

AI-ASSISTED CODING ASSIGNMENT 3.2

SHRUTHI ADDAGUDI | 2303A52133 | BATCH 37

Task Description-1

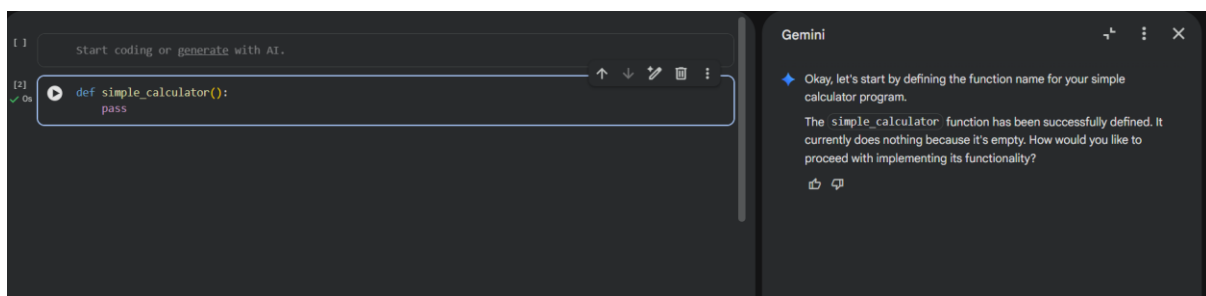
Progressive Prompting for Calculator Design: Ask the AI to design a simple calculator program by initially providing only the function name. Gradually enhance the prompt by adding comments and usage examples.

Prompt used:

design a simple calculator program by initially providing only the function name

code explanation:

Here, only the function name `simple_calculator()` is defined with a `pass` statement. Since no instructions or logic were provided, the AI generated an empty function. This shows that minimal prompts result in incomplete code.

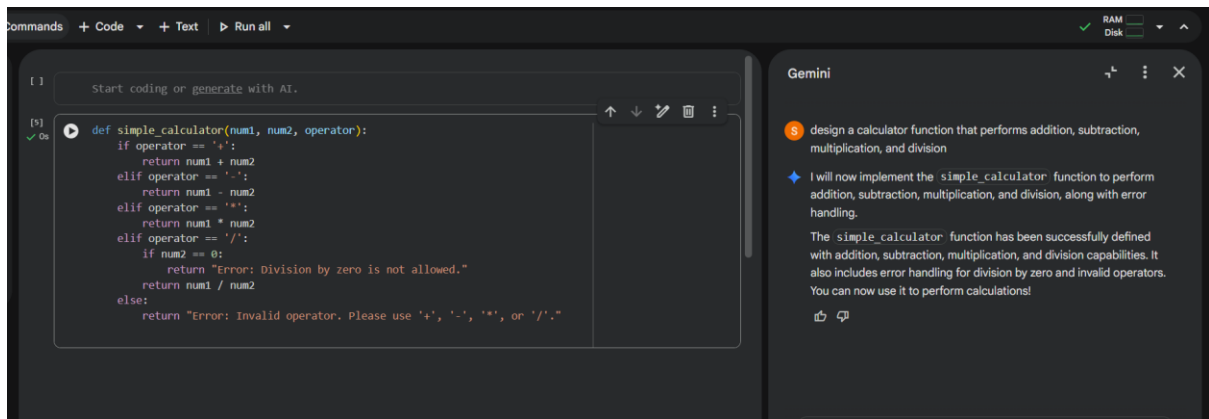


Prompt used:

design a calculator function that performs addition, subtraction, multiplication, and division

code explanation:

Here, the function accepts two numbers and an operator to perform arithmetic operations. Conditional statements are used to handle addition, subtraction, multiplication, and division. Error handling is added for division by zero and invalid operators due to clearer prompt instructions.

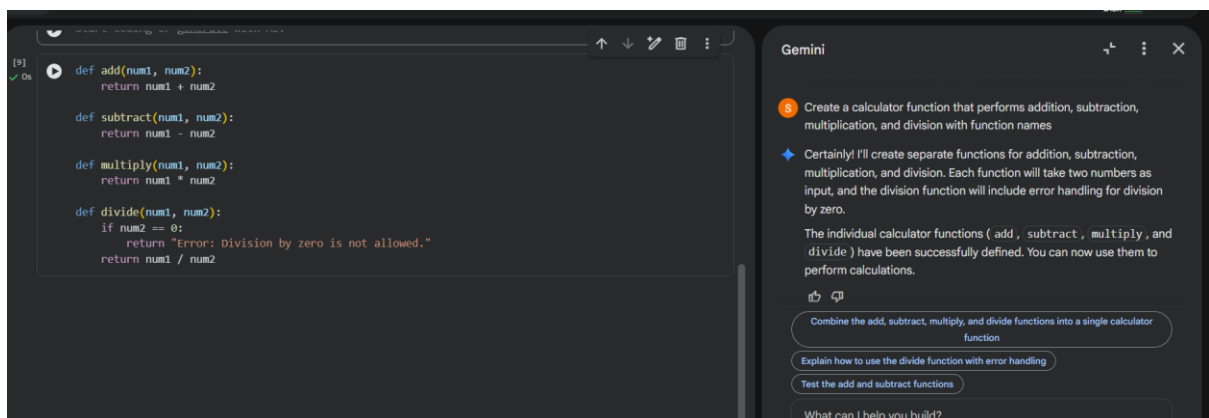


Prompt used:

Create a calculator function that performs addition, subtraction, multiplication, and division with function names

Code explanation:

In this stage, the calculator is divided into separate functions for add, subtract, multiply, and divide operations. This modular approach improves code readability and reusability. The AI generated cleaner and well-structured code based on refined prompting.



Prompt used:

Create a calculator function that performs addition, subtraction, multiplication, and division with function names and give examples for calculator performing for all functions

Output:

10 + 5 = 15

10 - 5 = 5

10*5 = 50

$$10/5 = 2$$

Code explanation:

Code gives us the execution and testing of all calculator functions using sample inputs. The outputs verify correct calculations and proper error handling. Including usage examples in the prompt helped the AI generate accurate test cases.

The screenshot shows a code editor on the left and a Gemini chat interface on the right. The code editor contains four examples of calculator functions: addition, subtraction, multiplication, and division. Each example includes a prompt, a function call, and the output. The division example also shows an error handling case for division by zero. The Gemini chat interface shows a prompt to create a calculator function, followed by a response that lists the functions and their outputs. Below the response are several prompts to refactor the code, explain error handling, and combine functions into a single class. The chat interface also includes a 'What can I help you build?' prompt and a 'Gemini 2.5 Flash' logo.

```
# Example for addition
print(f"10 + 5 = {add(10, 5)}")
10 + 5 = 15

# Example for subtraction
print(f"10 - 5 = {subtract(10, 5)}")
10 - 5 = 5

# Example for multiplication
print(f"10 * 5 = {multiply(10, 5)}")
10 * 5 = 50

# Example for division
print(f"10 / 5 = {divide(10, 5)}")

# Example for division by zero (error handling)
print(f"10 / 0 = {divide(10, 0)}")
10 / 5 = 2.0
10 / 0 = Error: Division by zero is not allowed.
```

Create a calculator function that performs addition, subtraction, multiplication, and division with function names and give examples for calculator performing for all functions

Certainly! Here are examples demonstrating how to use each of the individual calculator functions: `add`, `subtract`, `multiply`, and `divide`.

The examples have executed successfully, demonstrating each function:

- `add(10, 5)` returns 15.
- `subtract(10, 5)` returns 5.
- `multiply(10, 5)` returns 50.

Refactor the 'simple_calculator' function to return specific error messages

Explain the error handling in the 'divide' function

Combine the individual functions into a single calculator class

What can I help you build?

Gemini 2.5 Flash

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

prompt used :

Sort student marks

The screenshot shows a code editor on the left and a Gemini chat interface on the right. The code editor contains a sample list of student marks and two sorting functions: one for ascending order and one for descending order. The Gemini chat interface shows a prompt to sort student marks, followed by a response that explains the sorting process and displays the sorted marks in both ascending and descending order. Below the response are two prompts to calculate the average and find the median of the sorted marks.

```
# Sample list of student marks
student_marks = [85, 92, 78, 65, 95, 88, 70, 90]

# Sort the marks in ascending order
sorted_marks_ascending = sorted(student_marks)
print(f"Marks sorted in ascending order: {sorted_marks_ascending}")

# Sort the marks in descending order
sorted_marks_descending = sorted(student_marks, reverse=True)
print(f"Marks sorted in descending order: {sorted_marks_descending}")

Marks sorted in ascending order: [65, 70, 78, 85, 88, 90, 92, 95]
Marks sorted in descending order: [95, 92, 90, 88, 85, 78, 70, 65]
```

Sort student marks

Sure, I can help you sort student marks. To start, I'll show you how to sort a simple list of numerical marks in ascending order.

The student marks have been successfully sorted and displayed:

- Ascending Order:** [65, 70, 78, 85, 88, 90, 92, 95]
- Descending Order:** [95, 92, 90, 88, 85, 78, 70, 65]

This demonstrates how to use the `sorted()` function in Python to sort a list of numbers.

Calculate the average of the sorted student marks

Find the median of the sorted student marks

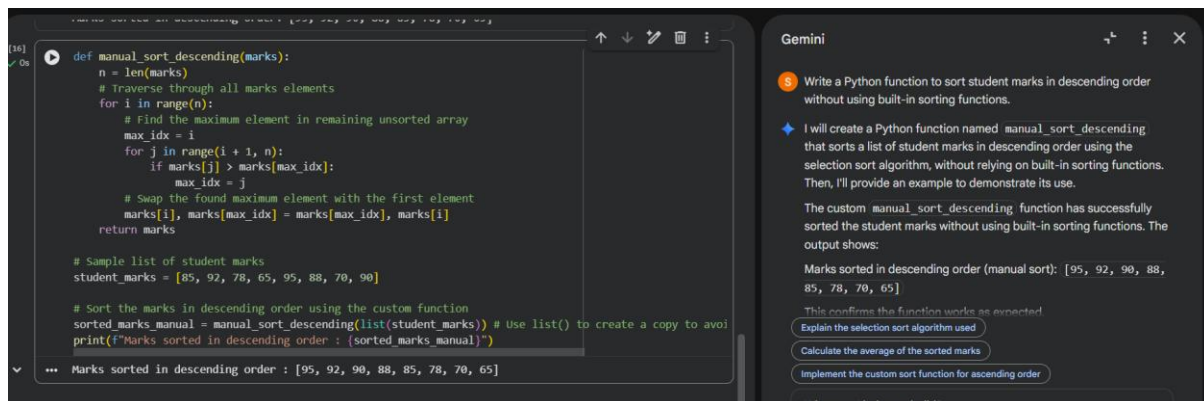
Code explanation:

The list `student_marks` stores the marks of different students. The `sorted()` function is used first to arrange the marks in ascending order and then in

descending order by setting `reverse=True`. This demonstrates how clear instructions help in sorting data accurately in different orders.

Prompt used:

Write a Python function to sort student marks in descending order without using built-in sorting functions.



The screenshot shows a code editor on the left and a Gemini chat interface on the right. The code editor contains a Python function `manual_sort_descending` that implements the Selection Sort algorithm. It takes a list of marks, finds the maximum element in the unsorted portion, swaps it with the first element, and repeats the process until the list is sorted in descending order. A sample list of marks is provided, and the output shows the sorted list.

```
def manual_sort_descending(marks):  
    n = len(marks)  
    # Traverse through all marks elements  
    for i in range(n):  
        # Find the maximum element in remaining unsorted array  
        max_idx = i  
        for j in range(i + 1, n):  
            if marks[j] > marks[max_idx]:  
                max_idx = j  
        # Swap the found maximum element with the first element  
        marks[i], marks[max_idx] = marks[max_idx], marks[i]  
    return marks  
  
# Sample list of student marks  
student_marks = [85, 92, 78, 65, 95, 88, 70, 90]  
  
# Sort the marks in descending order using the custom function  
sorted_marks_manual = manual_sort_descending(list(student_marks)) # Use list() to create a copy to avoid modifying the original list  
print(f"Marks sorted in descending order : {sorted_marks_manual}")
```

Marks sorted in descending order : [95, 92, 90, 88, 85, 78, 70, 65]

The Gemini chat interface shows the prompt: "Write a Python function to sort student marks in descending order without using built-in sorting functions." The response explains that the function `manual_sort_descending` is created, which sorts a list of student marks in descending order using the selection sort algorithm. It provides an example of the output: "Marks sorted in descending order (manual sort): [95, 92, 90, 88, 85, 78, 70, 65]".

Code explanation:

The function implements the **Selection Sort** algorithm to arrange student marks in descending order without using built-in sorting methods. It works by iterating through the list, finding the maximum element in the unsorted portion during each pass. This maximum element is then swapped into its correct position, and the process repeats until the entire list is sorted from largest to smallest.

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.

Prompt used:

The screenshot shows a code editor on the left and a Gemini chat window on the right. The code editor contains a Python function `is_prime(num)` that checks if a number is prime. It returns "Not Prime" for numbers less than 2 or if they are divisible by any number from 2 to the square root of the number. Otherwise, it returns "Prime". Below the function, there are example calls: `is_prime(2)`, `is_prime(4)`, `is_prime(1)`, and `is_prime(17)`. The output of these calls is shown below the code: `Input: 2 → Output: Prime`, `Input: 4 → Output: Not Prime`, `Input: 1 → Output: Not Prime`, and `Input: 17 → Output: Prime`.

The Gemini chat window on the right shows a prompt: "Write a function to check whether a number is prime. Examples: Input: 2 → Output: Prime Input: 4 → Output: Not Prime Input: 1 → Output: Not Prime Input: 17 → Output: Prime". The Gemini response is: "Certainly! I'll create a Python function `is_prime` that checks whether a given number is prime, and then demonstrate it with the examples you provided. The `is_prime` function has been successfully executed, and the results match the expected outputs:

- Input 2 correctly returns Prime.
- Input 4 correctly returns Not Prime.
- Input 1 correctly returns Not Prime.
- Input 17 correctly returns Prime.

"

Code explanation:

This function determines if a number is prime. It first checks if the number is less than 2, returning 'Not Prime' if true. Otherwise, it efficiently iterates from 2 up to the square root of the number. If any number in this range divides the input evenly, it's 'Not Prime'; otherwise, the number is declared 'Prime'.

Task Description-4

Guided UI Design for Student Grading System

Prompt used :

Create a user interface for a student grading system that calculates total marks, percentage, and grade based on user input.

The screenshot shows a code editor on the left and a Gemini chat window on the right. The code editor contains a Python program that takes student marks as input, validates them, and calculates the total sum. The program uses a `while True` loop to accept input until the user enters 'done'. It validates the input to ensure it is a number between 0 and 100. If the input is valid, it is added to a list. Once the user enters 'done', the program calculates the total sum of the marks and prints it. The output of the program is shown below the code: `Enter student mark (or 'done' to finish): 56`, `Enter student mark (or 'done' to finish): 89`, `Enter student mark (or 'done' to finish): done`, `Entered marks: [56.0, 89.0]`, `Total marks: 145.0`.

The Gemini chat window on the right shows a prompt: "Sure, here's a Python program that will allow you to input student marks one by one and then display the total sum of those marks. The program successfully accepted the student marks you entered (56.0 , 89.0) and correctly calculated their total, which is 145.0 .". The Gemini response is: "Sure, here's a Python program that will allow you to input student marks one by one and then display the total sum of those marks. The program successfully accepted the student marks you entered (56.0 , 89.0) and correctly calculated their total, which is 145.0 .".

Code explanation:

This program initializes an empty list and enters a loop to continuously accept student marks as input. It validates each mark, ensuring it's a number between 0 and 100, and adds valid entries to the list, while gracefully handling non-numeric input. The loop continues until the user types 'done', at which point it

exits. Finally, if any marks were entered, the program calculates and displays both the list of entered marks and their total sum.

Prompt used:

Create a Python program that accepts marks of 5 subjects from the user, calculates total marks, percentage, and displays grade based on percentage.

The screenshot shows a code editor with a Python program and its execution output. The program prompts the user to enter marks for 5 subjects, calculates the total marks, percentage, and displays the grade based on the percentage.

```
[19] marks = []
num_subjects = 5
max_marks_per_subject = 100

print(f'Please enter marks for {num_subjects} subjects (out of {max_marks_per_subject} each):')

for i in range(num_subjects):
    while True:
        try:
            mark_input = input(f'Enter mark for subject {i + 1}: ')
            mark = float(mark_input)
            if 0 <= mark <= max_marks_per_subject:
                marks.append(mark)
                break
            else:
                print(f'Invalid mark. Please enter a mark between 0 and {max_marks_per_subject}.')
        except ValueError:
            print('Invalid input. Please enter a numeric value.')

# Calculate total marks
total_marks = sum(marks)

# Calculate percentage
max_possible_total = num_subjects * max_marks_per_subject
percentage = (total_marks / max_possible_total) * 100

# Determine grade based on percentage
if percentage >= 90:
    grade = 'A'
elif percentage >= 80:
    grade = 'B+'
elif percentage >= 70:
    grade = 'B'
elif percentage >= 60:
    grade = 'C'
elif percentage >= 50:
    grade = 'D'
elif percentage >= 40:
    grade = 'E'
else:
    grade = 'F'

# Display results
print(f'\n--- Results ---')
print(f'Marks entered: {marks}')
print(f'Total Marks: {total_marks:.2f} / {max_possible_total}')
print(f'Percentage: {percentage:.2f}%')
print(f'Grade: {grade}')
```

--- Results ---
Marks entered: [56.0, 78.0, 98.0, 89.0, 88.0]
Total Marks: 409.00 / 500
Percentage: 81.80%
Grade: A

Code explanation:

This program prompts the user to enter marks for five subjects, validating each input to ensure it's a number between 0 and 100. It then calculates the total

marks and the overall percentage achieved by the student. Finally, based on a predefined grading scale, it assigns and displays a corresponding letter grade for the student's performance.

Task Description-5

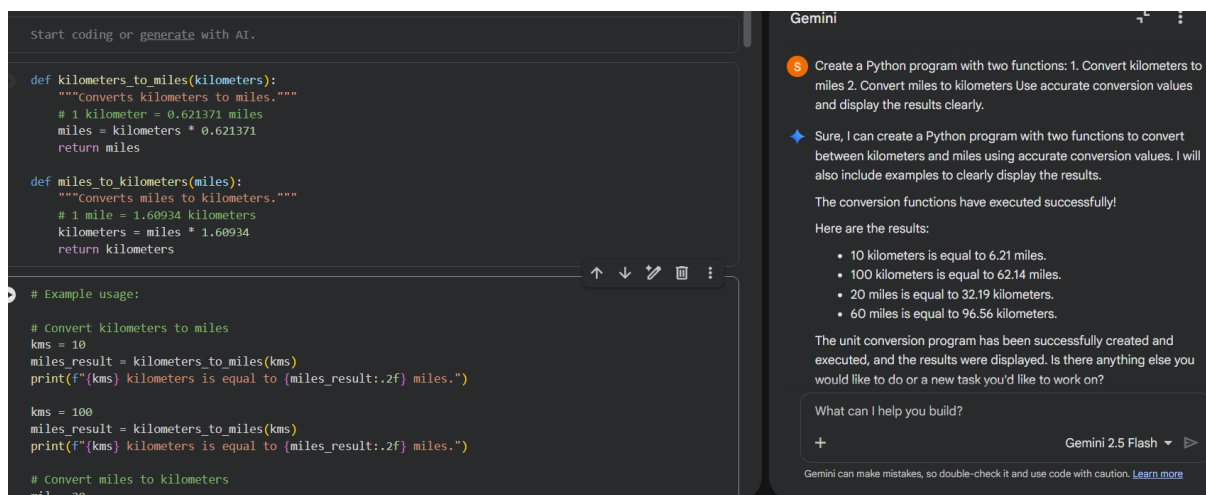
Analysing Prompt Specificity in Unit Conversion Functions: Improving a Unit Conversion Function (Kilometres to Miles and Miles to Kilometres) Using Clear Instructions.

Prompt used :

Create a Python program with two functions:

1. Convert kilometers to miles
2. Convert miles to kilometers

Use accurate conversion values and display the results clearly.



The screenshot shows a code editor on the left and the Gemini AI chat interface on the right. The code editor contains a Python program with two functions: `kilometers_to_miles` and `miles_to_kilometers`. The `kilometers_to_miles` function uses a conversion factor of 0.621371, and the `miles_to_kilometers` function uses a conversion factor of 1.60934. The code includes example usage for both functions. The Gemini chat interface shows a prompt asking to create a Python program with two functions to convert kilometers to miles and miles to kilometers, using accurate conversion values and displaying results clearly. The Gemini response confirms the creation of the program and provides the results of the conversions: 10 kilometers is equal to 6.21 miles, 100 kilometers is equal to 62.14 miles, 20 miles is equal to 32.19 kilometers, and 60 miles is equal to 96.56 kilometers.

```
def kilometers_to_miles(kilometers):  
    """Converts kilometers to miles."""  
    # 1 kilometer = 0.621371 miles  
    miles = kilometers * 0.621371  
    return miles  
  
def miles_to_kilometers(miles):  
    """Converts miles to kilometers."""  
    # 1 mile = 1.60934 kilometers  
    kilometers = miles * 1.60934  
    return kilometers  
  
# Example usage:  
  
# Convert kilometers to miles  
kms = 10  
miles_result = kilometers_to_miles(kms)  
print(f"{kms} kilometers is equal to {miles_result:.2f} miles.")  
  
kms = 100  
miles_result = kilometers_to_miles(kms)  
print(f"{kms} kilometers is equal to {miles_result:.2f} miles.")  
  
# Convert miles to kilometers  
mil = 20  
kilometers_result = miles_to_kilometers(mil)  
print(f"{mil} miles is equal to {kilometers_result:.2f} kilometers.")  
  
mil = 60  
kilometers_result = miles_to_kilometers(mil)  
print(f"{mil} miles is equal to {kilometers_result:.2f} kilometers.")
```

Gemini

Create a Python program with two functions: 1. Convert kilometers to miles 2. Convert miles to kilometers Use accurate conversion values and display the results clearly.

Sure, I can create a Python program with two functions to convert between kilometers and miles using accurate conversion values. I will also include examples to clearly display the results.

The conversion functions have executed successfully!

Here are the results:

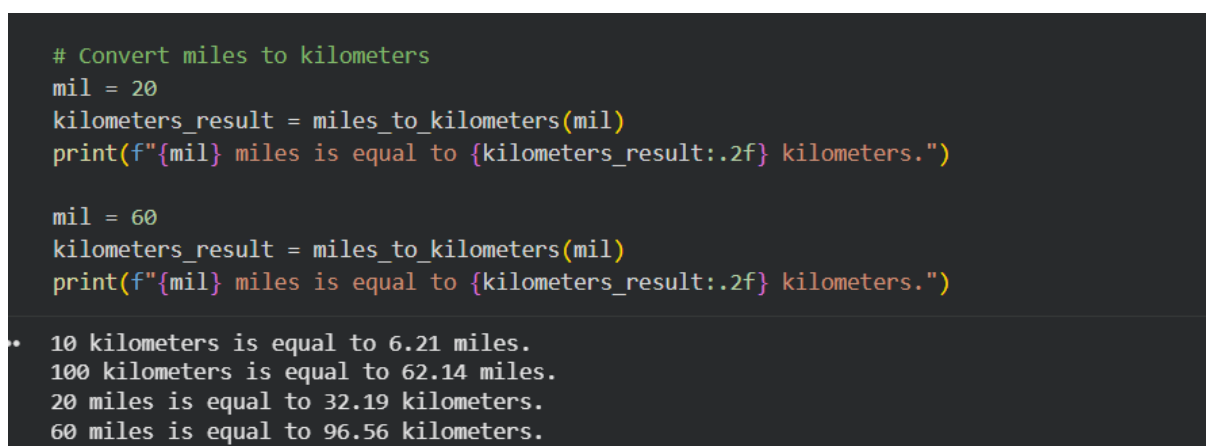
- 10 kilometers is equal to 6.21 miles.
- 100 kilometers is equal to 62.14 miles.
- 20 miles is equal to 32.19 kilometers.
- 60 miles is equal to 96.56 kilometers.

The unit conversion program has been successfully created and executed, and the results were displayed. Is there anything else you would like to do or a new task you'd like to work on?

What can I help you build?

Gemini 2.5 Flash

Gemini can make mistakes, so double-check it and use code with caution. [Learn more](#)



The screenshot shows a code editor with the final output of the unit conversion program. The code is the same as in the previous screenshot, but the output is displayed. The output shows the results of the conversions: 10 kilometers is equal to 6.21 miles, 100 kilometers is equal to 62.14 miles, 20 miles is equal to 32.19 kilometers, and 60 miles is equal to 96.56 kilometers.

```
# Convert miles to kilometers  
mil = 20  
kilometers_result = miles_to_kilometers(mil)  
print(f"{mil} miles is equal to {kilometers_result:.2f} kilometers.")  
  
mil = 60  
kilometers_result = miles_to_kilometers(mil)  
print(f"{mil} miles is equal to {kilometers_result:.2f} kilometers.")
```

• 10 kilometers is equal to 6.21 miles.
• 100 kilometers is equal to 62.14 miles.
• 20 miles is equal to 32.19 kilometers.
• 60 miles is equal to 96.56 kilometers.

Code explanation:

The Python program defines two functions: `kilometers_to_miles` and `miles_to_kilometers`. These functions facilitate unit conversion between kilometers and miles using precise conversion factors. The code then demonstrates their usage with various input values, clearly printing the converted results to the console, formatted to two decimal places for readability.