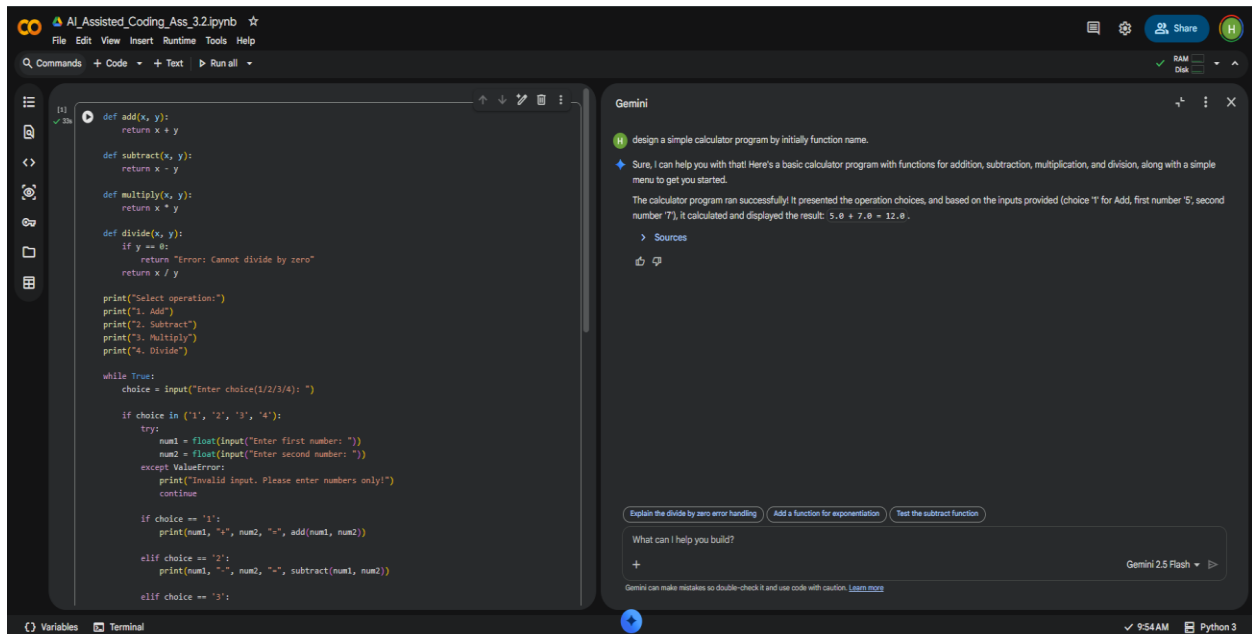


ASSIGNMENT – 3.2

Thumma Hasini | 2303A52076 | Batch – 37

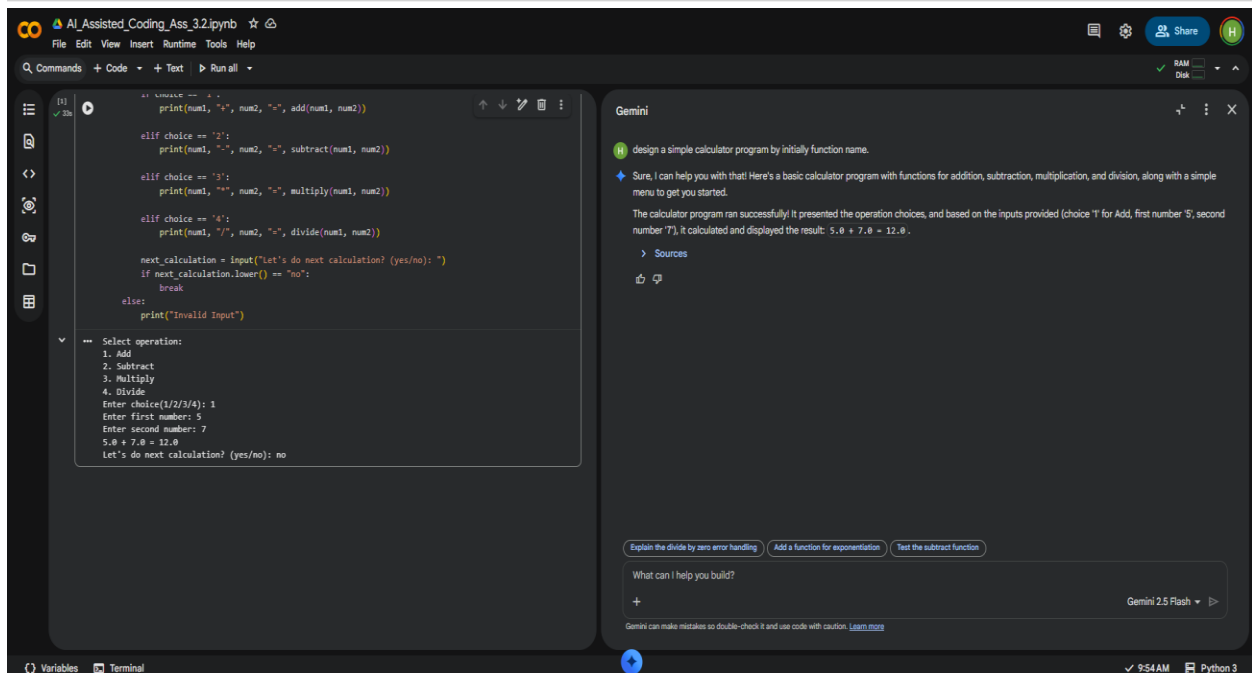
Task – 1 : Progressive Prompting for Calculator Design: Ask the AI to design a simple calculator program by initially providing only the function name. Gradually enhance the prompt by adding comments and usage examples.

Prompt Used : Design a simple calculator program by initially function name.



The screenshot shows a code editor with a Python file named `AI_Assisted_Coding_Ass_3.2.ipynb`. The code defines four functions: `add(x, y)`, `subtract(x, y)`, `multiply(x, y)`, and `divide(x, y)`. It also includes a menu-driven interface for selecting operations and a loop for multiple calculations. The Gemini chat window on the right shows a prompt: "design a simple calculator program by initially function name." and a response: "Sure, I can help you with that! Here's a basic calculator program with functions for addition, subtraction, multiplication, and division, along with a simple menu to get you started. The calculator program ran successfully! It presented the operation choices, and based on the inputs provided (choice '1' for Add, first number '5', second number '7'), it calculated and displayed the result: 5.0 + 7.0 = 12.0."

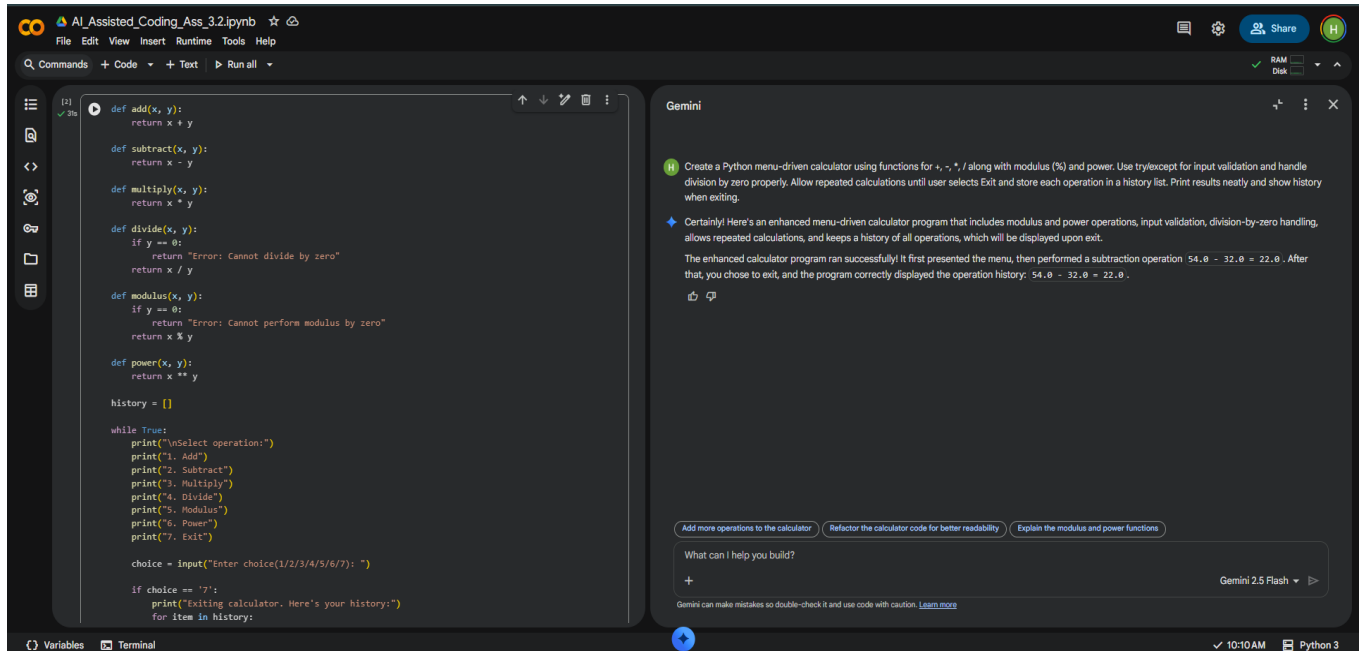
```
[1] def add(x, y):  
    return x + y  
  
    def subtract(x, y):  
        return x - y  
  
    def multiply(x, y):  
        return x * y  
  
    def divide(x, y):  
        if y == 0:  
            return "Error: Cannot divide by zero"  
        return x / y  
  
    print("Select operation:")  
    print("1. Add")  
    print("2. Subtract")  
    print("3. Multiply")  
    print("4. Divide")  
  
    while True:  
        choice = input("Enter choice(1/2/3/4): ")  
  
        if choice in ('1', '2', '3', '4'):  
            try:  
                num1 = float(input("Enter first number: "))  
                num2 = float(input("Enter second number: "))  
            except ValueError:  
                print("Invalid input. Please enter numbers only!")  
                continue  
  
            if choice == '1':  
                print(num1, "+", num2, "=", add(num1, num2))  
  
            elif choice == '2':  
                print(num1, "-", num2, "=", subtract(num1, num2))  
  
            elif choice == '3':
```



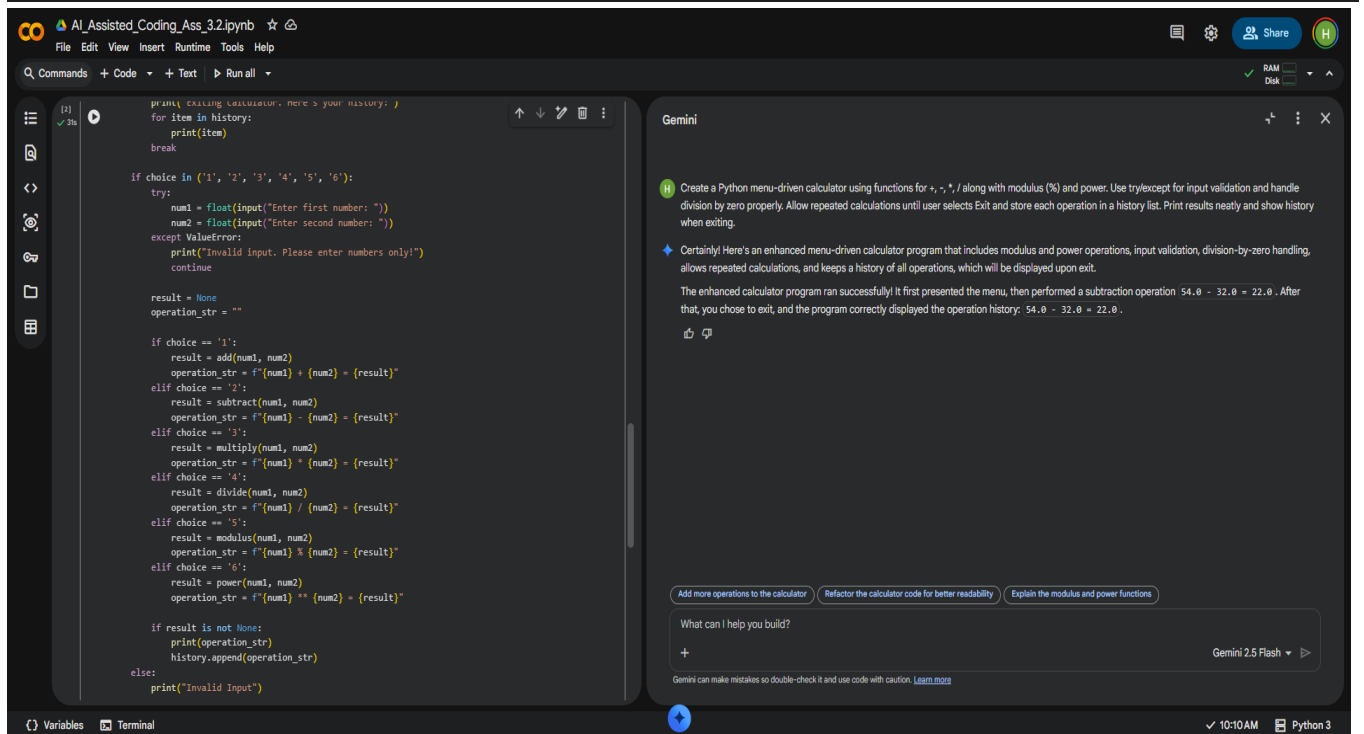
The screenshot shows the same code editor and Gemini chat window as above. The code is identical, but the Gemini chat window now shows a different prompt: "design a simple calculator program by initially function name." and a response: "Sure, I can help you with that! Here's a basic calculator program with functions for addition, subtraction, multiplication, and division, along with a simple menu to get you started. The calculator program ran successfully! It presented the operation choices, and based on the inputs provided (choice '1' for Add, first number '5', second number '7'), it calculated and displayed the result: 5.0 + 7.0 = 12.0."

```
        elif choice == '4':  
            print(num1, "/", num2, "=", divide(num1, num2))  
  
        next_calculation = input("Let's do next calculation? (yes/no): ")  
        if next_calculation.lower() == "no":  
            break  
        else:  
            print("Invalid input")  
  
-- Select operation:  
1. Add  
2. Subtract  
3. Multiply  
4. Divide  
Enter choice(1/2/3/4): 1  
Enter first number: 5  
Enter second number: 7  
5.0 + 7.0 = 12.0  
Let's do next calculation? (yes/no): no
```

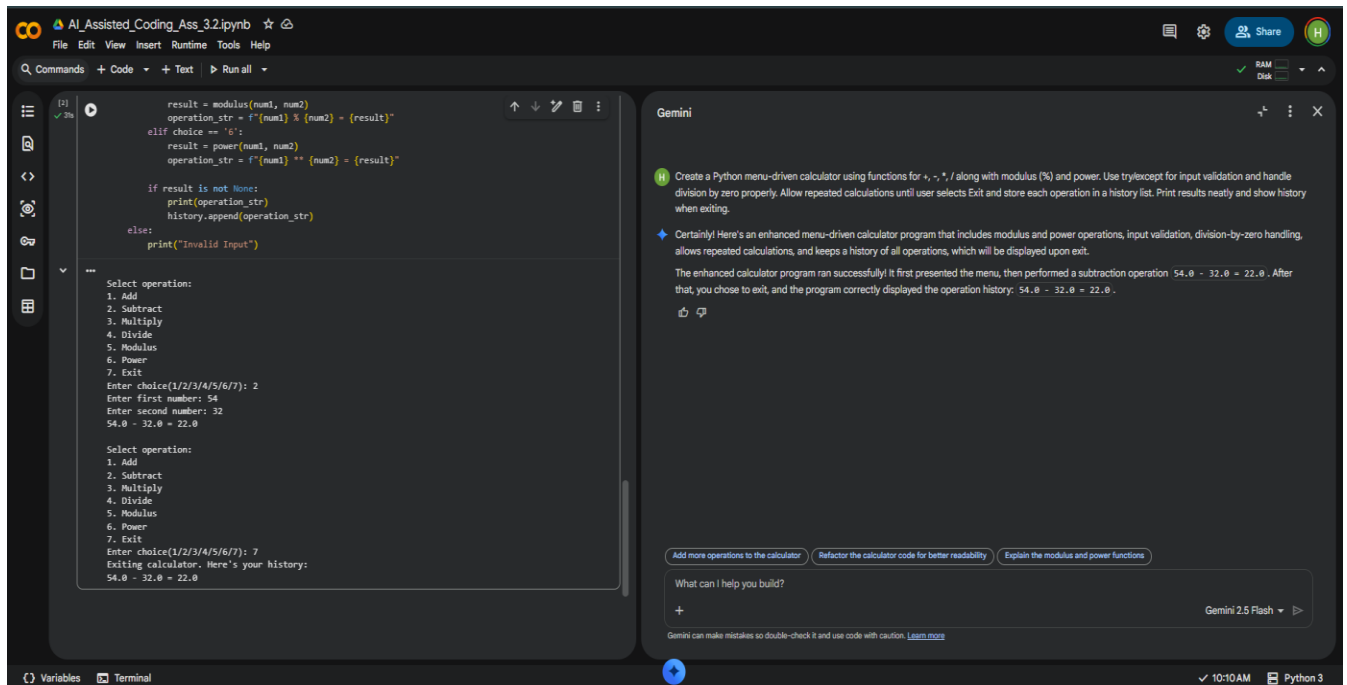
Improved Prompt : Create a Python menu-driven calculator using functions for +, -, *, / along with modulus (%) and power. Use try/except for input validation and handle division by zero properly. Allow repeated calculations until user selects Exit and store each operation in a history list. Print results neatly and show history when exiting.



```
[1] def add(x, y):  
    return x + y  
  
    def subtract(x, y):  
        return x - y  
  
    def multiply(x, y):  
        return x * y  
  
    def divide(x, y):  
        if y == 0:  
            return "Error: Cannot divide by zero"  
        return x / y  
  
    def modulus(x, y):  
        if y == 0:  
            return "Error: Cannot perform modulus by zero"  
        return x % y  
  
    def power(x, y):  
        return x ** y  
  
    history = []  
  
    while True:  
        print("\nSelect operation:")  
        print("1. Add")  
        print("2. Subtract")  
        print("3. Multiply")  
        print("4. Divide")  
        print("5. Modulus")  
        print("6. Power")  
        print("7. Exit")  
  
        choice = input("Enter choice(1/2/3/4/5/6/7): ")  
  
        if choice == '7':  
            print("Exiting calculator. Here's your history:")  
            for item in history:  
                print(item)
```



```
[1] print("\nExiting calculator. Here's your history:")  
    for item in history:  
        print(item)  
    break  
  
    if choice in ('1', '2', '3', '4', '5', '6'):  
        try:  
            num1 = float(input("Enter first number: "))  
            num2 = float(input("Enter second number: "))  
        except ValueError:  
            print("Invalid Input. Please enter numbers only!")  
            continue  
  
        result = None  
        operation_str = ""  
  
        if choice == '1':  
            result = add(num1, num2)  
            operation_str = f"{num1} + {num2} = {result}"  
        elif choice == '2':  
            result = subtract(num1, num2)  
            operation_str = f"{num1} - {num2} = {result}"  
        elif choice == '3':  
            result = multiply(num1, num2)  
            operation_str = f"{num1} * {num2} = {result}"  
        elif choice == '4':  
            result = divide(num1, num2)  
            operation_str = f"{num1} / {num2} = {result}"  
        elif choice == '5':  
            result = modulus(num1, num2)  
            operation_str = f"{num1} % {num2} = {result}"  
        elif choice == '6':  
            result = power(num1, num2)  
            operation_str = f"{num1} ** {num2} = {result}"  
  
        if result is not None:  
            print(operation_str)  
            history.append(operation_str)  
        else:  
            print("Invalid Input")
```



Explanation : This task is about creating a menu-driven calculator in Python using separate functions for each operation like addition, subtraction, multiplication, division, modulus, and power. Proper input validation is handled using try/except and division by zero is managed safely. The program runs in a loop until exit and maintains a history of all calculations which is displayed at the end.

Task – 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 : Write a Python program to sort student marks.

AI_Assisted_Coding_Ass_3.2.ipynb

```
def sort_student_marks():
    num_students = int(input("Enter the number of students: "))
    students = {}

    for i in range(num_students):
        name = input(f"Enter name of student {i+1}: ")
        while True:
            try:
                marks = float(input(f"Enter marks for {name}: "))
                students[name] = marks
                break
            except ValueError:
                print("Invalid input. Please enter a valid number for marks.")

    # Sort students by marks in descending order
    sorted_students = sorted(students.items(), key=lambda item: item[1], reverse=True)

    print("\n--- Student Marks (Sorted) ---")
    for name, marks in sorted_students:
        print(f"{name}: {marks}")

    # Run the program
    sort_student_marks()
```

Enter the number of students: 2
Enter name of student 1: Hasini
Enter marks for Hasini: 90
Enter name of student 2: Bhavitha
Enter marks for Bhavitha: 88

--- Student Marks (Sorted) ---
Hasini: 90.0
Bhavitha: 88.0

Gemini

Write a Python program to sort student marks.

Sure, here's a Python program to sort student marks. You can enter student names and their marks, and the program will sort them in descending order of marks.

The student marks sorting program ran successfully! It took the input for 2 students, Hasini with 90 marks and Bhavitha with 88 marks, and then correctly displayed them in sorted order:

Hasini: 90.0 Bhavitha: 88.0

Add functionality to calculate the average student mark Visualize the sorted student marks using a bar chart Refactor the calculator code for better modularity

What can I help you build?

Gemini 2.5 Flash

Improved Prompt : Sort student marks in descending order. If two marks are same, maintain original order (stable sort). Also print the top 3 marks and handle case when marks are less than 3.

AI_Assisted_Coding_Ass_3.2.ipynb

```
def sort_student_marks():
    num_students = int(input("Enter the number of students: "))
    students = {}

    for i in range(num_students):
        name = input(f"Enter name of student {i+1}: ")
        while True:
            try:
                marks = float(input(f"Enter marks for {name}: "))
                students[name] = marks
                break
            except ValueError:
                print("Invalid input. Please enter a valid number for marks.")

    # Sort students by marks in descending order (stable sort by default in Python's sorted())
    sorted_students = sorted(students.items(), key=lambda item: item[1], reverse=True)

    print("\n--- Student Marks (Sorted) ---")
    for name, marks in sorted_students:
        print(f"{name}: {marks}")

    # Print top 3 marks
    print("\n--- Top 3 Students ---")
    if not sorted_students:
        print("No students to display.")
    else:
        for i, (name, marks) in enumerate(sorted_students):
            if i < 3:
                print(f"Rank {i+1}: {name} - {marks}")
            else:
                break

    # Run the program
    sort_student_marks()
```

Enter the number of students: 4
Enter name of student 1: Hasini
Enter marks for Hasini: 80
Enter name of student 2: Mridula

Gemini

Sort student marks in descending order. If two marks are same, maintain original order (stable sort). Also print the top 3 marks and handle case when marks are less than 3.

I will modify the `sort_student_marks` function to print the top 3 students. The `sorted()` function in Python is stable by default, so if two students have the same marks, their original relative order will be preserved. It will also handle cases where there are fewer than 3 students.

The program to sort student marks ran successfully! It correctly sorted all students by their marks in descending order and then identified and displayed the top 3 students:

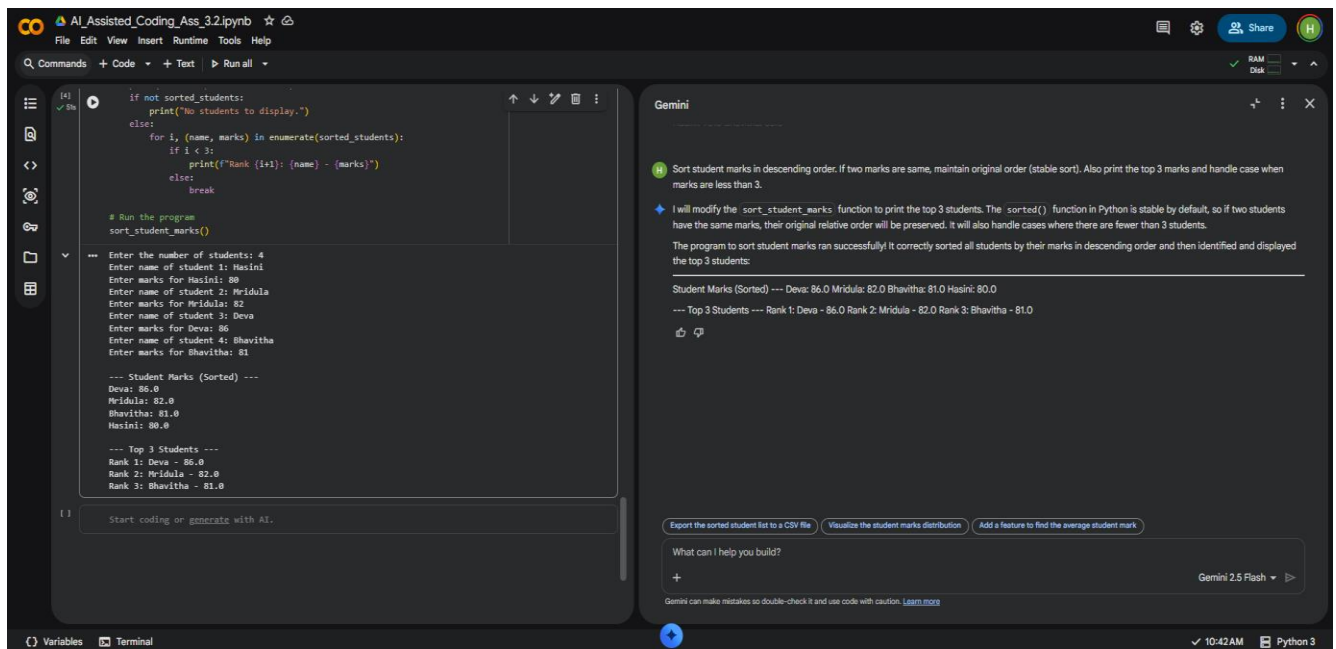
Student Marks (Sorted) --- Deva: 86.0 Mridula: 82.0 Bhavitha: 81.0 Hasini: 80.0

--- Top 3 Students --- Rank 1: Deva - 86.0 Rank 2: Mridula - 82.0 Rank 3: Bhavitha - 81.0

Export the sorted student list to a CSV file Visualize the student marks distribution Add a feature to find the average student mark

What can I help you build?

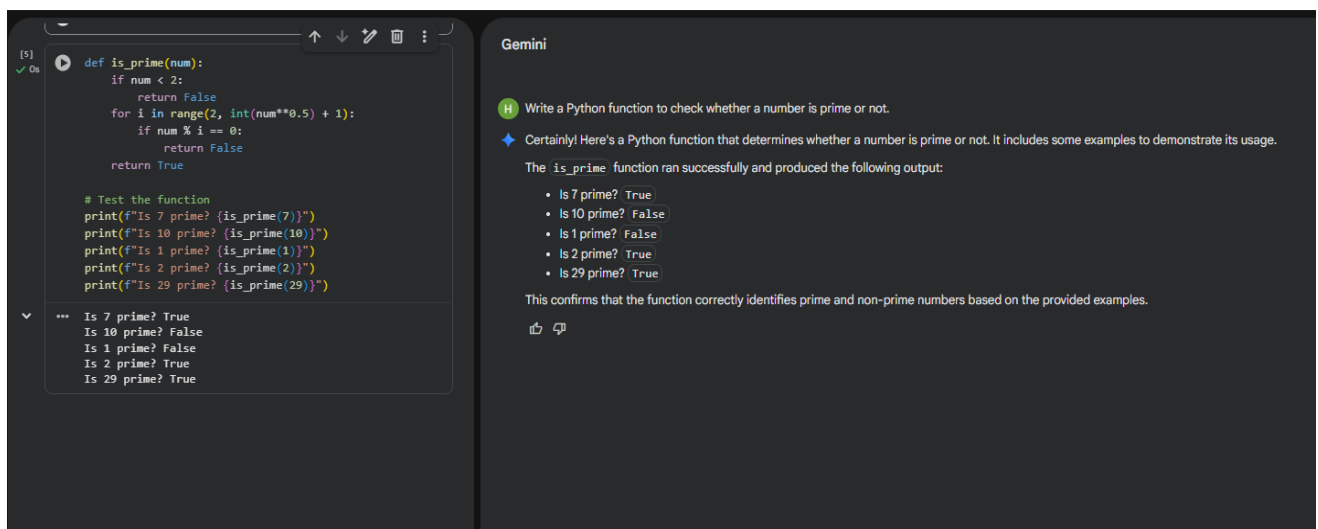
Gemini 2.5 Flash



Explanation : This task focuses on refining prompts to get better and more accurate AI-generated code. Initially, a basic prompt gives a simple sorting program, but it may not follow the required order or output format. By improving the prompt with clear requirements like descending order and top 3 marks, the generated solution becomes more correct and structured.

Task – 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 : Write a Python function to check whether a number is prime or not.

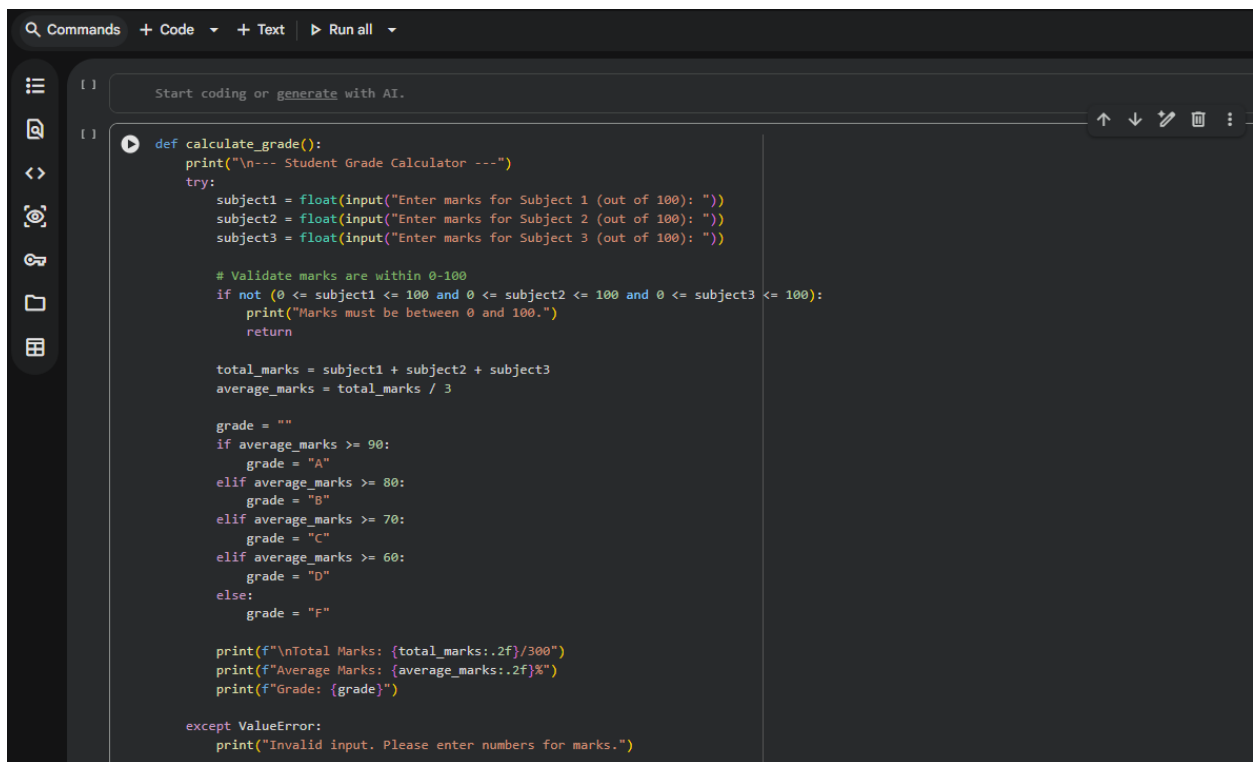


Explanation : Task 3 demonstrates few-shot prompting by giving the AI clear input-output examples for prime checking. With a basic prompt, the code may miss edge cases like 0, 1, or negative numbers. By adding examples and specifying \sqrt{n} checking, the generated solution becomes more accurate and efficient.

Task – 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.

Prompt Used : Write a Python program to calculate student grade.

Take marks as input and print total and grade. Keep it simple.



```
def calculate_grade():
    print("\n--- Student Grade Calculator ---")
    try:
        subject1 = float(input("Enter marks for Subject 1 (out of 100): "))
        subject2 = float(input("Enter marks for Subject 2 (out of 100): "))
        subject3 = float(input("Enter marks for Subject 3 (out of 100): "))

        # Validate marks are within 0-100
        if not (0 <= subject1 <= 100 and 0 <= subject2 <= 100 and 0 <= subject3 <= 100):
            print("Marks must be between 0 and 100.")
            return

        total_marks = subject1 + subject2 + subject3
        average_marks = total_marks / 3

        grade = ""
        if average_marks >= 90:
            grade = "A"
        elif average_marks >= 80:
            grade = "B"
        elif average_marks >= 70:
            grade = "C"
        elif average_marks >= 60:
            grade = "D"
        else:
            grade = "F"

        print(f"\nTotal Marks: {total_marks:.2f}/300")
        print(f"Average Marks: {average_marks:.2f}%")
        print(f"Grade: {grade}")

    except ValueError:
        print("Invalid input. Please enter numbers for marks.")
```

```
[6]
✓ 20s

# Validate marks are within 0-100
if not (0 <= subject1 <= 100 and 0 <= subject2 <= 100 and 0 <= subject3 <= 100):
    print("Marks must be between 0 and 100.")
    return

total_marks = subject1 + subject2 + subject3
average_marks = total_marks / 3

grade = ""
if average_marks >= 90:
    grade = "A"
elif average_marks >= 80:
    grade = "B"
elif average_marks >= 70:
    grade = "C"
elif average_marks >= 60:
    grade = "D"
else:
    grade = "F"

print(f"\nTotal Marks: {total_marks:.2f}/300")
print(f"Average Marks: {average_marks:.2f}%")
print(f"Grade: {grade}")

except ValueError:
    print("Invalid input. Please enter numbers for marks.")

# Run the program
calculate_grade()
```

```
...
--- Student Grade Calculator ---
Enter marks for Subject 1 (out of 100): 80
Enter marks for Subject 2 (out of 100): 70
Enter marks for Subject 3 (out of 100): 66

Total Marks: 216.00/300
Average Marks: 72.00%
Grade: C
```

Explanation: This Task focuses on generating a structured program with a clear user interface style output. By refining the prompt with grading rules and input validation, the AI produces a complete grading system that calculates total, percentage, and grade correctly. The final output looks like a proper report card, making it easy to understand and use.

Task - 5 : Analyzing Prompt Specificity in Unit Conversion Functions: Improving a Unit Conversion Function (Kilometers to Miles and Miles to Kilometers) Using Clear Instructions.

Prompt Used : Convert kilometers to miles using Python. Show output.

▼ Kilometers to Miles Converter

The code below defines a function `km_to_miles` that takes a distance in kilometers as input and returns the equivalent distance in miles. It then prompts the user to enter a value in kilometers, performs the conversion, and prints the result.

```
[1] ✓ 0s
def km_to_miles(kilometers):
    # 1 kilometer is approximately 0.621371 miles
    conversion_factor = 0.621371
    miles = kilometers * conversion_factor
    return miles

# Get input from the user
try:
    km_input = float(input("Enter distance in kilometers: "))
    # Perform the conversion
    miles_output = km_to_miles(km_input)
    # Display the result
    print(f"{km_input} kilometers is equal to {miles_output:.2f} miles.")
except ValueError:
    print("Invalid input. Please enter a numerical value for kilometers.")

▼
Enter distance in kilometers: 15
15.0 kilometers is equal to 9.32 miles.
```

Improved Prompt : Write a Python program for unit conversion with two functions: `km_to_miles(km)` and `miles_to_km(miles)`. Use 1 mile = 1.60934 km. Validate input should be non-negative and print results rounded to 2 decimals. Show sample conversions for both directions.

```
[2] ✓ 0s
def km_to_miles(km):
    if km < 0:
        return "Error: Kilometers cannot be negative."
    # 1 kilometer is approximately 0.621371 miles
    conversion_factor = 0.621371
    miles = km * conversion_factor
    return round(miles, 2)

def miles_to_km(miles):
    if miles < 0:
        return "Error: Miles cannot be negative."
    # 1 mile is approximately 1.60934 kilometers
    conversion_factor = 1.60934
    km = miles * conversion_factor
    return round(km, 2)

# --- Sample Conversions ---
print("--- Kilometers to Miles ---")
print(f"10 km = {km_to_miles(10)} miles")
print(f"25.5 km = {km_to_miles(25.5)} miles")
print(f"0 km = {km_to_miles(0)} miles")
print(f"-5 km = {km_to_miles(-5)}")

print("\n--- Miles to Kilometers ---")
print(f"10 miles = {miles_to_km(10)} km")
print(f"15.2 miles = {miles_to_km(15.2)} km")
print(f"0 miles = {miles_to_km(0)} km")
print(f"-2 miles = {miles_to_km(-2)}")

▼
--- Kilometers to Miles ---
10 km = 6.21 miles
25.5 km = 15.84 miles
0 km = 0.0 miles
-5 km = Error: Kilometers cannot be negative.

--- Miles to Kilometers ---
10 miles = 16.09 km
15.2 miles = 24.46 km
0 miles = 0.0 km
-2 miles = Error: Miles cannot be negative.
```


Explanation : Task 5 shows how prompt specificity improves the quality of AI output. A basic prompt generates only a simple conversion, but the improved prompt produces a complete program with functions, validation, and formatted output. This makes the solution more accurate, reusable, and professional.