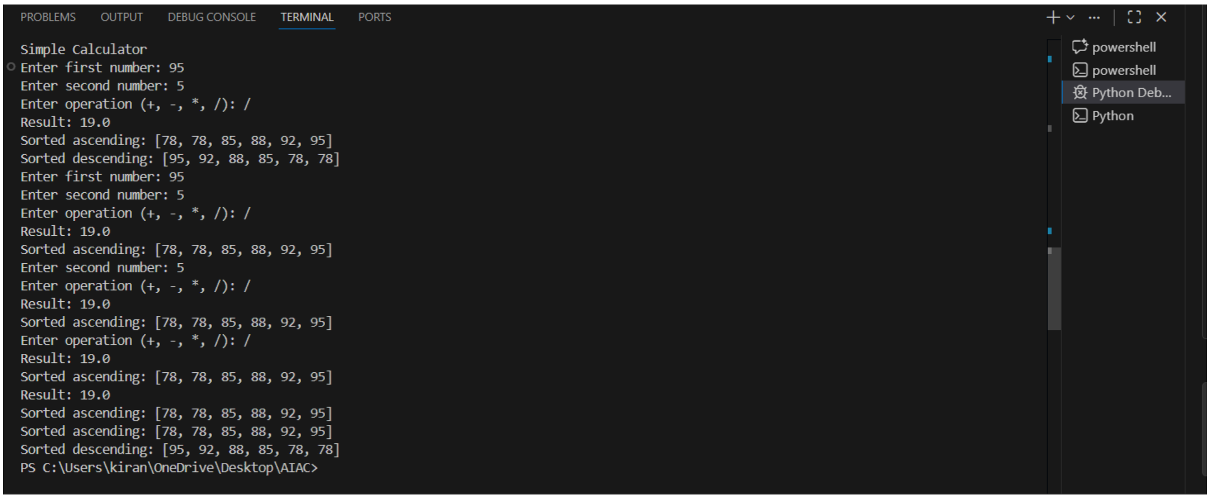
```python
def Calculator(num1, num2, operation):
    if operation == '+':
        return num1 + num2
    elif operation == '-':
        return num1 - num2
    elif operation == '*':
        return num1 * num2
    elif operation == '/':
        if num2 != 0:
            return num1 / num2
        else:
            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.
    """
```



File
Edit
Selection
View
Go
Run
Terminal
Help
AIAC
Welcome
Ass- 3
Ass- 3 > sort_student_marks
CHAT
SIMPLE CALCULATOR PROGRAM DESIGN WITH...
Ass- 3

```python
def sort_student_marks(marks, order='ascending'):
        if not marks:
            raise ValueError("Marks list cannot be empty.")
        if not all(isinstance(mark, (int, float)) for mark in marks):
            raise ValueError("All marks must be numerical.")
        if order not in ['ascending', 'descending']:
            raise ValueError("Order must be 'ascending' or 'descending'.")

        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 number: "))
    num2 = float(input("Enter second number: "))
    operation = input("Enter operation (+, -, *, /): ")
    result = Calculator(num1, num2, operation)
    print(f"Result: {result}")

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

```python
            raise ValueError("All marks m
        if order not in ['ascending', 'de
            raise ValueError("Order must b

        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 nu
    num2 = float(input("Enter second n
    operation = input("Enter operation
    result = Calculator(num1, num2, op
    print(f"Result: {result}")

    # Sample marks for sorting
    marks = [85, 92, 78, 88, 95, 78]
    try:
        sorted_asc = sort_student_marl
        print(f"Sorted ascending: {sor
        sorted_desc = sort_student_mar
        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.

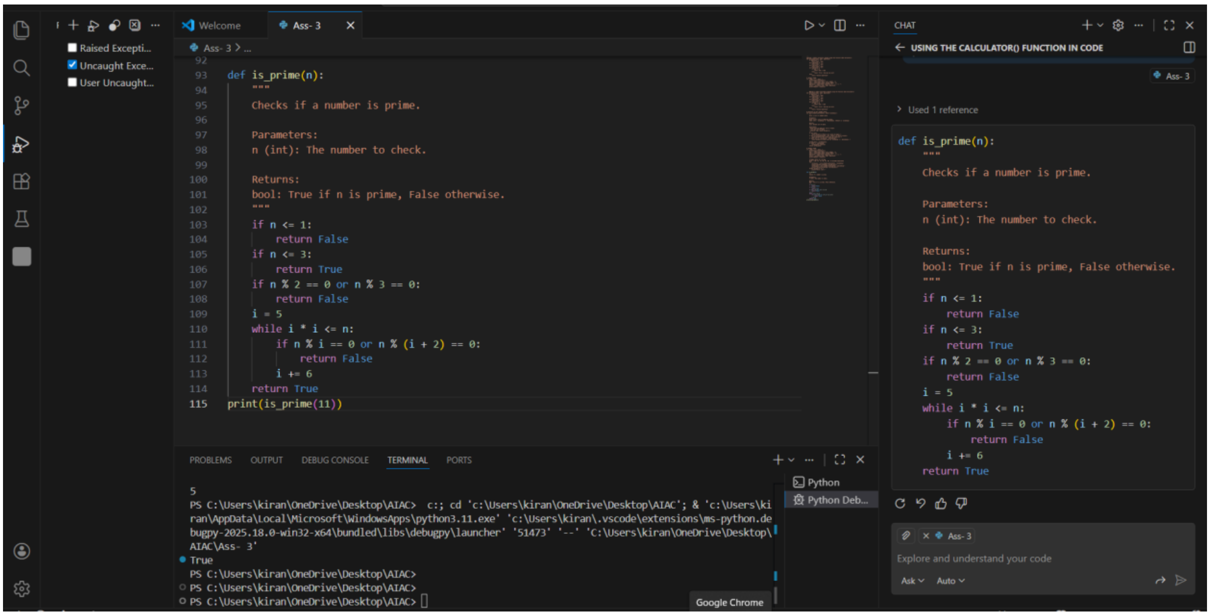
Expected Output-3

- Improved prime-checking function with better edge-case handling.

Prompt 1:

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

Output:



☐ Raised Excepti...
☑ Uncaught Exce...
☐ User Uncaught...

```python
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))
```

```
5
PS C:\Users\kiran\OneDrive\Desktop\AIAC>  c:; cd 'c:\Users\kiran\OneDrive\Desktop\AIAC'; & 'c:\Users\ki
ran\AppData\Local\Microsoft\WindowsApps\python3.11.exe' 'c:\Users\kiran\.vscode\extensions\ms-python.de
bugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '51473' '--' 'C:\Users\kiran\OneDrive\Desktop\
AIAC\Ass- 3'
True
PS C:\Users\kiran\OneDrive\Desktop\AIAC>
PS C:\Users\kiran\OneDrive\Desktop\AIAC>
PS C:\Users\kiran\OneDrive\Desktop\AIAC>
```

```python
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
```

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

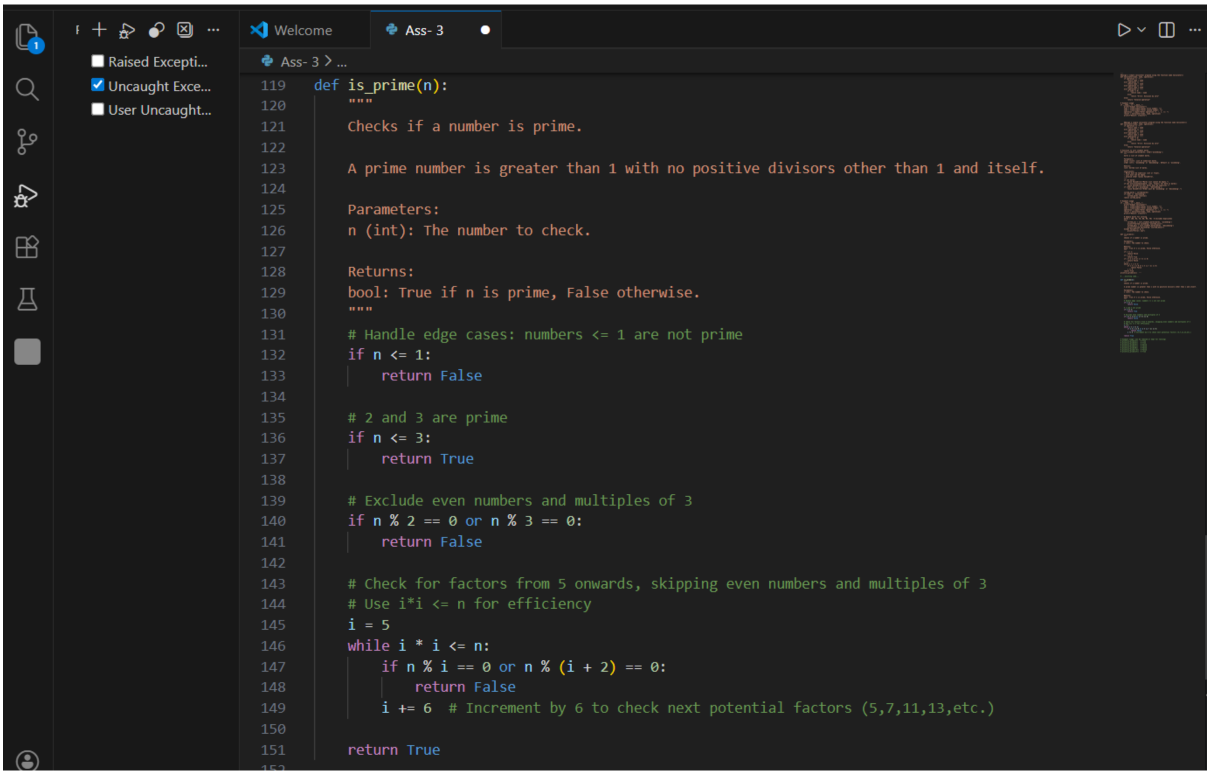
is_prime (17) → True

Requirements:

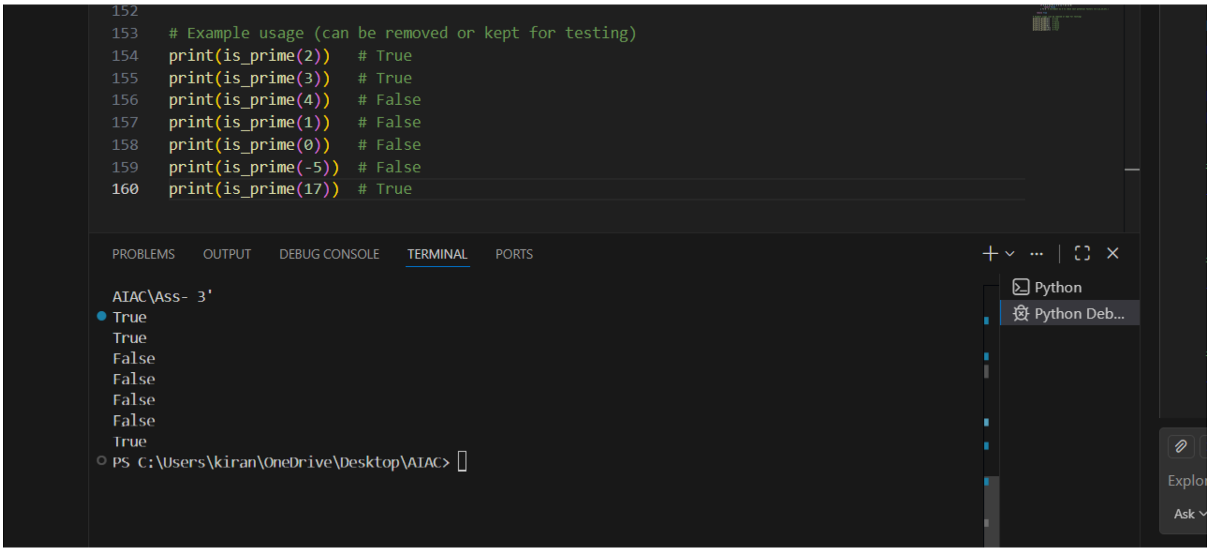
- Return True if the number is prime
- Return False otherwise
- Handle edge cases properly
- Use efficient logic

- 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
```

```
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.

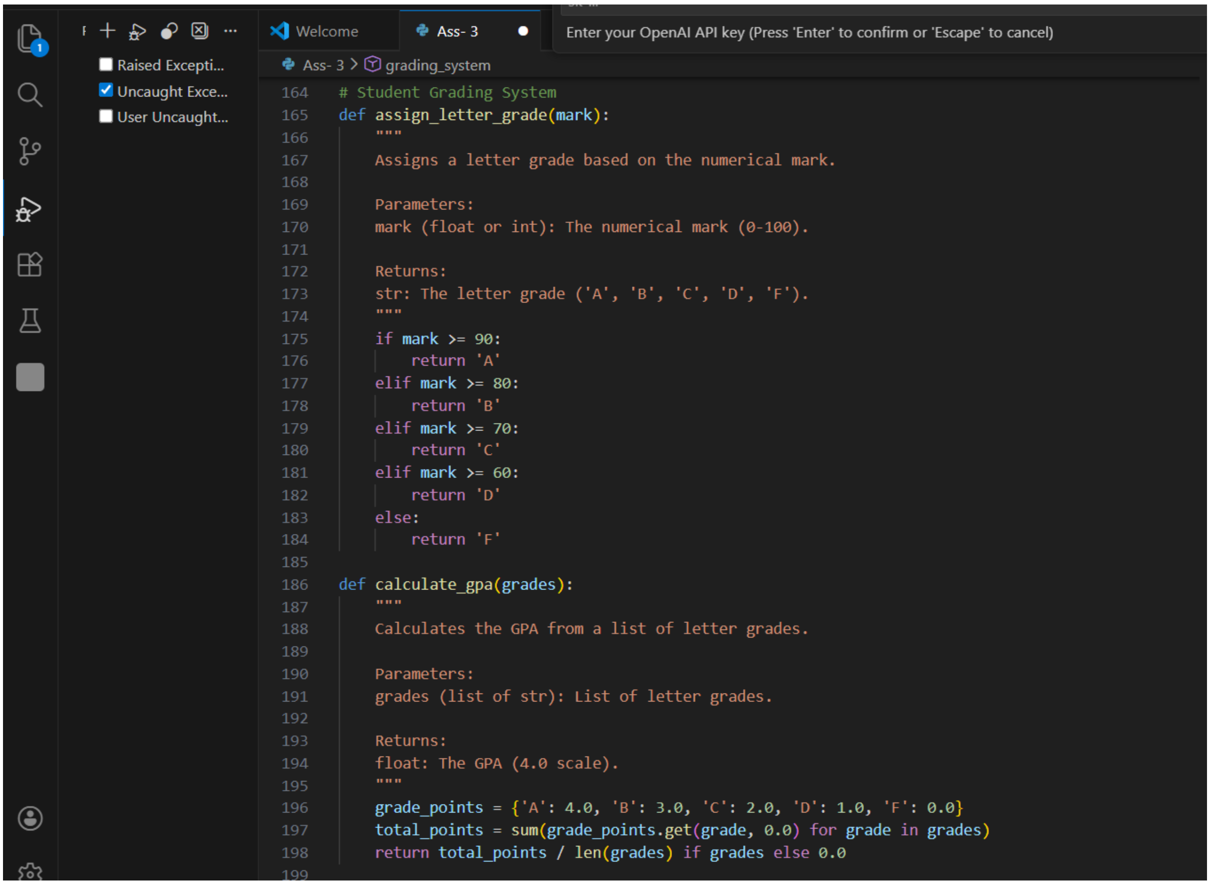
Expected Output-4

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

Prompt 1:

Create a student grading system.

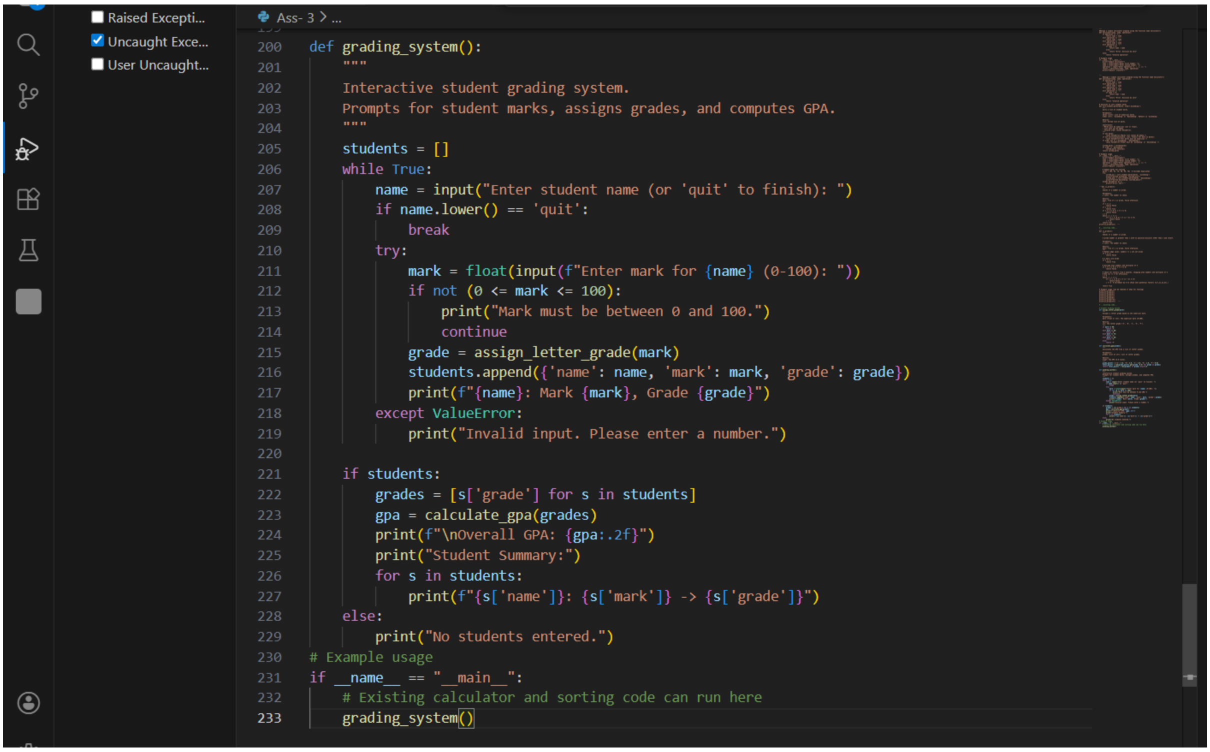
Output:



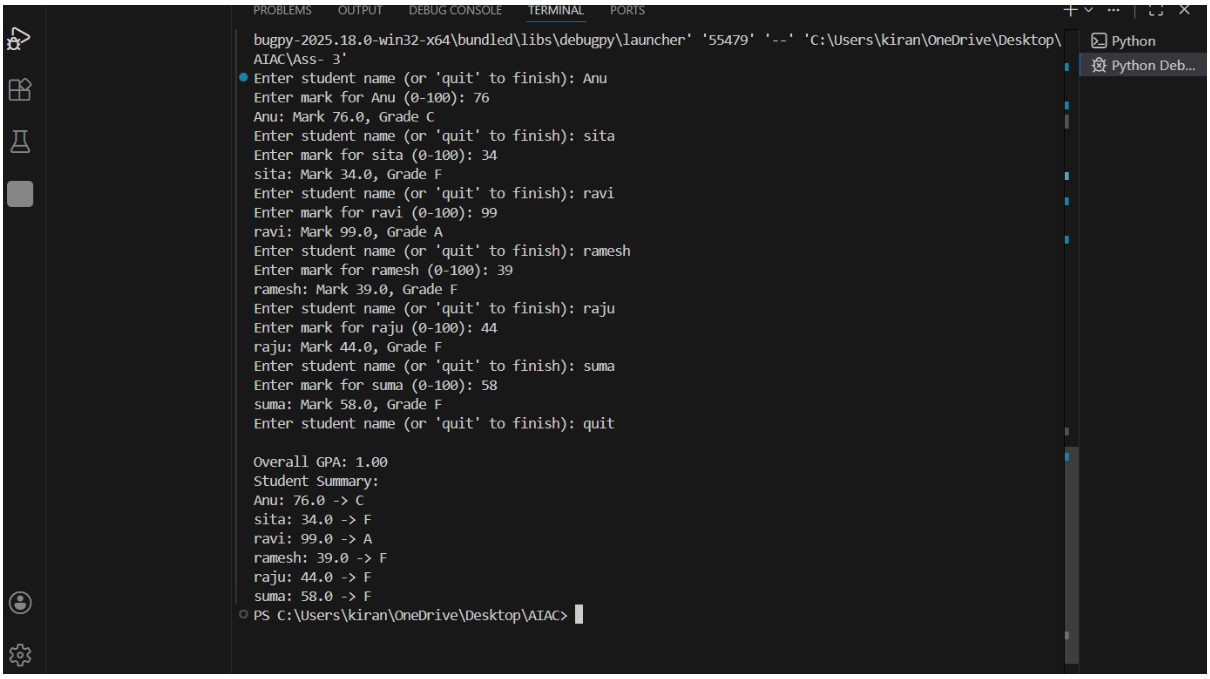
☐ Raised Excepti...
☑ Uncaught Exce...
☐ User Uncaught...

Enter your OpenAI API key (Press 'Enter' to confirm or 'Escape' to cancel)

```
# Student Grading System
def assign_letter_grade(mark):
    """
    Assigns a letter grade based on the numerical mark.

    Parameters:
    mark (float or int): The numerical mark (0-100).

    Returns:
    str: The letter grade ('A', 'B', 'C', 'D', 'F').
    """
    if mark >= 90:
        return 'A'
    elif mark >= 80:
        return 'B'
    elif mark >= 70:
        return 'C'
    elif mark >= 60:
        return 'D'
    else:
        return 'F'

def calculate_gpa(grades):
    """
    Calculates the GPA from a list of letter grades.

    Parameters:
    grades (list of str): List of letter grades.

    Returns:
    float: The GPA (4.0 scale).
    """
    grade_points = {'A': 4.0, 'B': 3.0, 'C': 2.0, 'D': 1.0, 'F': 0.0}
    total_points = sum(grade_points.get(grade, 0.0) for grade in grades)
    return total_points / len(grades) if grades else 0.0
```



```python
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
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

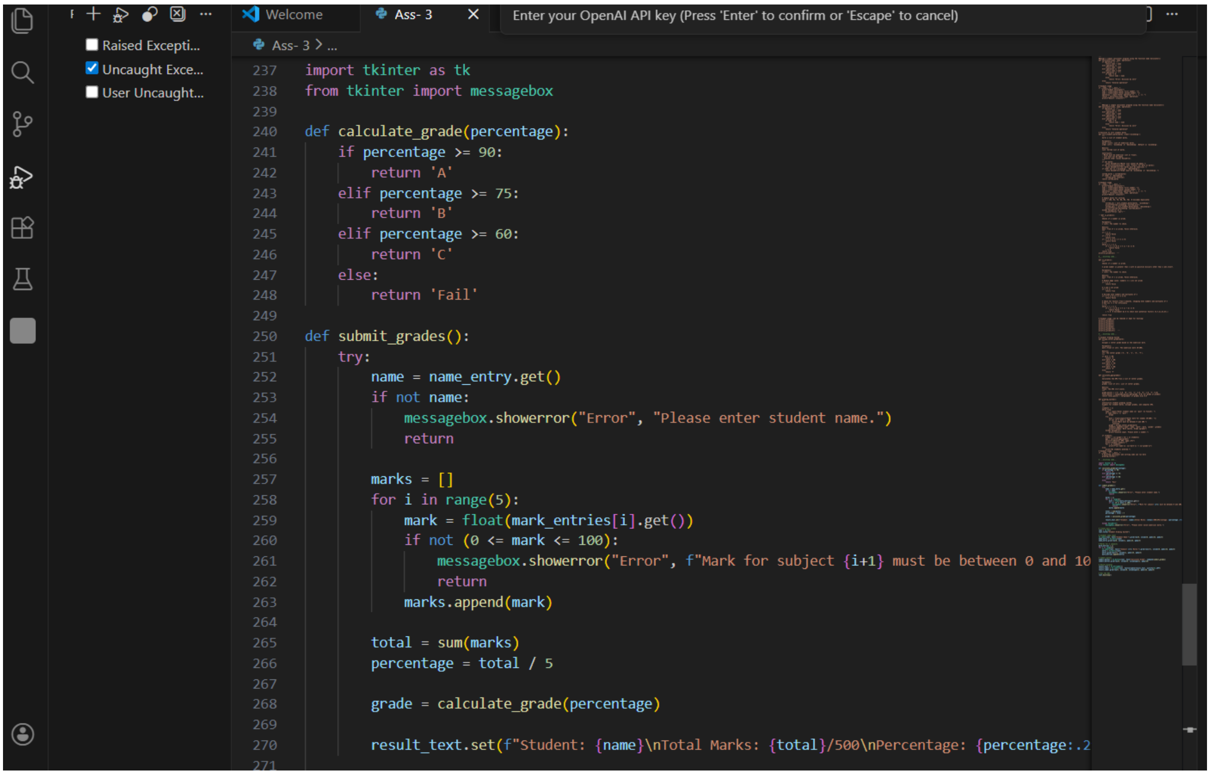
  B: >= 75

  C: >= 60

  Fail: < 60

- Display results clearly

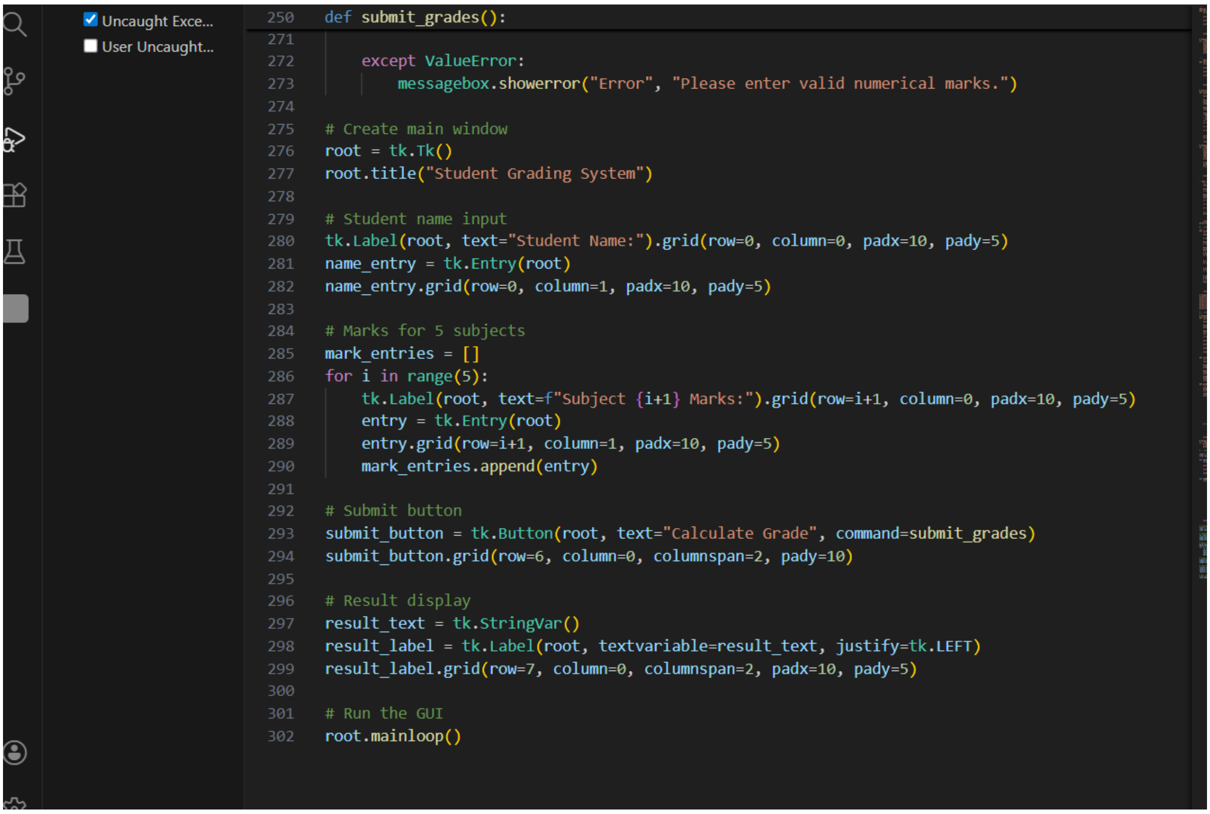
Output:



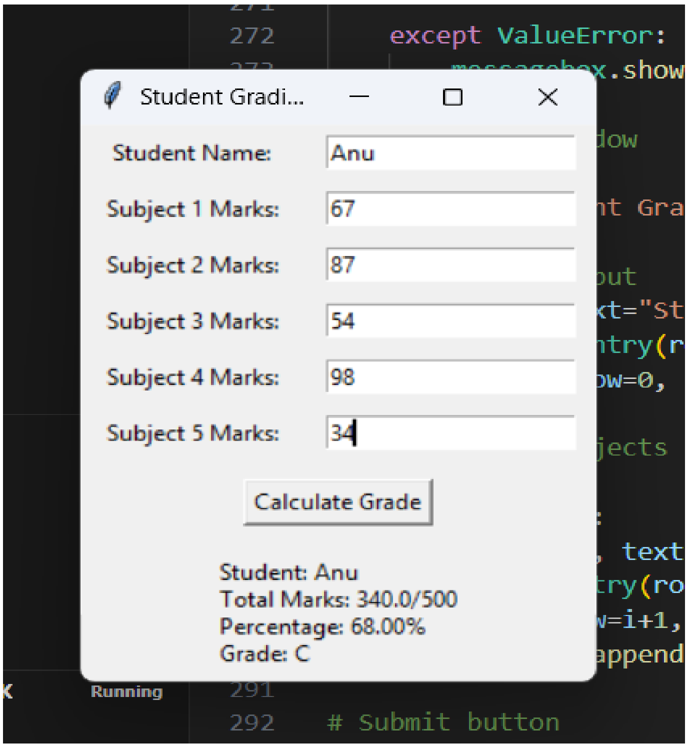
Enter your OpenAI API key (Press 'Enter' to confirm or 'Escape' to cancel)

☐ Raised Excepti...
☑ Uncaught Exce...
☐ User Uncaught...

```python
import tkinter as tk
from tkinter import messagebox

def calculate_grade(percentage):
    if percentage >= 90:
        return 'A'
    elif percentage >= 75:
        return 'B'
    elif percentage >= 60:
        return 'C'
    else:
        return 'Fail'

def submit_grades():
    try:
        name = name_entry.get()
        if not name:
            messagebox.showerror("Error", "Please enter student name.")
            return

        marks = []
        for i in range(5):
            mark = float(mark_entries[i].get())
            if not (0 <= mark <= 100):
                messagebox.showerror("Error", f"Mark for subject {i+1} must be between 0 and 10
                return
            marks.append(mark)

        total = sum(marks)
        percentage = total / 5

        grade = calculate_grade(percentage)

        result_text.set(f"Student: {name}\nTotal Marks: {total}/500\nPercentage: {percentage:.2
```



```python
def submit_grades():

    except ValueError:
        messagebox.showerror("Error", "Please enter valid numerical marks.")

# Create main window
root = tk.Tk()
root.title("Student Grading System")

# Student name input
tk.Label(root, text="Student Name:").grid(row=0, column=0, padx=10, pady=5)
name_entry = tk.Entry(root)
name_entry.grid(row=0, column=1, padx=10, pady=5)

# Marks for 5 subjects
mark_entries = []
for i in range(5):
    tk.Label(root, text=f"Subject {i+1} Marks:").grid(row=i+1, column=0, padx=10, pady=5)
    entry = tk.Entry(root)
    entry.grid(row=i+1, column=1, padx=10, pady=5)
    mark_entries.append(entry)

# Submit button
submit_button = tk.Button(root, text="Calculate Grade", command=submit_grades)
submit_button.grid(row=6, column=0, columnspan=2, pady=10)

# Result display
result_text = tk.StringVar()
result_label = tk.Label(root, textvariable=result_text, justify=tk.LEFT)
result_label.grid(row=7, column=0, columnspan=2, padx=10, pady=5)

# Run the GUI
root.mainloop()
```



Student Name: Anu
Subject 1 Marks: 67
Subject 2 Marks: 87
Subject 3 Marks: 54
Subject 4 Marks: 98
Subject 5 Marks: 34

```
Student: Anu
Total Marks: 340.0/500
Percentage: 68.00%
Grade: C
```

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.

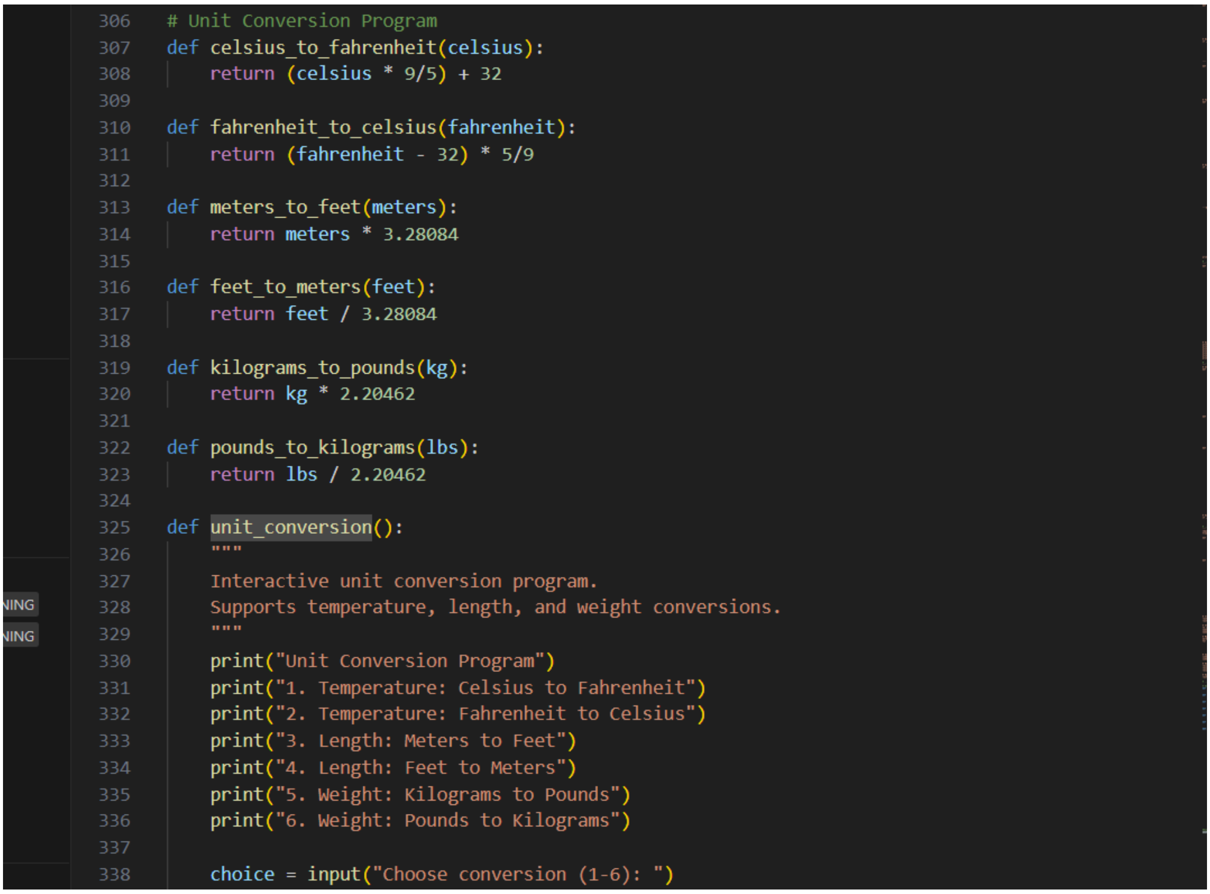
Expected Output-5

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

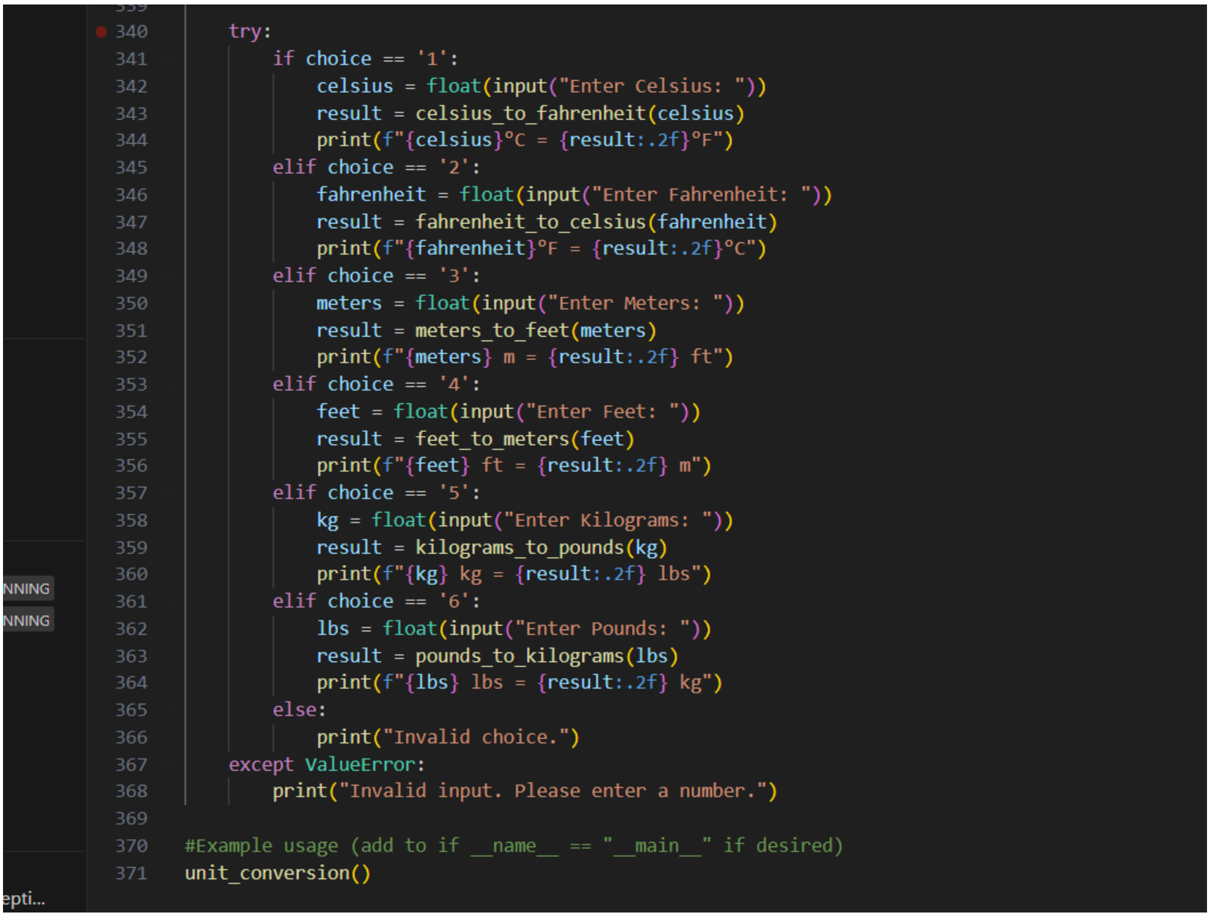
Prompt:

Write a unit conversion program.

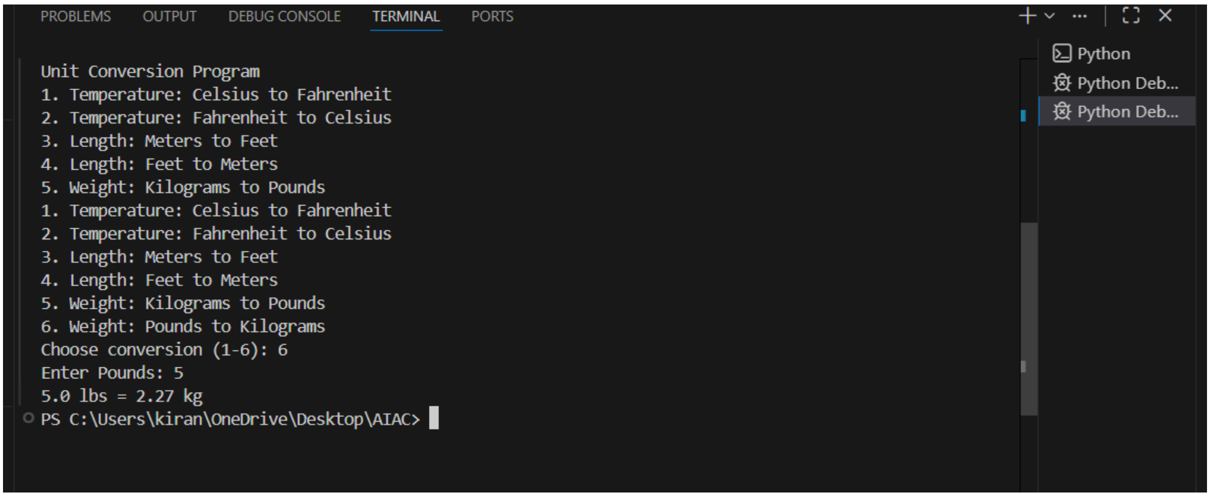
Output:



```python
# Unit Conversion Program
def celsius_to_fahrenheit(celsius):
    return (celsius * 9/5) + 32

def fahrenheit_to_celsius(fahrenheit):
    return (fahrenheit - 32) * 5/9

def meters_to_feet(meters):
    return meters * 3.28084

def feet_to_meters(feet):
    return feet / 3.28084

def kilograms_to_pounds(kg):
    return kg * 2.20462

def pounds_to_kilograms(lbs):
    return lbs / 2.20462

def unit_conversion():
    """
    Interactive unit conversion program.
    Supports temperature, length, and weight conversions.
    """
    print("Unit Conversion Program")
    print("1. Temperature: Celsius to Fahrenheit")
    print("2. Temperature: Fahrenheit to Celsius")
    print("3. Length: Meters to Feet")
    print("4. Length: Feet to Meters")
    print("5. Weight: Kilograms to Pounds")
    print("6. Weight: Pounds to Kilograms")

    choice = input("Choose conversion (1-6): ")
```



```
    try:
        if choice == '1':
            celsius = float(input("Enter Celsius: "))
            result = celsius_to_fahrenheit(celsius)
            print(f"{celsius}°C = {result:.2f}°F")
        elif choice == '2':
            fahrenheit = float(input("Enter Fahrenheit: "))
            result = fahrenheit_to_celsius(fahrenheit)
            print(f"{fahrenheit}°F = {result:.2f}°C")
        elif choice == '3':
            meters = float(input("Enter Meters: "))
            result = meters_to_feet(meters)
            print(f"{meters} m = {result:.2f} ft")
        elif choice == '4':
            feet = float(input("Enter Feet: "))
            result = feet_to_meters(feet)
            print(f"{feet} ft = {result:.2f} m")
        elif choice == '5':
            kg = float(input("Enter Kilograms: "))
            result = kilograms_to_pounds(kg)
            print(f"{kg} kg = {result:.2f} lbs")
        elif choice == '6':
            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()
```

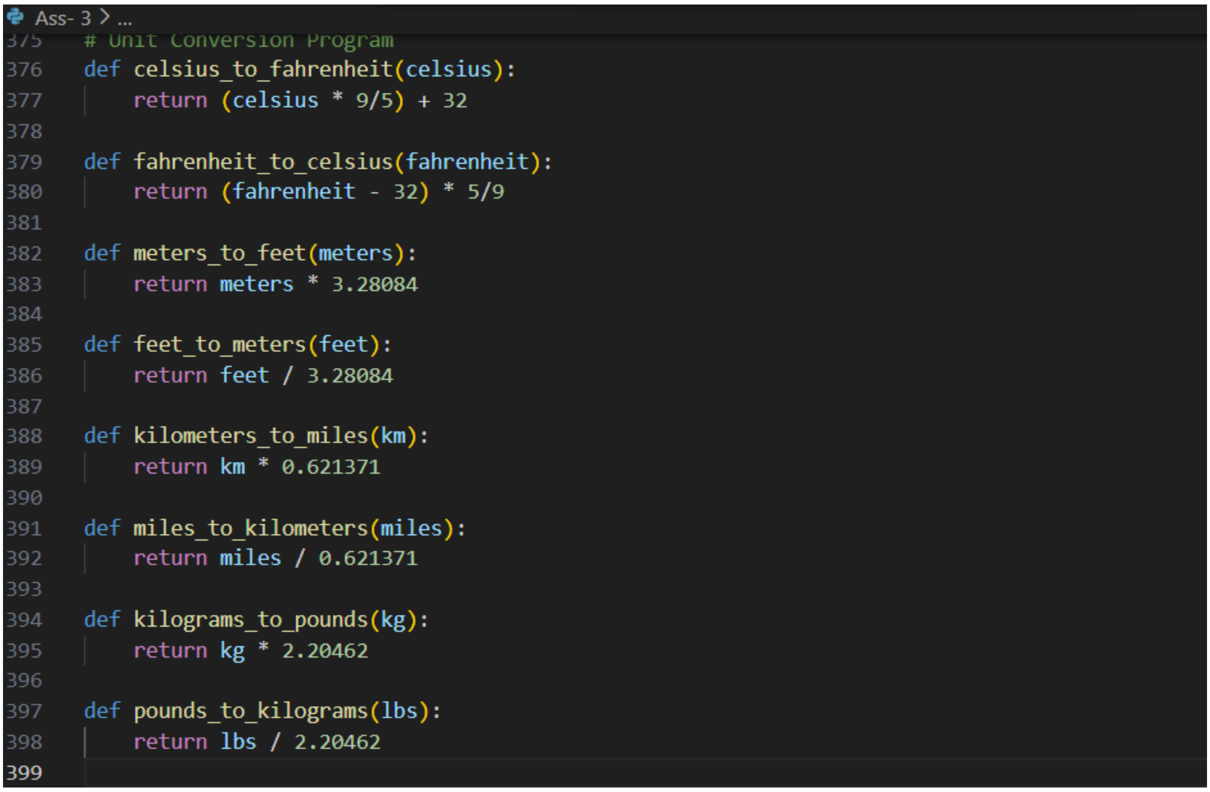


```
Unit Conversion Program
1. Temperature: Celsius to Fahrenheit
2. Temperature: Fahrenheit to Celsius
3. Length: Meters to Feet
4. Length: Feet to Meters
5. Weight: Kilograms to Pounds
1. Temperature: Celsius to Fahrenheit
2. Temperature: Fahrenheit to Celsius
3. Length: Meters to Feet
4. Length: Feet 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>
```

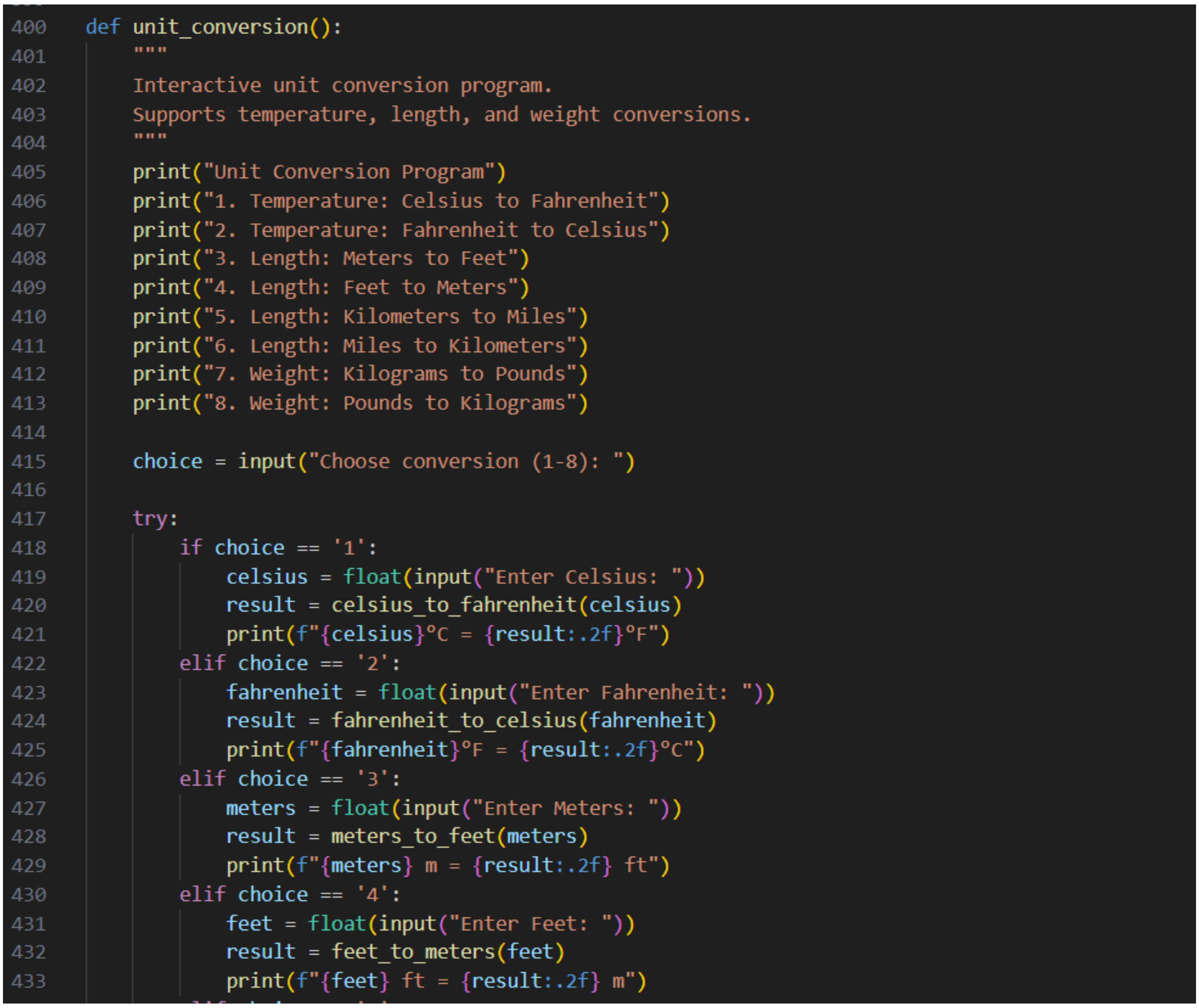
Prompt 2:

Convert kilometers to miles.

Output:



```python
# Unit Conversion Program
def celsius_to_fahrenheit(celsius):
    return (celsius * 9/5) + 32

def fahrenheit_to_celsius(fahrenheit):
    return (fahrenheit - 32) * 5/9

def meters_to_feet(meters):
    return meters * 3.28084

def feet_to_meters(feet):
    return feet / 3.28084

def kilometers_to_miles(km):
    return km * 0.621371

def miles_to_kilometers(miles):
    return miles / 0.621371

def kilograms_to_pounds(kg):
    return kg * 2.20462

def pounds_to_kilograms(lbs):
    return lbs / 2.20462

```



```python
def unit_conversion():
    """
    Interactive unit conversion program.
    Supports temperature, length, and weight conversions.
    """
    print("Unit Conversion Program")
    print("1. Temperature: Celsius to Fahrenheit")
    print("2. Temperature: Fahrenheit to Celsius")
    print("3. Length: Meters to Feet")
    print("4. Length: Feet to Meters")
    print("5. Length: Kilometers to Miles")
    print("6. Length: Miles to Kilometers")
    print("7. Weight: Kilograms to Pounds")
    print("8. Weight: Pounds to Kilograms")

    choice = input("Choose conversion (1-8): ")

    try:
        if choice == '1':
            celsius = float(input("Enter Celsius: "))
            result = celsius_to_fahrenheit(celsius)
            print(f"{celsius}°C = {result:.2f}°F")
        elif choice == '2':
            fahrenheit = float(input("Enter Fahrenheit: "))
            result = fahrenheit_to_celsius(fahrenheit)
            print(f"{fahrenheit}°F = {result:.2f}°C")
        elif choice == '3':
            meters = float(input("Enter Meters: "))
            result = meters_to_feet(meters)
            print(f"{meters} m = {result:.2f} ft")
        elif choice == '4':
            feet = float(input("Enter Feet: "))
            result = feet_to_meters(feet)
            print(f"{feet} ft = {result:.2f} m")
```

Prompt 3:

Write two Python functions:

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

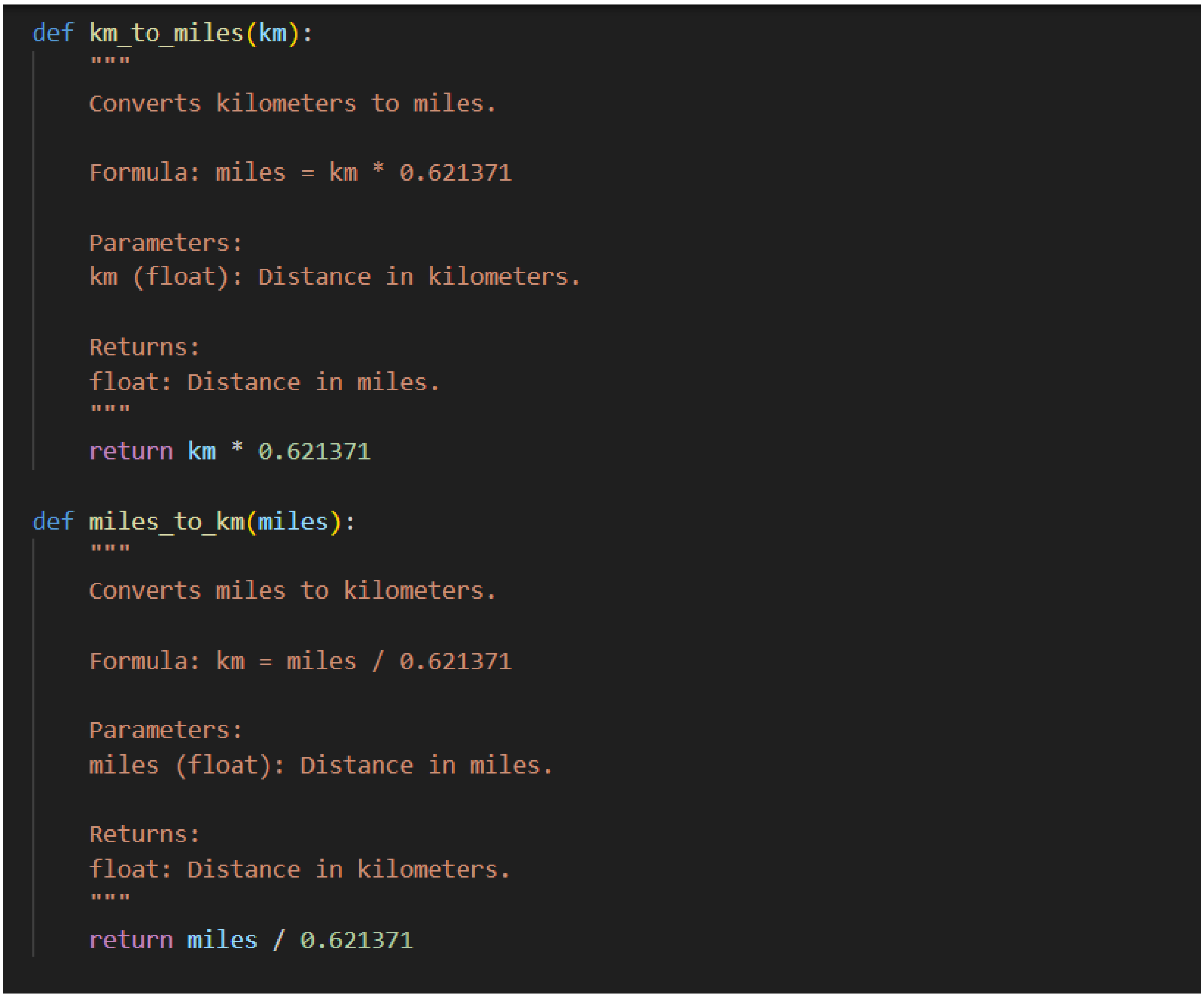
Use formulas:

miles = km * 0.621371

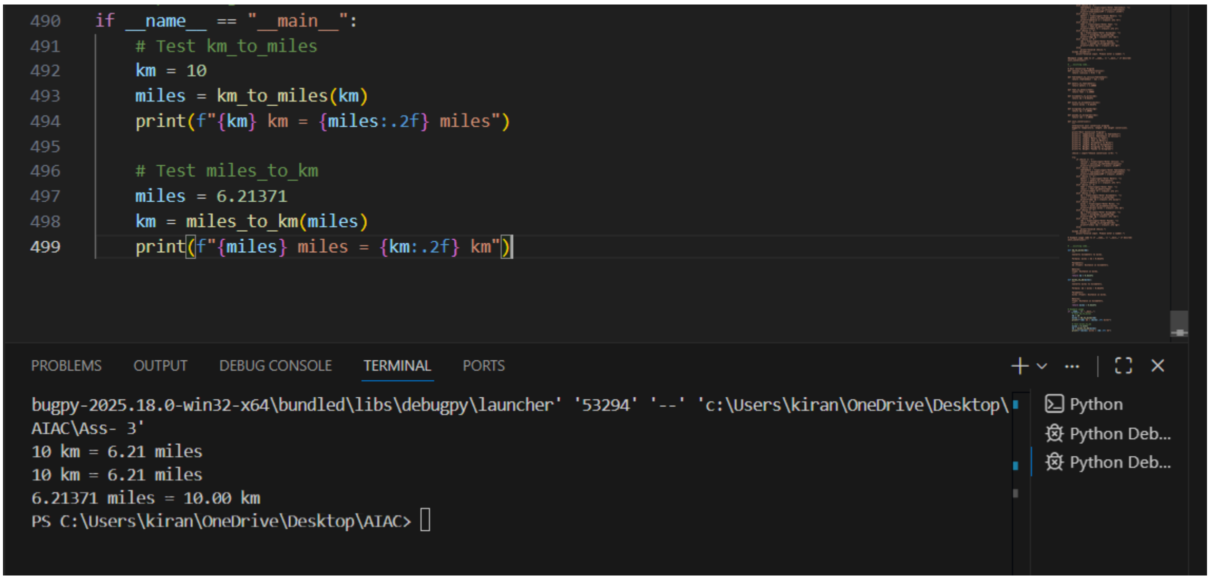
km = miles / 0.621371

Add comments and example usage.

Output:

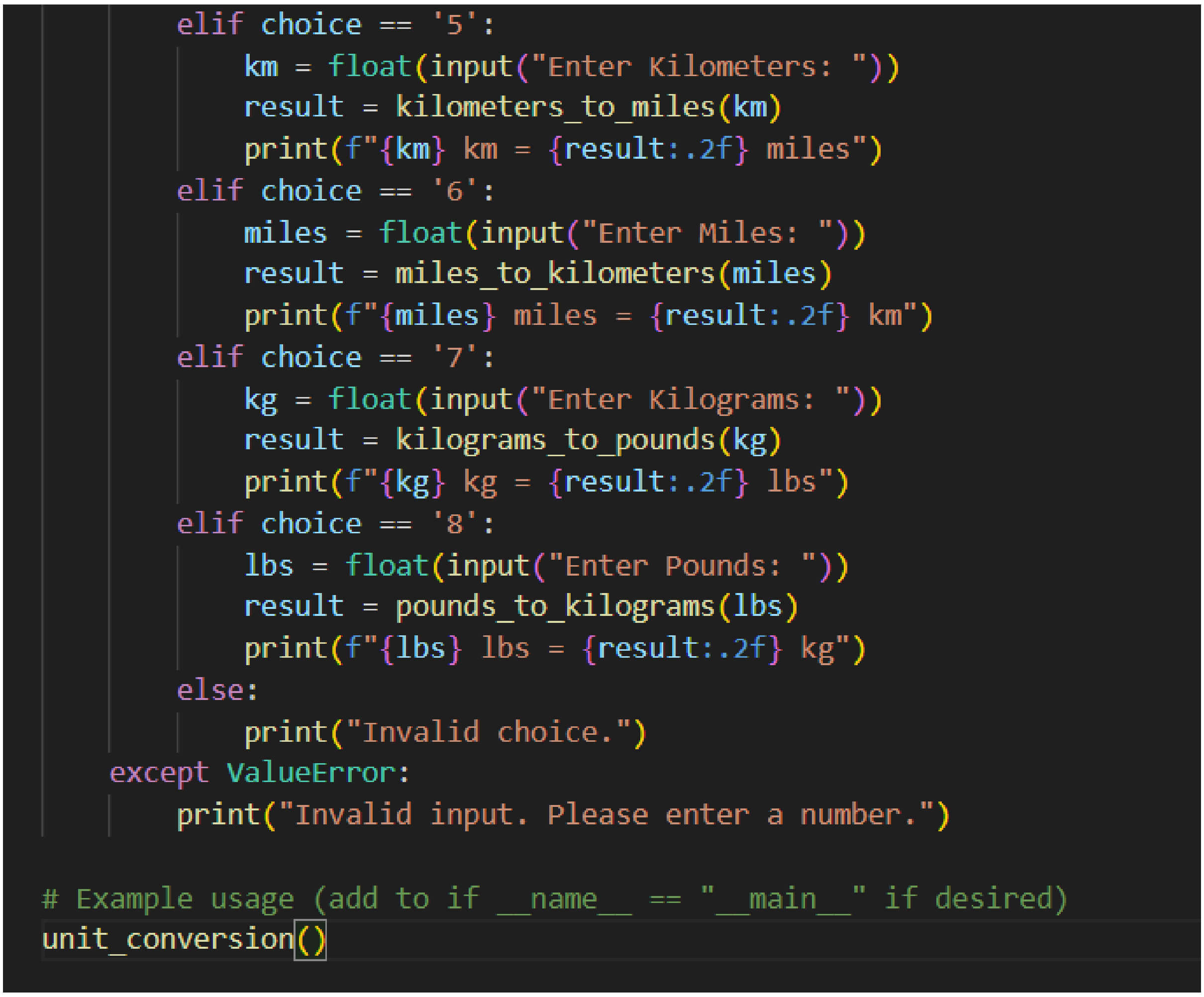


```python
def km_to_miles(km):
    """
    Converts kilometers to miles.

    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
```

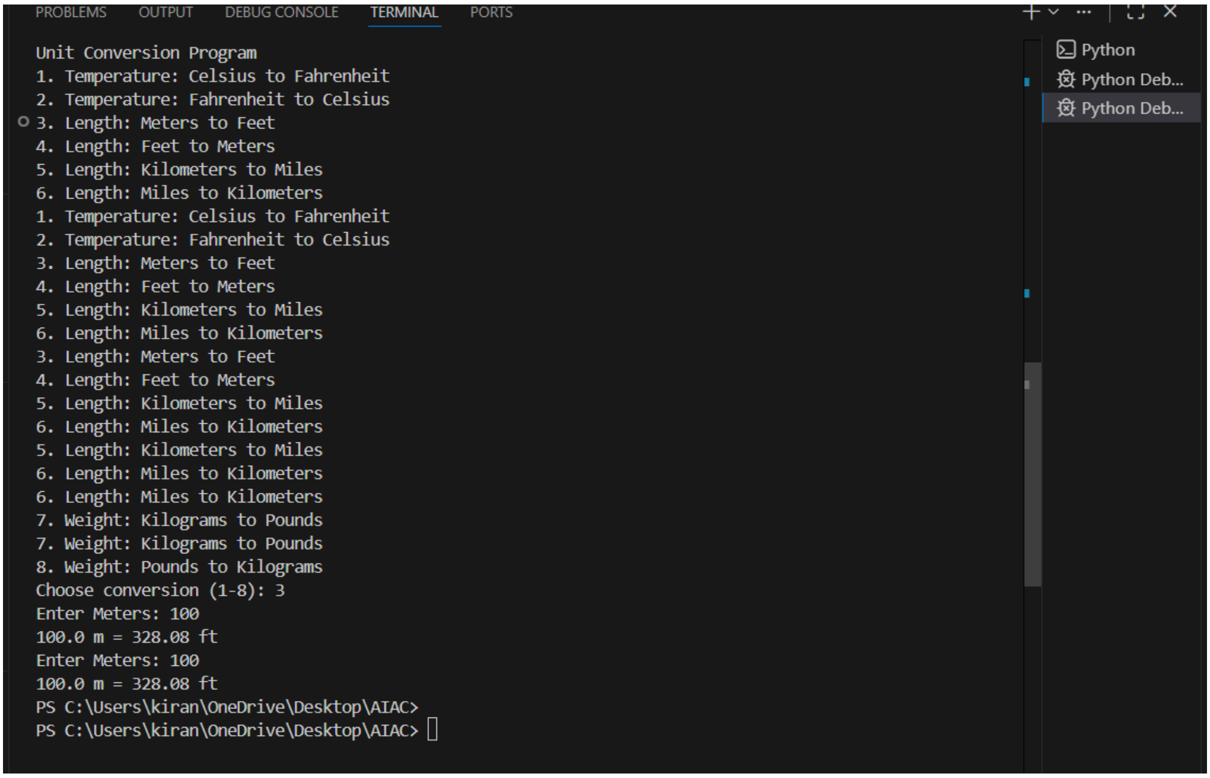


```python
if __name__ == "__main__":
    # Test km_to_miles
    km = 10
    miles = km_to_miles(km)
    print(f"{km} km = {miles:.2f} miles")

    # Test miles_to_km
    miles = 6.21371
    km = miles_to_km(miles)
    print(f"{miles} miles = {km:.2f} km")
```

```
bugpy-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> 
```



```python
            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.