ASSIGNMENT-3

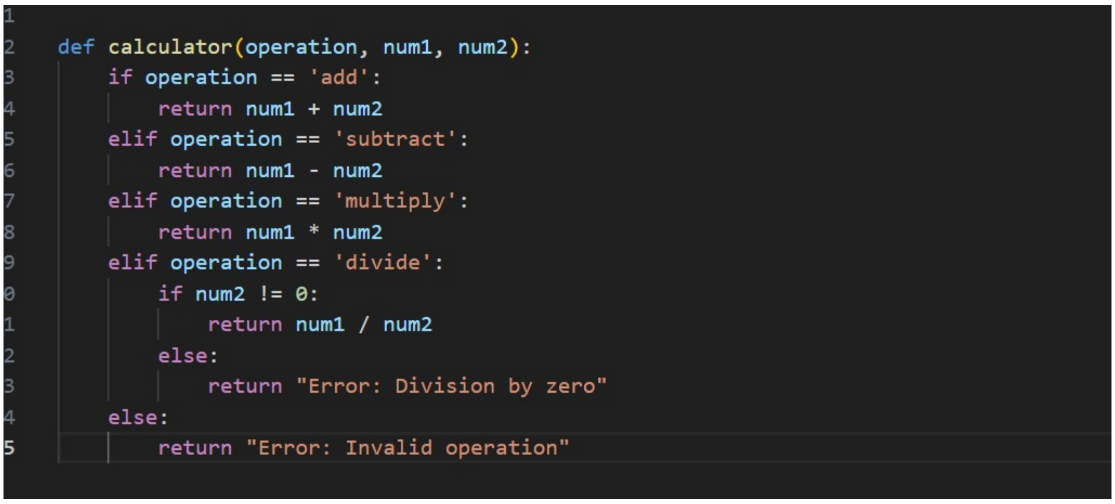
Roll no:2303A52074
Batch-37

Task Description-1

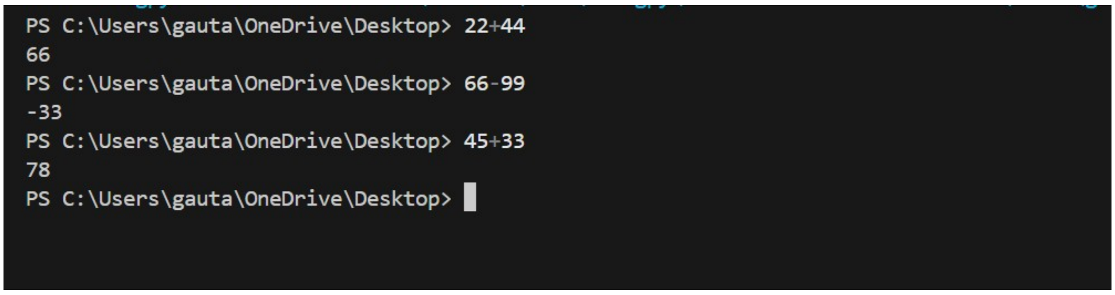
- Progressive Prompting for Calculator Design: Ask the AI to design a simple calculator program by initially providing only the function name. Gradually enhance the prompt by adding comments and usage examples.

Expected Output:
The AI-generated calculator program improves progressively as the prompt becomes more detailed.

PROMPT
Create a calculator function.
# Create a function that performs basic arithmetic operations
# The function should handle addition, subtraction, multiplication, and division

CODE:

```python

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

OUTPUT:

```
PS C:\Users\gauta\OneDrive\Desktop> 22+44
66
PS C:\Users\gauta\OneDrive\Desktop> 66-99
-33
PS C:\Users\gauta\OneDrive\Desktop> 45+33
78
PS C:\Users\gauta\OneDrive\Desktop>
```

Explanation:
We initially asked the AI to create a calculator by providing only the function name. Then, we gradually enhanced the prompt by adding comments describing the required operations.Finally, we included usage examples, which resulted in a more structured and complete 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.

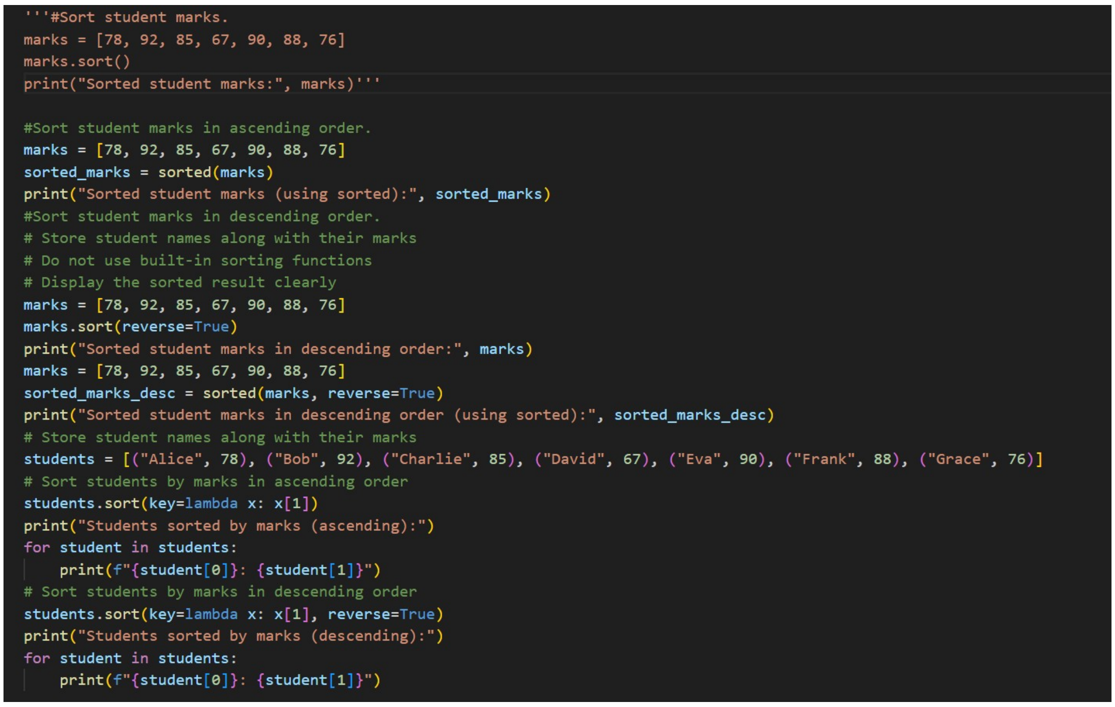
Expected output:

With a vague prompt, the AI produces a basic sorting program without clear order or rules.After refining the prompt, the AI correctly sorts student marks in the specified order.

PROMPT

Sort student marks.
Sort student marks in ascending order.
Sort student marks in descending order.
Store student names along with their marks
Do not use built-in sorting functions
Display the sorted result clearly

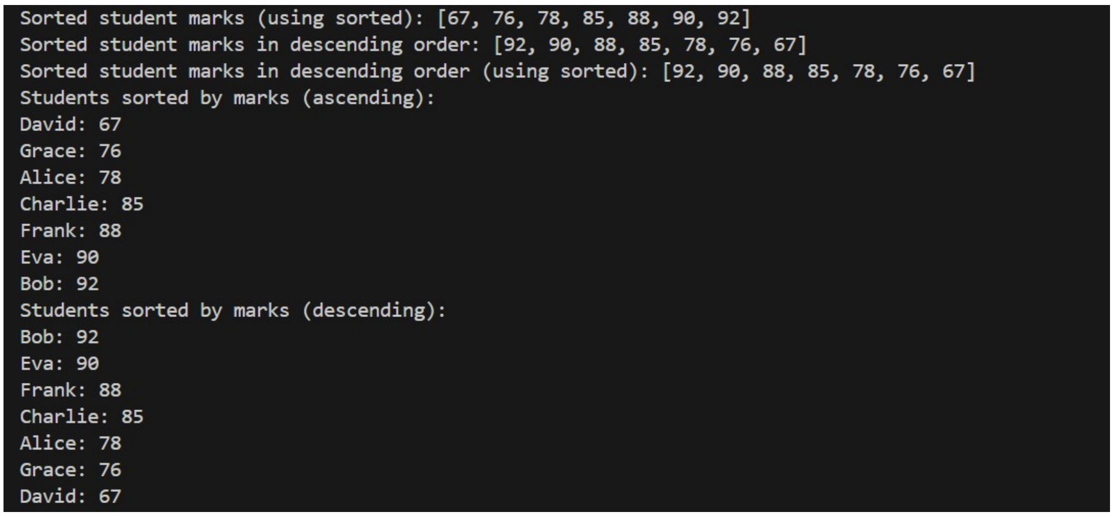
Code:



```python
'''#Sort student marks.
marks = [78, 92, 85, 67, 90, 88, 76]
marks.sort()
print("Sorted student marks:", marks)'''

#Sort student marks in ascending order.
marks = [78, 92, 85, 67, 90, 88, 76]
sorted_marks = sorted(marks)
print("Sorted student marks (using sorted):", sorted_marks)
#Sort student marks in descending order.
# Store student names along with their marks
# Do not use built-in sorting functions
# Display the sorted result clearly
marks = [78, 92, 85, 67, 90, 88, 76]
marks.sort(reverse=True)
print("Sorted student marks in descending order:", marks)
marks = [78, 92, 85, 67, 90, 88, 76]
sorted_marks_desc = sorted(marks, reverse=True)
print("Sorted student marks in descending order (using sorted):", sorted_marks_desc)
# Store student names along with their marks
students = [("Alice", 78), ("Bob", 92), ("Charlie", 85), ("David", 67), ("Eva", 90), ("Frank", 88), ("Grace", 76)]
# Sort students by marks in ascending order
students.sort(key=lambda x: x[1])
print("Students sorted by marks (ascending):")
for student in students:
    print(f"{student[0]}: {student[1]}")
# Sort students by marks in descending order
students.sort(key=lambda x: x[1], reverse=True)
print("Students sorted by marks (descending):")
for student in students:
    print(f"{student[0]}: {student[1]}")
```

Output:



```
Sorted student marks (using sorted): [67, 76, 78, 85, 88, 90, 92]
Sorted student marks in descending order: [92, 90, 88, 85, 78, 76, 67]
Sorted student marks in descending order (using sorted): [92, 90, 88, 85, 78, 76, 67]
Students sorted by marks (ascending):
David: 67
Grace: 76
Alice: 78
Charlie: 85
Frank: 88
Eva: 90
Bob: 92
Students sorted by marks (descending):
Bob: 92
Eva: 90
Frank: 88
Charlie: 85
Alice: 78
Grace: 76
David: 67
```

Explanation:First, there was a vague order to rank student marks, therefore making the process of sorting a very simple and vague one.
This query was further limited by indicating the order of sorting and the result of the query became more precise.Finally, there were additional restrictions that were put in place, leading to a comprehensible, well-organized, and sufficient sorting program.

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.

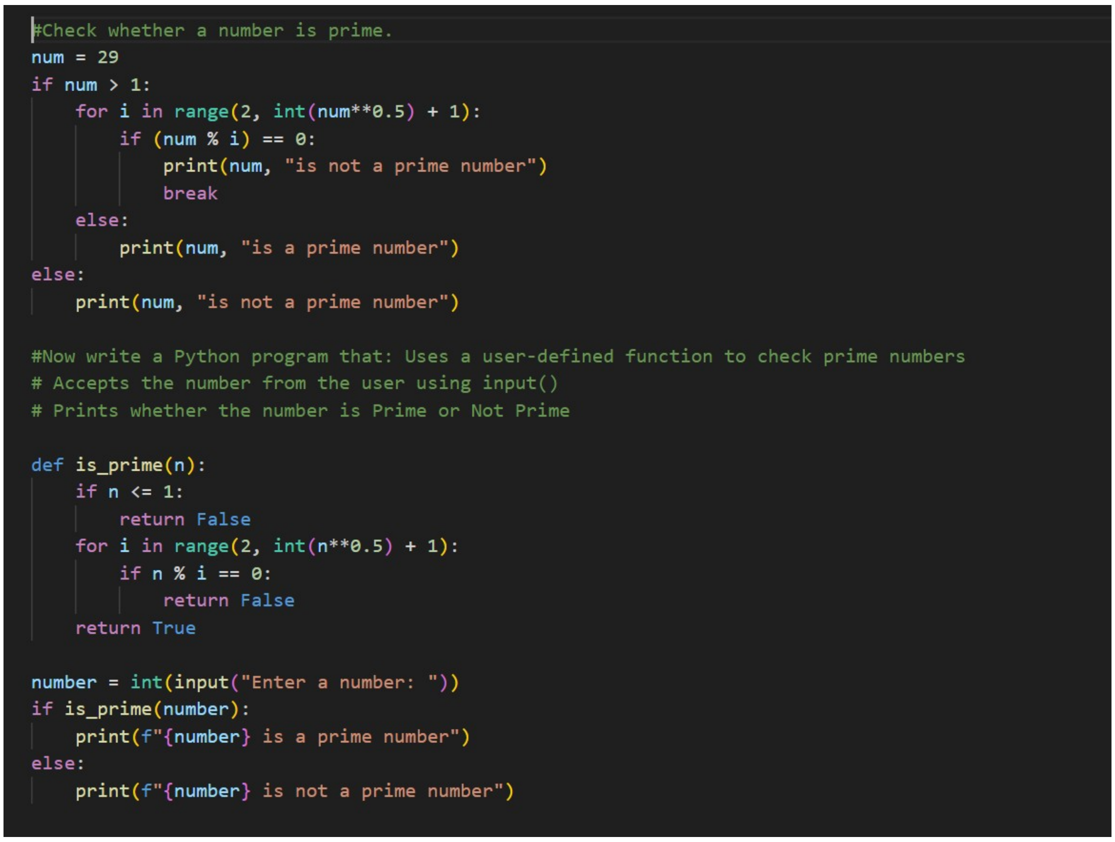
Expected Output:

The AI correctly identifies whether a given number is prime or not. Prime numbers return **“Prime”**, and non-prime numbers return **“Not Prime”**.

PROMPT

Check whether a number is prime.Now write a Python program that:-Uses a user-defined function to check prime numbers-Accepts the number from the user using input-Prints whether the number is Prime or Not Prime

Code:



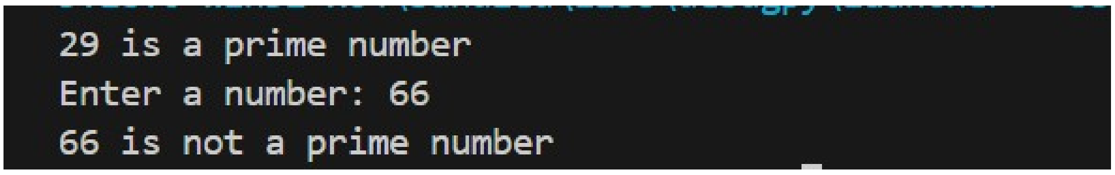
```python
#Check whether a number is prime.
num = 29
if num > 1:
    for i in range(2, int(num**0.5) + 1):
        if (num % i) == 0:
            print(num, "is not a prime number")
            break
    else:
        print(num, "is a prime number")
else:
    print(num, "is not a prime number")

#Now write a Python program that: Uses a user-defined function to check prime numbers
# Accepts the number from the user using input()
# Prints whether the number is Prime or Not Prime

def is_prime(n):
    if n <= 1:
        return False
    for i in range(2, int(n**0.5) + 1):
        if n % i == 0:
            return False
    return True

number = int(input("Enter a number: "))
if is_prime(number):
    print(f"{number} is a prime number")
else:
    print(f"{number} is not a prime number")
```

Output:



```
29 is a prime number
Enter a number: 66
66 is not a prime number
```

Explanation

To start with, we provided the AI example with inputs and outputs of prime numbers.We then requested it to code a program where the program reads in the input of the user.The AI had created a right and precise prime-checking program due to the examples.

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.

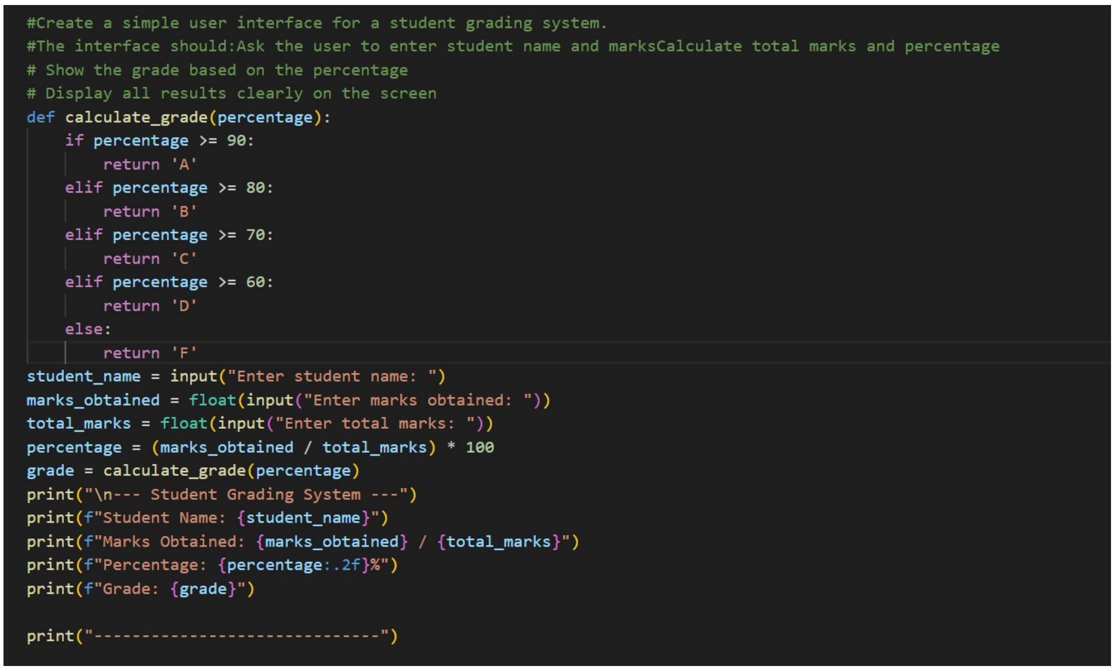
Expected output:

The screen allows the user to enter student details and marks.
The system calculates the total marks and percentage.
Finally, it shows the student's grade clearly.

Prompt:

Create a simple user interface for a student grading system.The interface should:#Ask the user to enter student name and marks# Calculate total marks and percentage# Show the grade based on the percentage# Display all results clearly on the screen

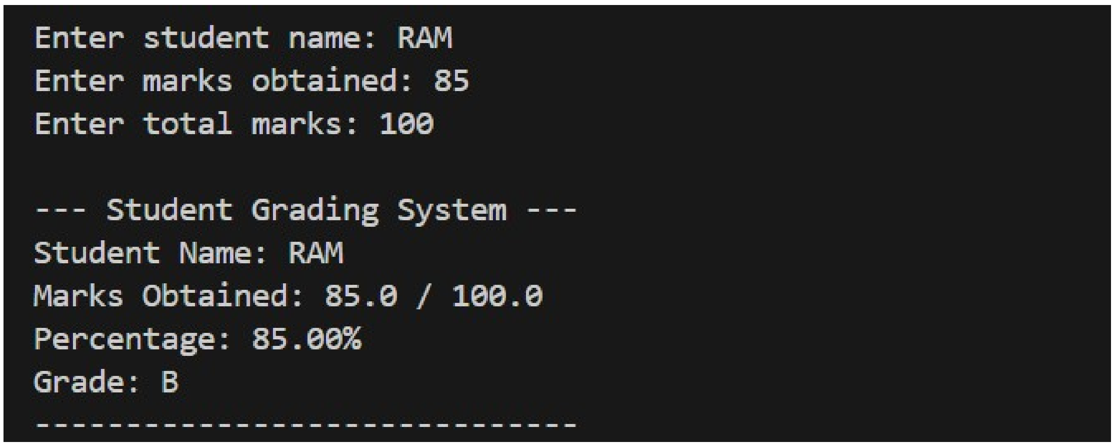
CODE:



```python
#Create a simple user interface for a student grading system.
#The interface should:Ask the user to enter student name and marksCalculate total marks and percentage
# Show the grade based on the percentage
# Display all results clearly on the screen
def calculate_grade(percentage):
    if percentage >= 90:
        return 'A'
    elif percentage >= 80:
        return 'B'
    elif percentage >= 70:
        return 'C'
    elif percentage >= 60:
        return 'D'
    else:
        return 'F'
student_name = input("Enter student name: ")
marks_obtained = float(input("Enter marks obtained: "))
total_marks = float(input("Enter total marks: "))
percentage = (marks_obtained / total_marks) * 100
grade = calculate_grade(percentage)
print("\n--- Student Grading System ---")
print(f"Student Name: {student_name}")
print(f"Marks Obtained: {marks_obtained} / {total_marks}")
print(f"Percentage: {percentage:.2f}%")
print(f"Grade: {grade}")

print("-----------------------------")
```

Output:


```
Enter student name: RAM
Enter marks obtained: 85
Enter total marks: 100

--- Student Grading System ---
Student Name: RAM
Marks Obtained: 85.0 / 100.0
Percentage: 85.00%
Grade: B
------------------------------
```

Explanation:

We gave the AI a clear prompt to design a student grading system interface.
The interface takes marks from the user and calculates total, percentage, and grade.
Clear prompts help the AI create a simple and useful user interface.

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.

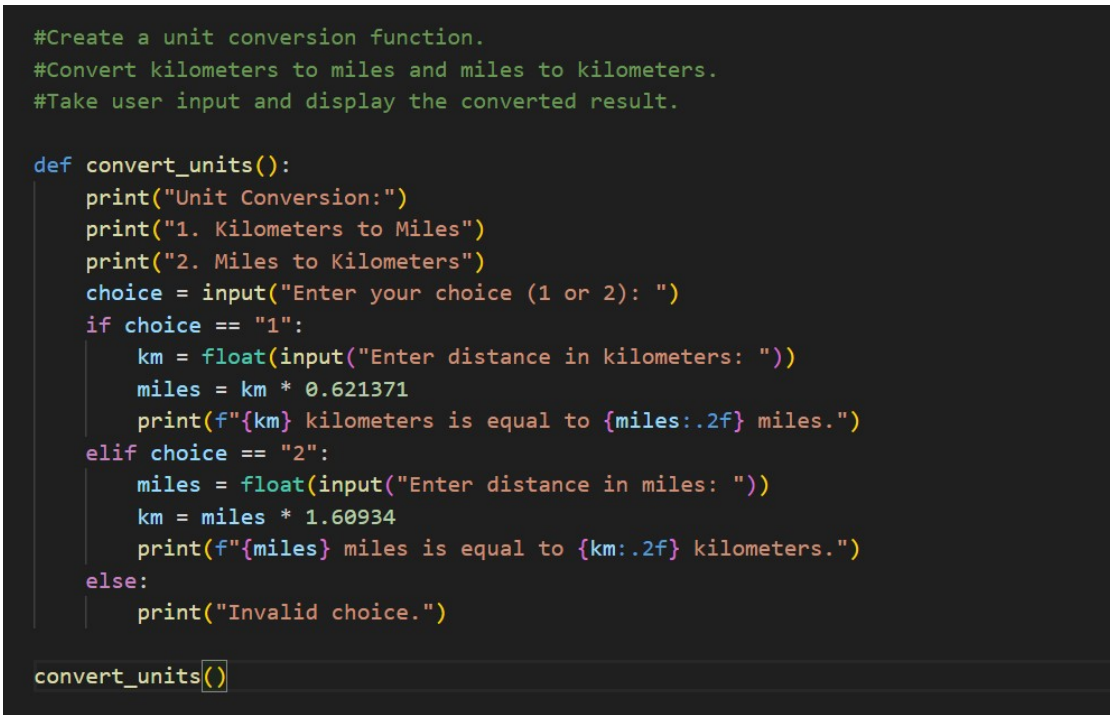
Expected output

The program correctly converts kilometers to miles and correctly converts miles to kilometers with accurate and clearly displayed to the user.

PROMPT:

Create a unit conversion function.Convert kilometers to miles and miles to kilometers.Take user input and display the converted result.

Code:

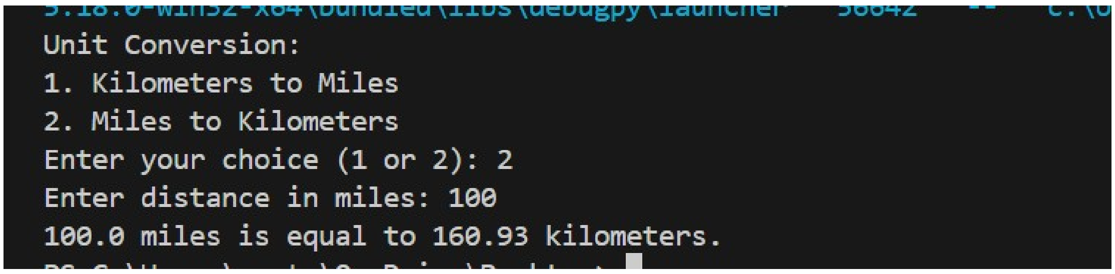


```python
#Create a unit conversion function.
#Convert kilometers to miles and miles to kilometers.
#Take user input and display the converted result.

def convert_units():
    print("Unit Conversion:")
    print("1. Kilometers to Miles")
    print("2. Miles to Kilometers")
    choice = input("Enter your choice (1 or 2): ")
    if choice == "1":
        km = float(input("Enter distance in kilometers: "))
        miles = km * 0.621371
        print(f"{km} kilometers is equal to {miles:.2f} miles.")
    elif choice == "2":
        miles = float(input("Enter distance in miles: "))
        km = miles * 1.60934
        print(f"{miles} miles is equal to {km:.2f} kilometers.")
    else:
        print("Invalid choice.")

convert_units()
```

Output:



```
3.18.0-win32-x64\bundled\libs\debugpy\launcher' '58642' '--' 'c:\U
Unit Conversion:
1. Kilometers to Miles
2. Miles to Kilometers
Enter your choice (1 or 2): 2
Enter distance in miles: 100
100.0 miles is equal to 160.93 kilometers.
```

Explanation:

The first thing that we asked the AI to do is to develop a unit conversion frequency.We then explained quite plainly that it had to change kilometers into miles and miles to kilometers.The instructions were clear and hence the AI produced a correct and accurate conversion program.