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:

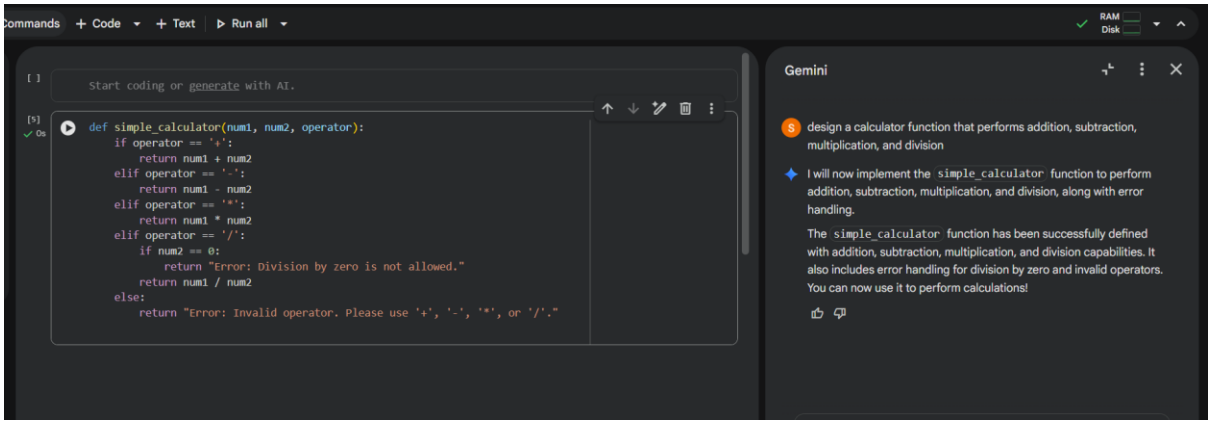
Here, only the function name simple_calculator() is defined with a pass statement. Since no instructions or logic were provided, the AI generated an empty function. This shows that minimal prompts result in incomplete code.

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.

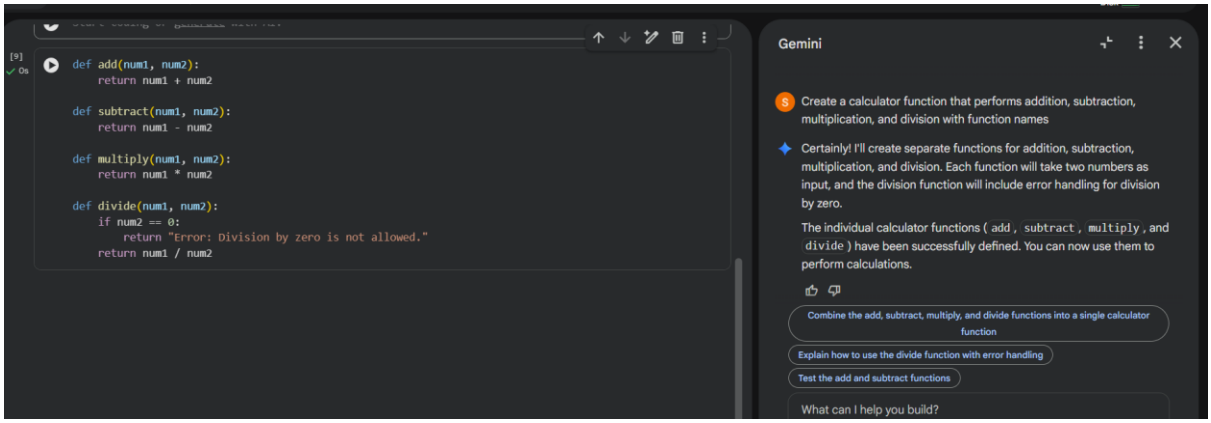


Prompt used:

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

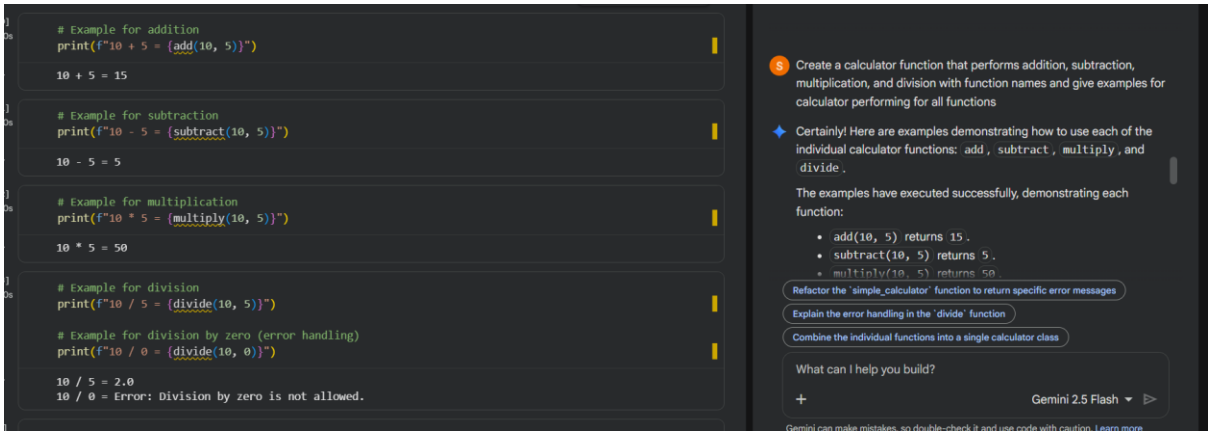
10 - 5 = 5

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.

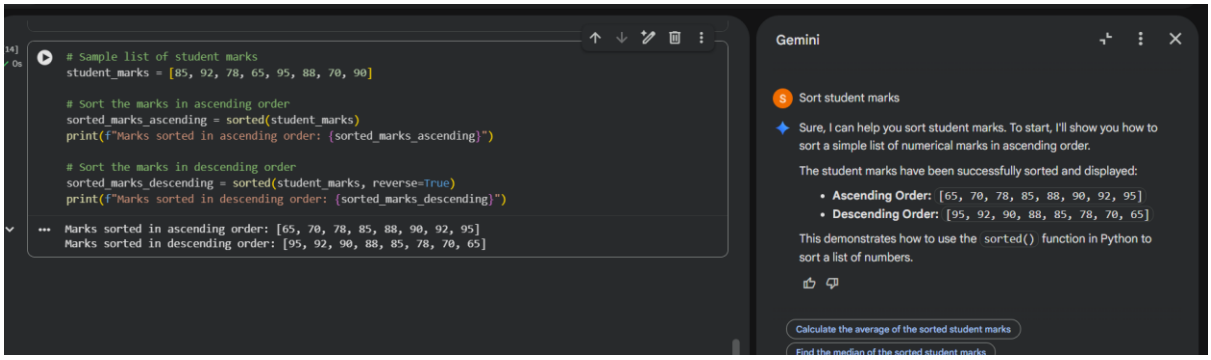


Task Description-2

• 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



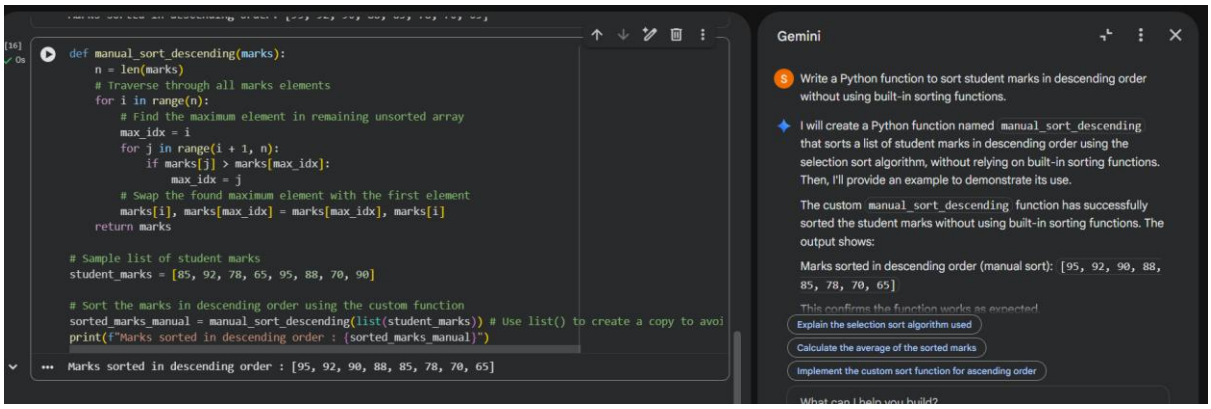
Code explanation:

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.



Code explanation:

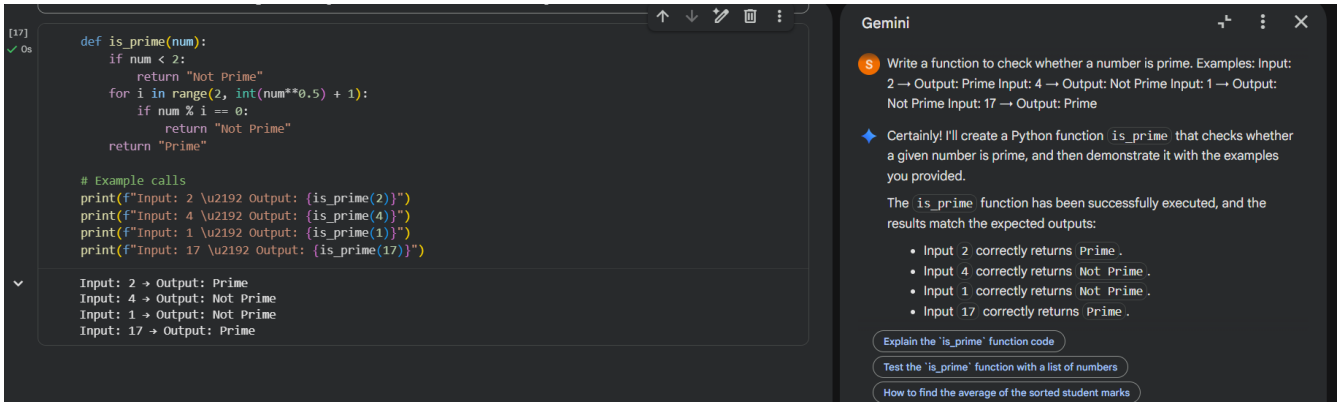
The function implements the **Selection Sort** algorithm to arrange student marks in descending order without using built-in sorting methods. It works by iterating through the list, finding the maximum element in the unsorted portion during each pass. This maximum element is then swapped into its correct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



Code explanation:

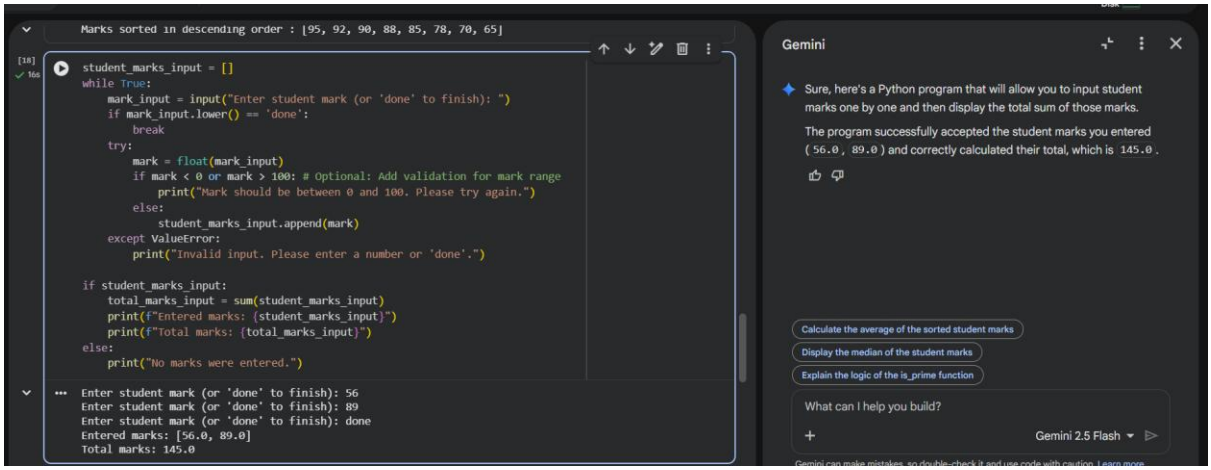
This function determines if a number is prime. It first checks if the number is less than 2, returning 'Not Prime' if true. Otherwise, it efficiently iterates from 2 up to the square root of the number. If any number in this range divides the input evenly, it's 'Not Prime'; otherwise, the number is declared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

Task Description-4

Guided UI Design for Student Grading System

Prompt used :

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



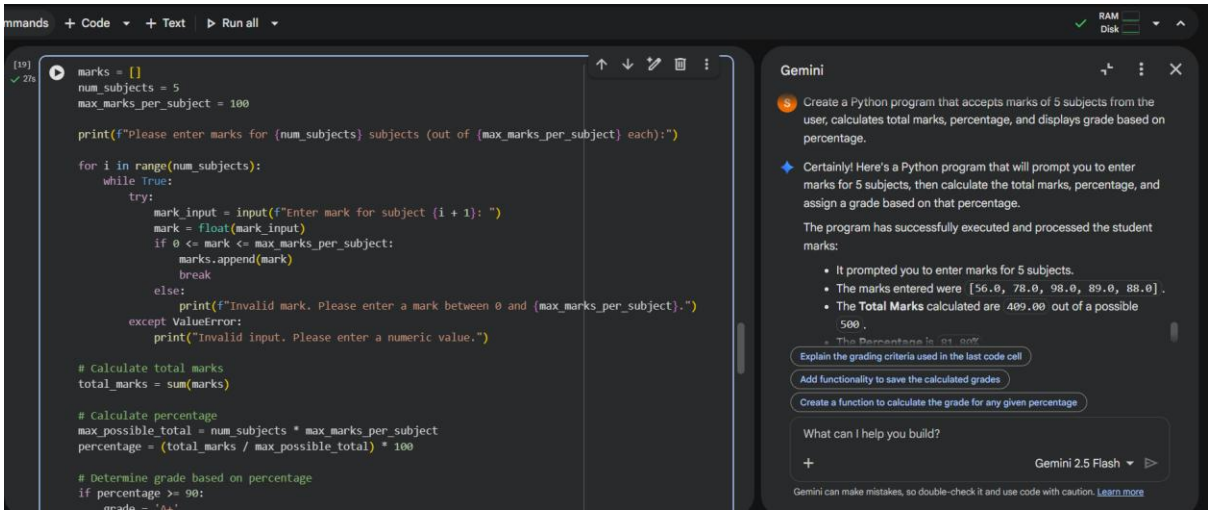
Code explanation:

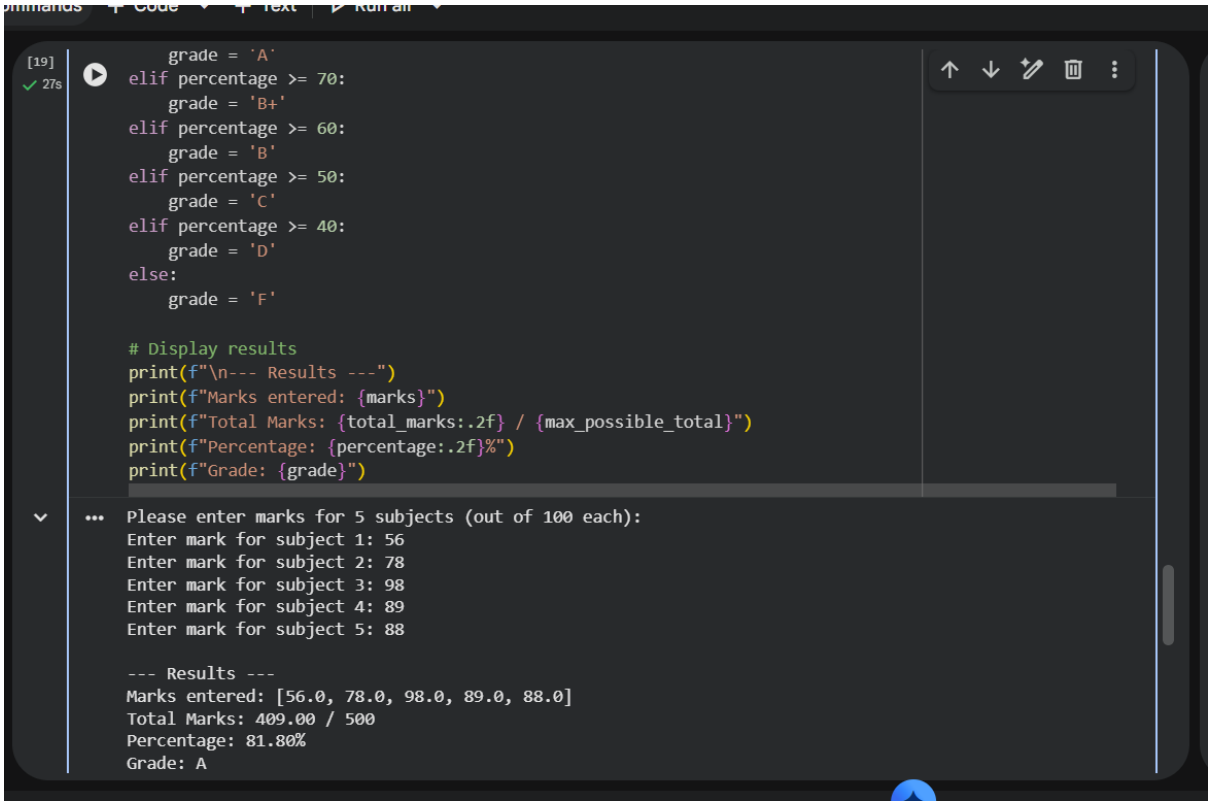
This program initializes an empty list and enters a loop to continuously accept student marks as input. It validates each mark, ensuring it's a number between 0 and 100, and adds valid entries to the list, while gracefully handling non-numeric input. The loop continues until the user types 'done', at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





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.

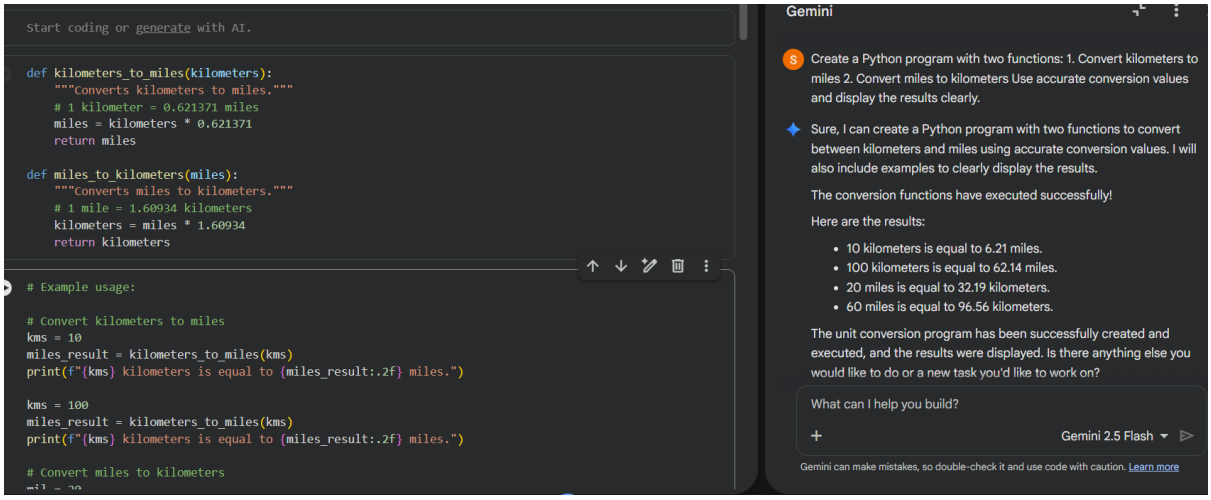
Prompt used :

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.




```
# 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.