

# AI ASSISTED CODING

SHASHANK

2303A510i3

BATCH – 03

20 – 01 – 2026

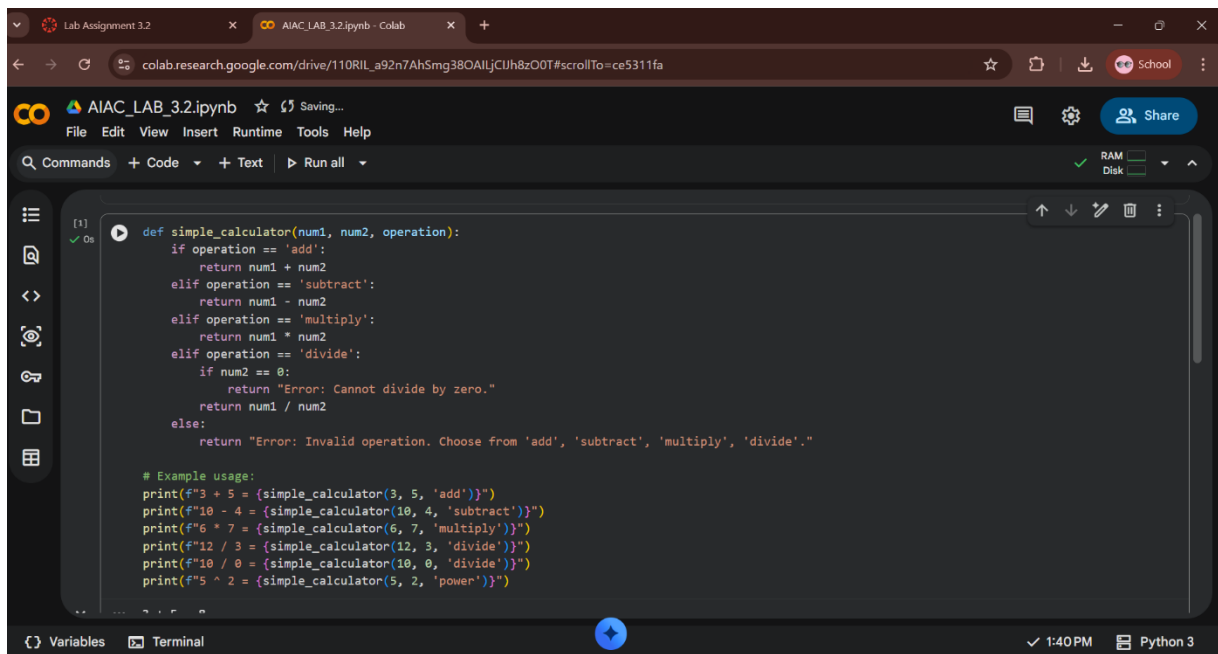
## ASSIGNMENT – 3.2

**LAB – 03 :** Prompt Engineering – Improving Prompts and Context Management.

**TASK - 01 :** Progressive Prompting for Calculator Design.

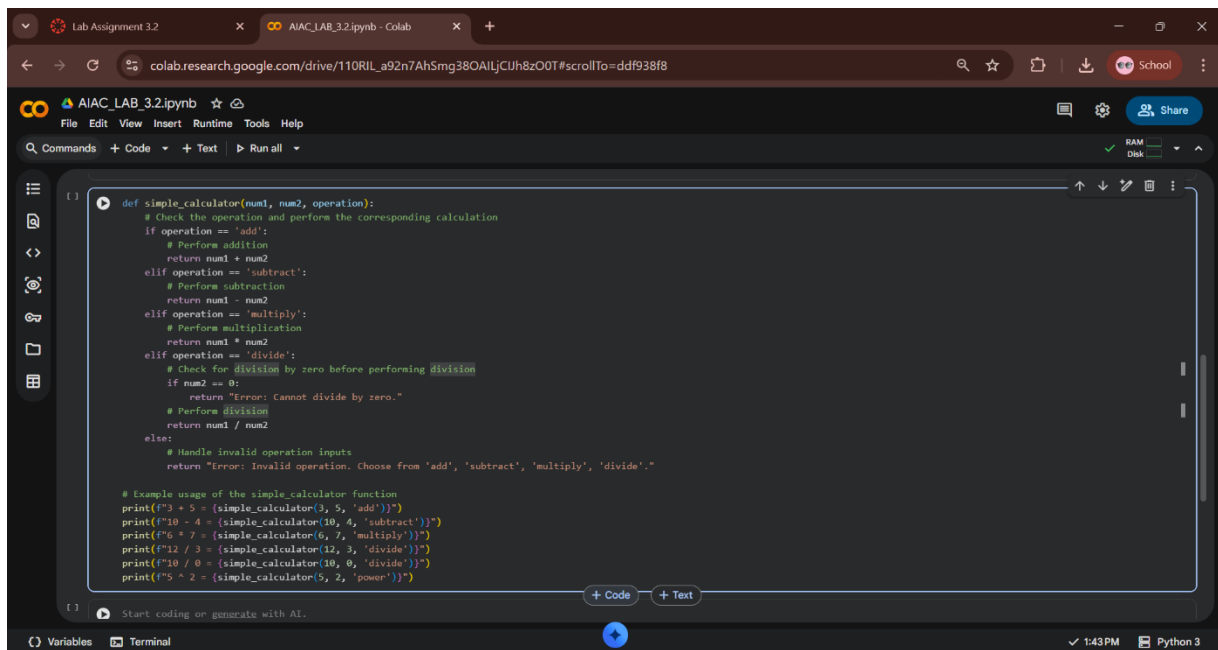
**Prompt – 01:** Create a Python function named simple\_calculator.

**Code :**

A screenshot of a Google Colab notebook titled 'AIAC\_LAB\_3.2.ipynb'. The notebook is open in a web browser, showing the code editor and a left sidebar with icons for file explorer, search, and other tools. The code defines a function 'simple\_calculator' that takes three arguments: 'num1', 'num2', and 'operation'. It uses if-elif-else statements to perform addition, subtraction, multiplication, and division. It also includes error handling for division by zero and invalid operations. Below the function definition, there is a comment '# Example usage:' followed by several print statements that call the function with different inputs and operations. The status bar at the bottom shows 'Variables', 'Terminal', '1:40 PM', and 'Python 3'.

**Prompt – 02 :** Create a Python function named simple\_calculator that performs basic arithmetic operations (addition, subtraction, multiplication, division). Add comments explaining each step.

## Code:

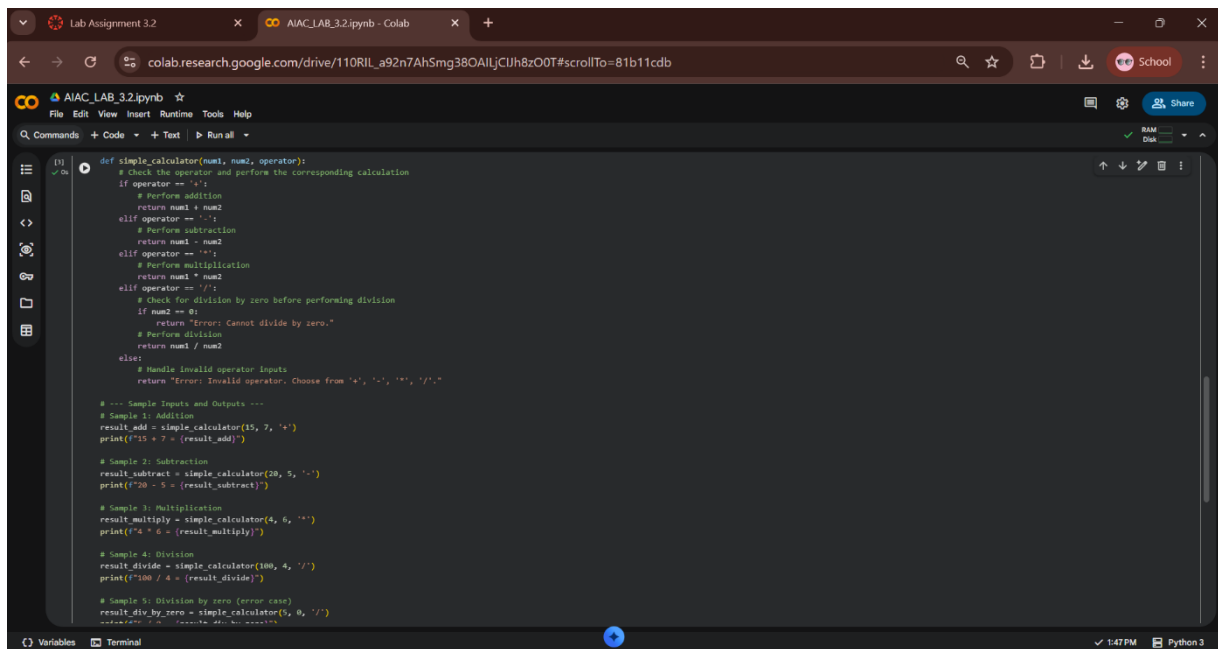


```
def simple_calculator(num1, num2, operation):
    # Check the operation and perform the corresponding calculation
    if operation == 'add':
        # Perform addition
        return num1 + num2
    elif operation == 'subtract':
        # Perform subtraction
        return num1 - num2
    elif operation == 'multiply':
        # Perform multiplication
        return num1 * num2
    elif operation == 'divide':
        # Check for division by zero before performing division
        if num2 == 0:
            return "Error: Cannot divide by zero."
        # Perform division
        return num1 / num2
    else:
        # Handle invalid operation inputs
        return "Error: Invalid operation. Choose from 'add', 'subtract', 'multiply', 'divide'."

# Example usage of the simple_calculator function
print(f"3 + 5 = {simple_calculator(3, 5, 'add')}")
print(f"10 - 4 = {simple_calculator(10, 4, 'subtract')}")
print(f"6 * 7 = {simple_calculator(6, 7, 'multiply')}")
print(f"12 / 3 = {simple_calculator(12, 3, 'divide')}")
print(f"10 / 0 = {simple_calculator(10, 0, 'divide')}")
print(f"5 ^ 2 = {simple_calculator(5, 2, 'power')}")
```

**Prompt – 03 :** Create a Python function named `simple_calculator` that takes two numbers and an operator (+, -, \*, /) as input and returns the result. Include comments and provide at least two sample inputs with outputs.

## Code :



```
def simple_calculator(num1, num2, operator):
    # Check the operator and perform the corresponding calculation
    if operator == '+':
        # Perform addition
        return num1 + num2
    elif operator == '-':
        # Perform subtraction
        return num1 - num2
    elif operator == '*':
        # Perform multiplication
        return num1 * num2
    elif operator == '/':
        # Check for division by zero before performing division
        if num2 == 0:
            return "Error: Cannot divide by zero."
        # Perform division
        return num1 / num2
    else:
        # Handle invalid operator inputs
        return "Error: Invalid operator. Choose from '+', '-', '*', '/'."

# --- Sample Inputs and Outputs ---
# Sample 1: Addition
result_add = simple_calculator(15, 7, '+')
print(f"15 + 7 = {result_add}")

# Sample 2: Subtraction
result_subtract = simple_calculator(20, 5, '-')
print(f"20 - 5 = {result_subtract}")

# Sample 3: Multiplication
result_multiply = simple_calculator(4, 6, '*')
print(f"4 * 6 = {result_multiply}")

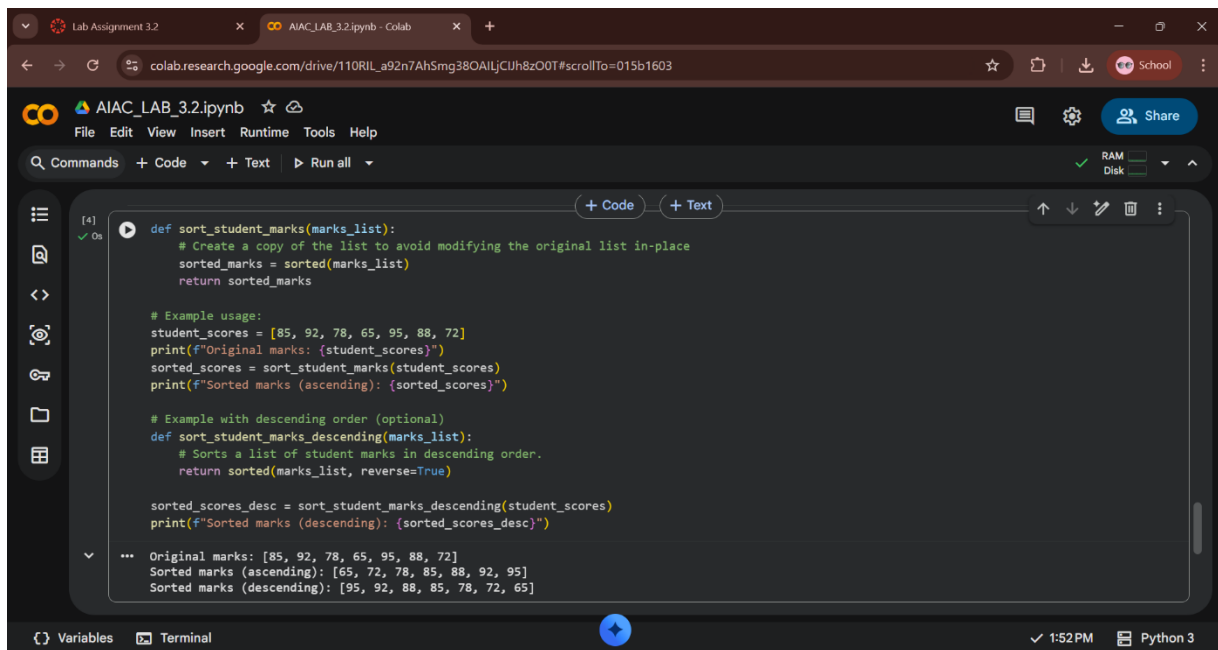
# Sample 4: Division
result_divide = simple_calculator(100, 4, '/')
print(f"100 / 4 = {result_divide}")

# Sample 5: Division by zero (error case)
result_div_by_zero = simple_calculator(5, 0, '/')
print(f"5 / 0 = {result_div_by_zero}")
```

## TASK – 02 : Refining Prompts for Sorting Logic

**Prompt – 01 :** Write a Python Function to sort the Student Marks.

**Code :**

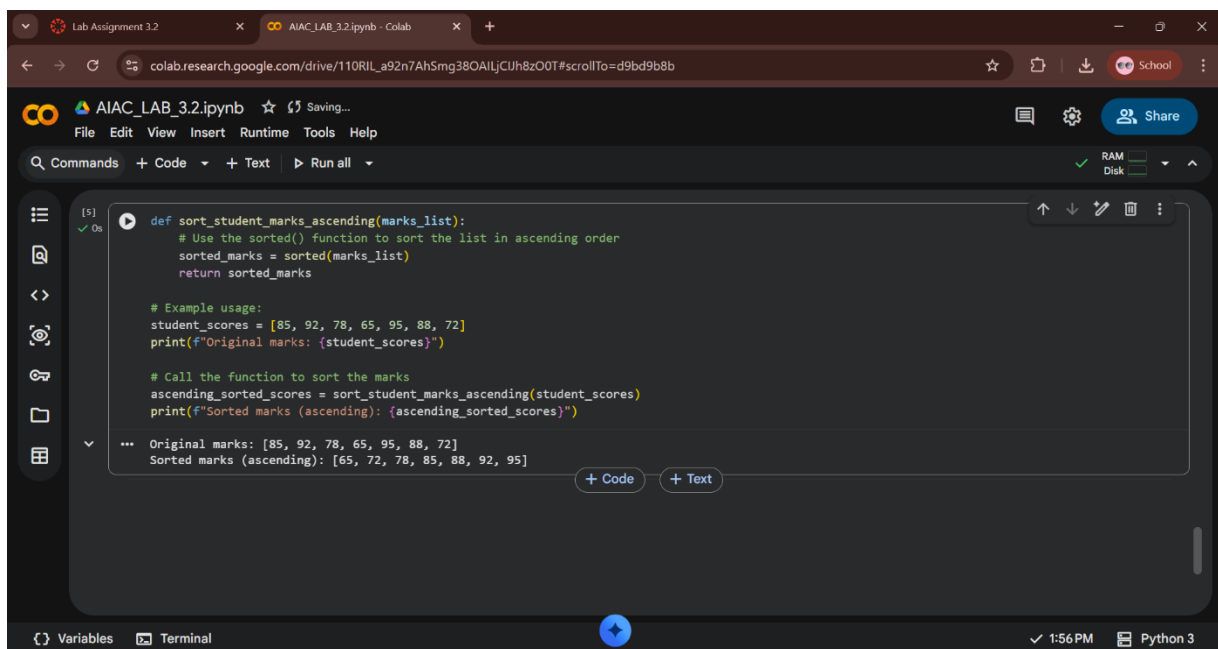


The screenshot shows a Google Colab notebook titled "AIAC\_LAB\_3.2.ipynb". The code defines two functions: `sort_student_marks` and `sort_student_marks_descending`. The first function sorts a list of marks in ascending order, and the second sorts them in descending order. The output shows the original marks [85, 92, 78, 65, 95, 88, 72] and the sorted marks in both ascending [65, 72, 78, 85, 88, 92, 95] and descending [95, 92, 88, 85, 78, 72, 65] orders.

```
[4] def sort_student_marks(marks_list):  
    # Create a copy of the list to avoid modifying the original list in-place  
    sorted_marks = sorted(marks_list)  
    return sorted_marks  
  
    # Example usage:  
    student_scores = [85, 92, 78, 65, 95, 88, 72]  
    print(f"Original marks: {student_scores}")  
    sorted_scores = sort_student_marks(student_scores)  
    print(f"Sorted marks (ascending): {sorted_scores}")  
  
    # Example with descending order (optional)  
    def sort_student_marks_descending(marks_list):  
        # Sorts a list of student marks in descending order.  
        return sorted(marks_list, reverse=True)  
  
    sorted_scores_desc = sort_student_marks_descending(student_scores)  
    print(f"Sorted marks (descending): {sorted_scores_desc}")  
  
... Original marks: [85, 92, 78, 65, 95, 88, 72]  
Sorted marks (ascending): [65, 72, 78, 85, 88, 92, 95]  
Sorted marks (descending): [95, 92, 88, 85, 78, 72, 65]
```

**Prompt – 02 :** Write a Python function that takes a list of student marks and sorts them in ascending order.

**Code :**

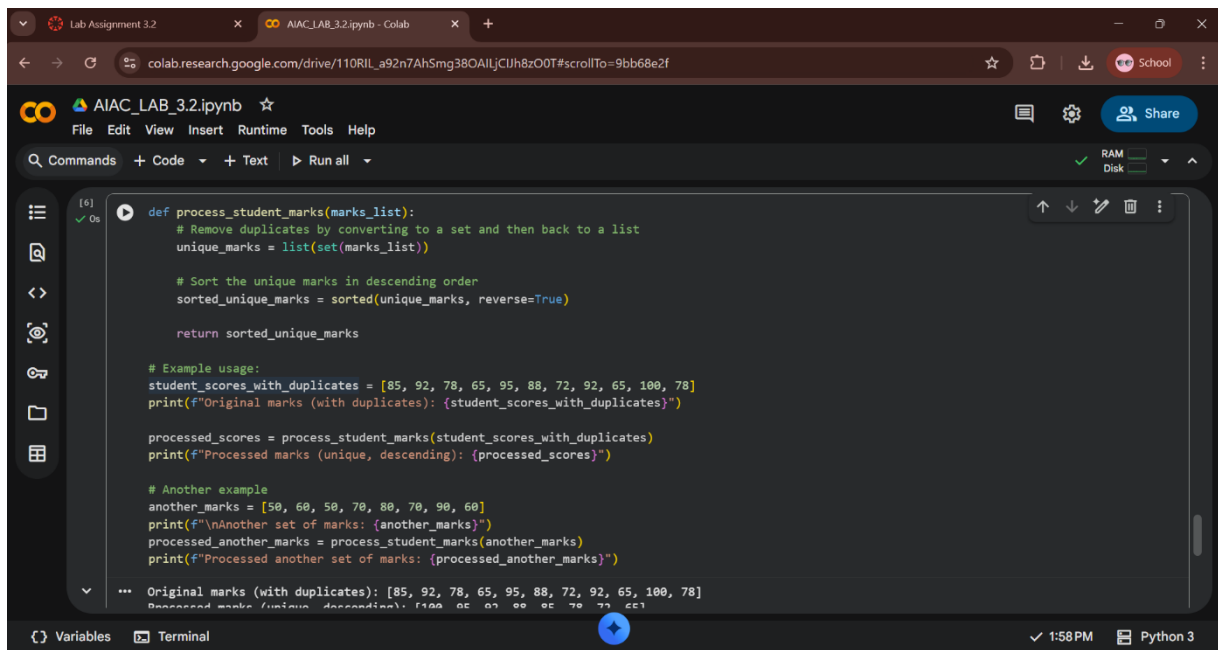


The screenshot shows a Google Colab notebook titled "AIAC\_LAB\_3.2.ipynb". The code defines a function `sort_student_marks_ascending` that sorts a list of marks in ascending order using the `sorted()` function. The output shows the original marks [85, 92, 78, 65, 95, 88, 72] and the sorted marks in ascending order [65, 72, 78, 85, 88, 92, 95].

```
[5] def sort_student_marks_ascending(marks_list):  
    # Use the sorted() function to sort the list in ascending order  
    sorted_marks = sorted(marks_list)  
    return sorted_marks  
  
    # Example usage:  
    student_scores = [85, 92, 78, 65, 95, 88, 72]  
    print(f"Original marks: {student_scores}")  
  
    # Call the function to sort the marks  
    ascending_sorted_scores = sort_student_marks_ascending(student_scores)  
    print(f"Sorted marks (ascending): {ascending_sorted_scores}")  
  
... Original marks: [85, 92, 78, 65, 95, 88, 72]  
Sorted marks (ascending): [65, 72, 78, 85, 88, 92, 95]
```

**Prompt – 03 :** Write an efficient Python function that takes a list of student marks (0–100), sorts them in descending order, removes duplicates, and returns the sorted list.

## Code:



```
[6] ✓ On def process_student_marks(marks_list):
    # Remove duplicates by converting to a set and then back to a list
    unique_marks = list(set(marks_list))

    # Sort the unique marks in descending order
    sorted_unique_marks = sorted(unique_marks, reverse=True)

    return sorted_unique_marks

# Example usage:
student_scores_with_duplicates = [85, 92, 78, 65, 95, 88, 72, 92, 65, 100, 78]
print(f"Original marks (with duplicates): {student_scores_with_duplicates}")

processed_scores = process_student_marks(student_scores_with_duplicates)
print(f"Processed marks (unique, descending): {processed_scores}")

# Another example
another_marks = [50, 60, 50, 70, 80, 70, 90, 60]
print(f"\nAnother set of marks: {another_marks}")
processed_another_marks = process_student_marks(another_marks)
print(f"Processed another set of marks: {processed_another_marks}")

... Original marks (with duplicates): [85, 92, 78, 65, 95, 88, 72, 92, 65, 100, 78]
Processed marks (unique, descending): [100, 95, 92, 88, 85, 78, 72, 65]
```

## **TASK – 03 :** Few-Shot Prompting for Prime Number Validation.

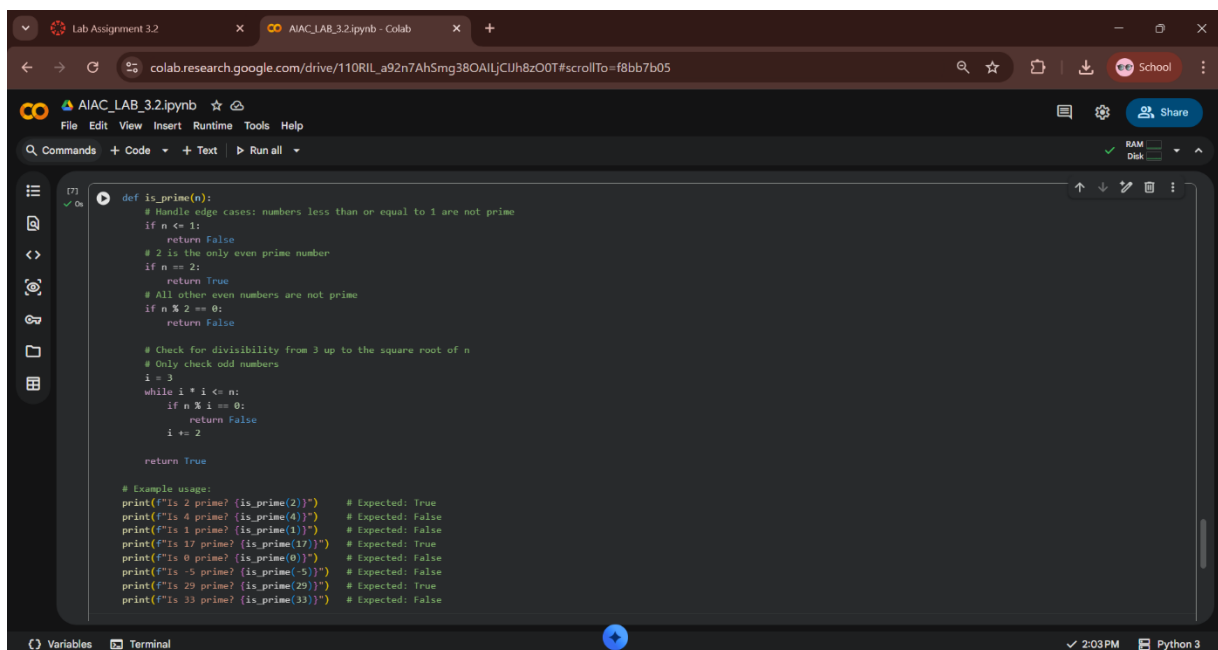
**Prompt:** Write a Python function is\_prime(n) that checks whether a given is Prime or Not, Using Examples:

**I/P:** 2 -> O/P True

**I/P:** 4 -> O/P False

Handle edge cases like negative numbers and 0 properly.

## Code:



```
[7] ✓ On def is_prime(n):
    # Handle edge cases: numbers less than or equal to 1 are not prime
    if n <= 1:
        return False
    # 2 is the only even prime number
    if n == 2:
        return True
    # All other even numbers are not prime
    if n % 2 == 0:
        return False

    # Check for divisibility from 3 up to the square root of n
    # Only check odd numbers
    i = 3
    while i * i <= n:
        if n % i == 0:
            return False
        i += 2
    return True

# Example usage:
print(f"Is 2 prime? {is_prime(2)}") # Expected: True
print(f"Is 4 prime? {is_prime(4)}") # Expected: False
print(f"Is 1 prime? {is_prime(1)}") # Expected: False
print(f"Is 17 prime? {is_prime(17)}") # Expected: True
print(f"Is 0 prime? {is_prime(0)}") # Expected: False
print(f"Is -5 prime? {is_prime(-5)}") # Expected: False
print(f"Is 29 prime? {is_prime(29)}") # Expected: True
print(f"Is 33 prime? {is_prime(33)}") # Expected: False
```

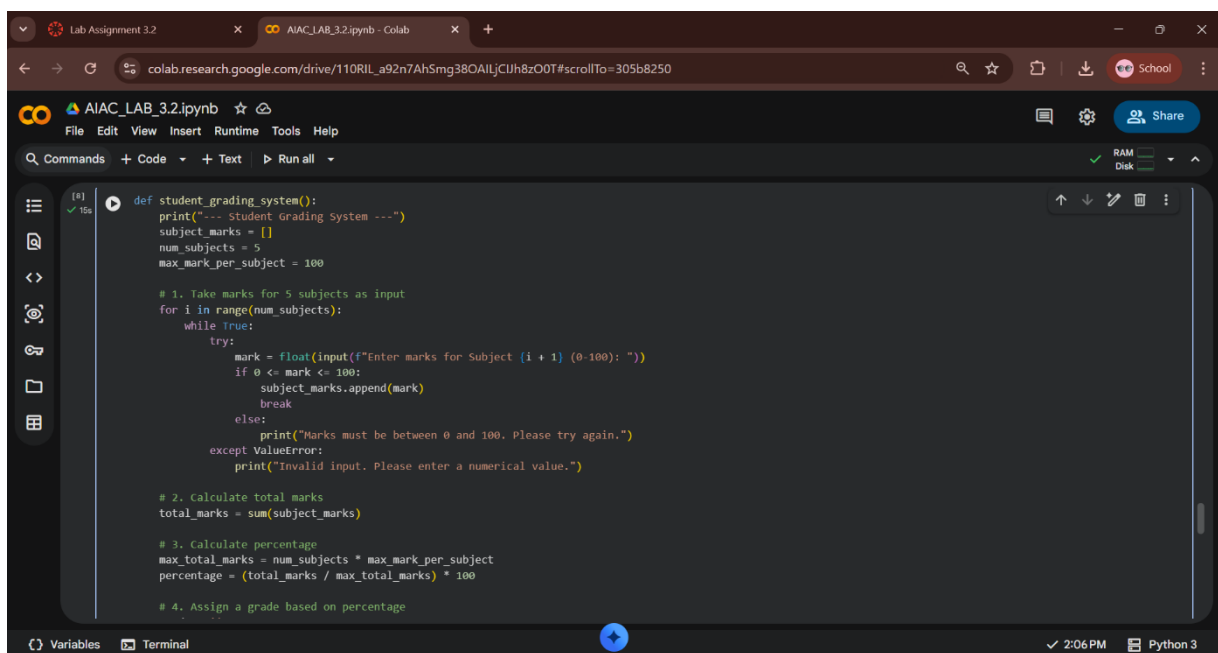
### **TASK – 04 :** Prompt-Guided UI Design for Student Grading System.

**Prompt :** Design a simple Python-based user interface (CLI or GUI) for a student grading system.

The program should:

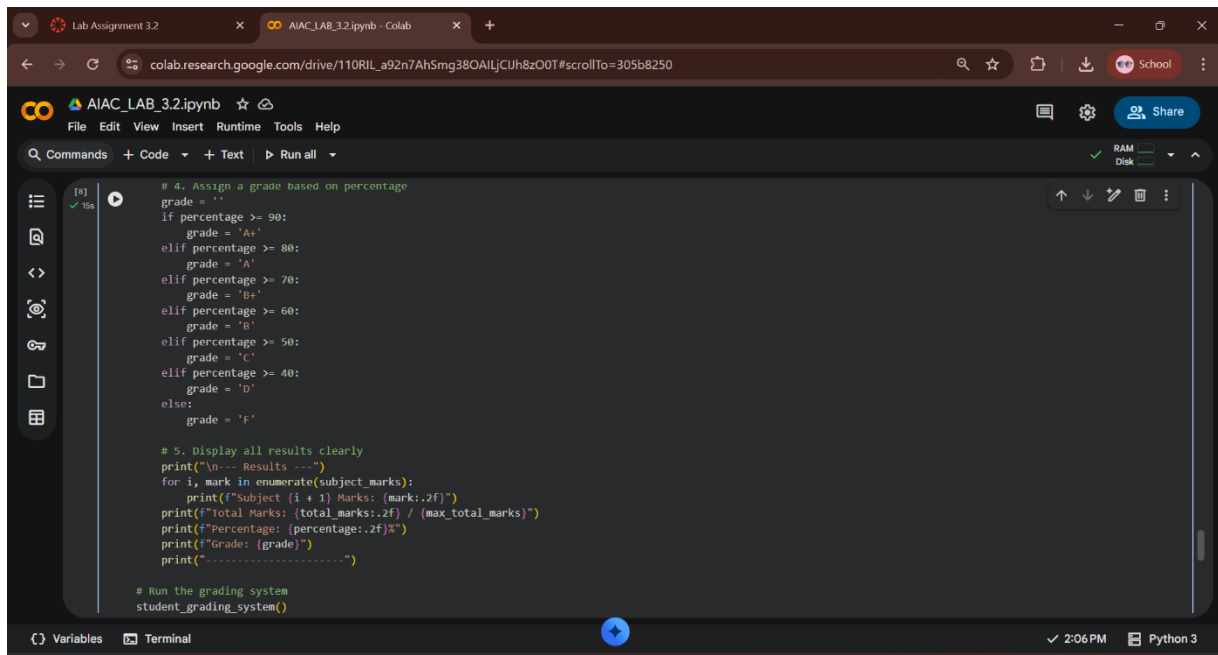
- Take marks for 5 subjects as input
- Calculate total marks
- Calculate percentage
- Assign a grade based on percentage
- Display all results clearly.

### **Code :**



The screenshot shows a Google Colab notebook titled "AIAC\_LAB\_3.2.ipynb". The code defines a function `student_grading_system()` that implements the requirements. It includes comments for each step: taking marks for 5 subjects, calculating total marks, calculating percentage, and assigning a grade. The code uses a loop to take input for each subject and includes error handling for invalid input.

```
[5] ✓ 15s def student_grading_system():  
    print("--- Student Grading System ---")  
    subject_marks = []  
    num_subjects = 5  
    max_mark_per_subject = 100  
  
    # 1. Take marks for 5 subjects as input  
    for i in range(num_subjects):  
        while True:  
            try:  
                mark = float(input(f"Enter marks for Subject {i + 1} (0-100): "))  
                if 0 <= mark <= 100:  
                    subject_marks.append(mark)  
                    break  
            else:  
                print("Marks must be between 0 and 100. Please try again.")  
        except ValueError:  
            print("Invalid input. Please enter a numerical value.")  
  
    # 2. Calculate total marks  
    total_marks = sum(subject_marks)  
  
    # 3. Calculate percentage  
    max_total_marks = num_subjects * max_mark_per_subject  
    percentage = (total_marks / max_total_marks) * 100  
  
    # 4. Assign a grade based on percentage
```

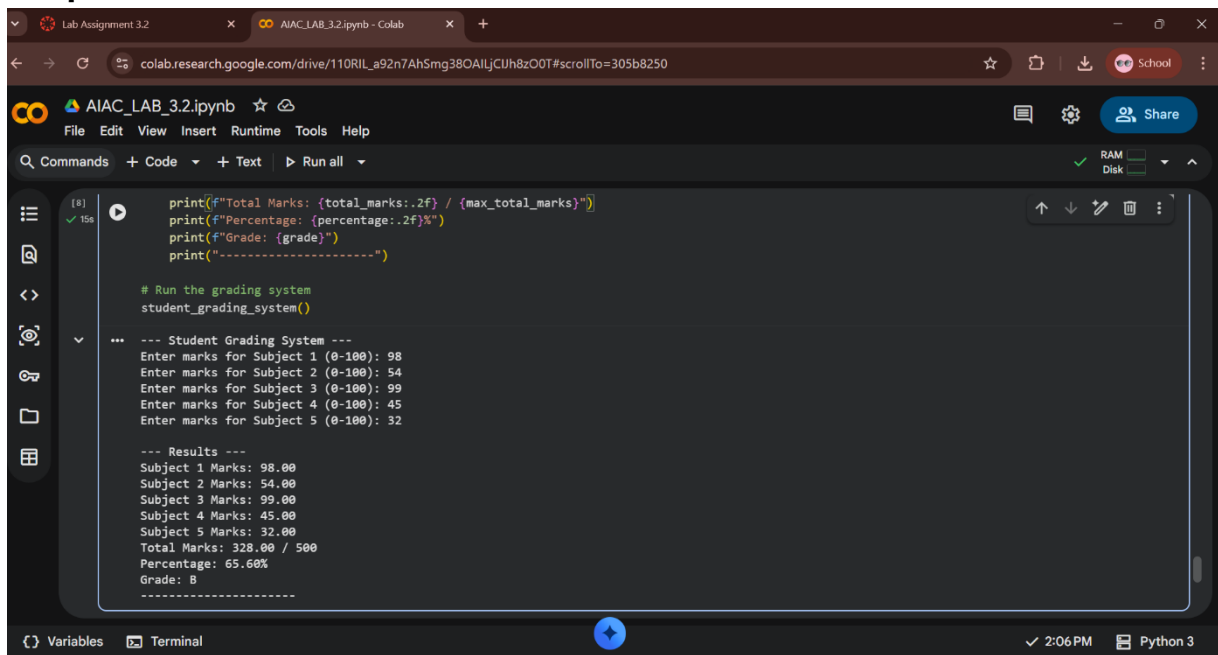


```
# 4. Assign a grade based on percentage
grade = ''
if percentage >= 90:
    grade = 'A+'
elif percentage >= 80:
    grade = 'A'
elif percentage >= 70:
    grade = 'B+'
elif percentage >= 60:
    grade = 'B'
elif percentage >= 50:
    grade = 'C'
elif percentage >= 40:
    grade = 'D'
else:
    grade = 'F'

# 5. Display all results clearly
print("\n--- Results ---")
for i, mark in enumerate(subject_marks):
    print(f"Subject {i + 1} Marks: {mark:.2f}")
print(f"Total Marks: {total_marks:.2f} / {max_total_marks}")
print(f"Percentage: {percentage:.2f}%")
print(f"Grade: {grade}")
print("-----")

# Run the grading system
student_grading_system()
```

## Output:



```
print(f"Total Marks: {total_marks:.2f} / {max_total_marks}")
print(f"Percentage: {percentage:.2f}%")
print(f"Grade: {grade}")
print("-----")

# Run the grading system
student_grading_system()

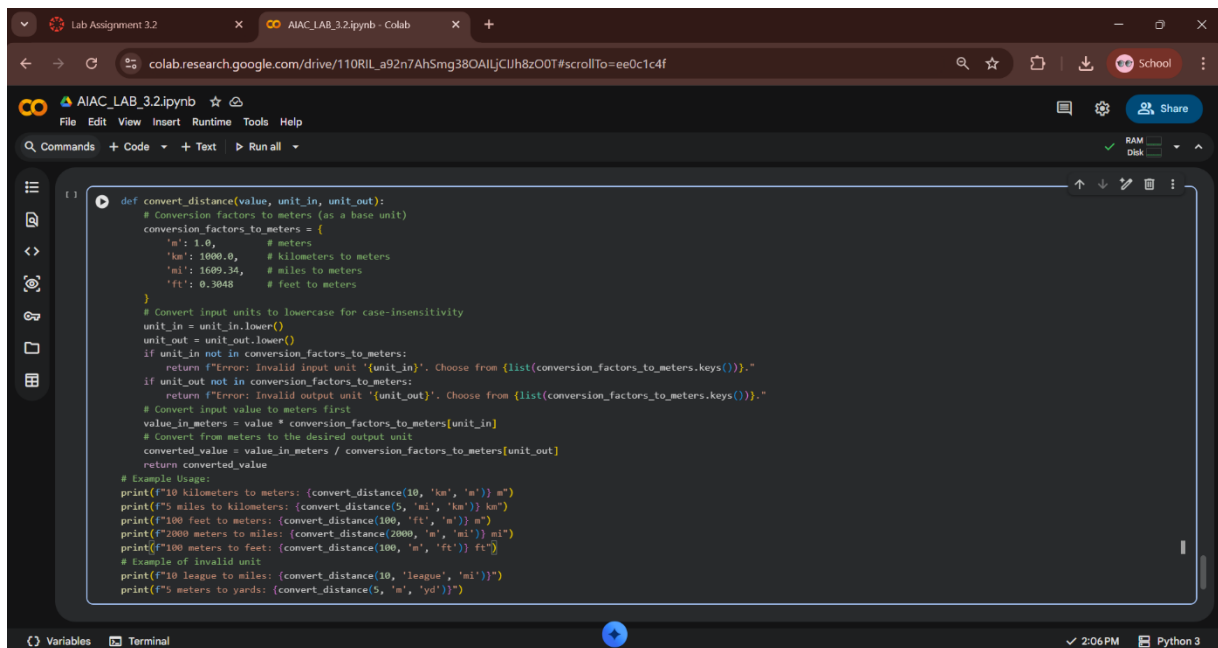
--- Student Grading System ---
Enter marks for Subject 1 (0-100): 98
Enter marks for Subject 2 (0-100): 54
Enter marks for Subject 3 (0-100): 99
Enter marks for Subject 4 (0-100): 45
Enter marks for Subject 5 (0-100): 32

--- Results ---
Subject 1 Marks: 98.00
Subject 2 Marks: 54.00
Subject 3 Marks: 99.00
Subject 4 Marks: 45.00
Subject 5 Marks: 32.00
Total Marks: 328.00 / 500
Percentage: 65.60%
Grade: B
-----
```

**TASK – 05 :** Analyzing Prompt Specificity in Unit Conversion Functions.

**Prompt – 01 :** Write a function to convert distance.

## Code:

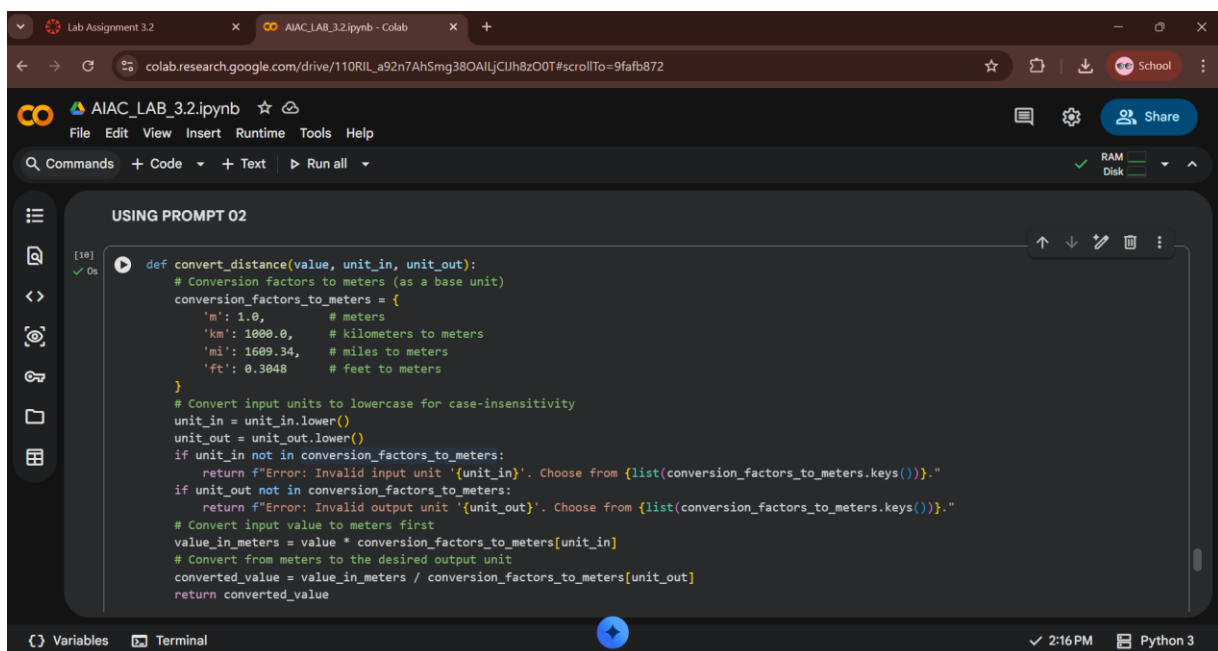


The screenshot shows a Google Colab notebook titled "AIAC\_LAB\_3.2.ipynb". The code defines a function `convert_distance` that takes three arguments: `value`, `unit_in`, and `unit_out`. It uses a dictionary `conversion_factors_to_meters` to convert various units to meters as a base unit. The function includes error handling for invalid input or output units. Example usage is provided at the bottom of the code block.

```
def convert_distance(value, unit_in, unit_out):  
    # Conversion factors to meters (as a base unit)  
    conversion_factors_to_meters = {  
        'm': 1.0,           # meters  
        'km': 1000.0,      # kilometers to meters  
        'mi': 1609.34,     # miles to meters  
        'ft': 0.3048       # feet to meters  
    }  
  
    # Convert input units to lowercase for case-insensitivity  
    unit_in = unit_in.lower()  
    unit_out = unit_out.lower()  
  
    if unit_in not in conversion_factors_to_meters:  
        return f"Error: Invalid input unit '{unit_in}'. Choose from {list(conversion_factors_to_meters.keys())}."  
    if unit_out not in conversion_factors_to_meters:  
        return f"Error: Invalid output unit '{unit_out}'. Choose from {list(conversion_factors_to_meters.keys())}."  
  
    # Convert input value to meters first  
    value_in_meters = value * conversion_factors_to_meters[unit_in]  
    # Convert from meters to the desired output unit  
    converted_value = value_in_meters / conversion_factors_to_meters[unit_out]  
    return converted_value  
  
# Example Usage:  
print(f"10 kilometers to meters: {convert_distance(10, 'km', 'm')} m")  
print(f"5 miles to kilometers: {convert_distance(5, 'mi', 'km')} km")  
print(f"100 feet to meters: {convert_distance(100, 'ft', 'm')} m")  
print(f"2000 meters to miles: {convert_distance(2000, 'm', 'mi')} mi")  
print(f"100 meters to feet: {convert_distance(100, 'm', 'ft')} ft")  
# Example of invalid unit  
print(f"10 league to miles: {convert_distance(10, 'league', 'mi')}")  
print(f"5 meters to yards: {convert_distance(5, 'm', 'yd')}")
```

**Prompt – 02:** Write a Python function to convert kilometers to miles.

## Code :



The screenshot shows a Google Colab notebook titled "AIAC\_LAB\_3.2.ipynb" with the text "USING PROMPT 02" at the top. The code defines a function `convert_distance` that takes three arguments: `value`, `unit_in`, and `unit_out`. It uses a dictionary `conversion_factors_to_meters` to convert various units to meters as a base unit. The function includes error handling for invalid input or output units. Example usage is provided at the bottom of the code block.

```
def convert_distance(value, unit_in, unit_out):  
    # Conversion factors to meters (as a base unit)  
    conversion_factors_to_meters = {  
        'm': 1.0,           # meters  
        'km': 1000.0,      # kilometers to meters  
        'mi': 1609.34,     # miles to meters  
        'ft': 0.3048       # feet to meters  
    }  
  
    # Convert input units to lowercase for case-insensitivity  
    unit_in = unit_in.lower()  
    unit_out = unit_out.lower()  
  
    if unit_in not in conversion_factors_to_meters:  
        return f"Error: Invalid input unit '{unit_in}'. Choose from {list(conversion_factors_to_meters.keys())}."  
    if unit_out not in conversion_factors_to_meters:  
        return f"Error: Invalid output unit '{unit_out}'. Choose from {list(conversion_factors_to_meters.keys())}."  
  
    # Convert input value to meters first  
    value_in_meters = value * conversion_factors_to_meters[unit_in]  
    # Convert from meters to the desired output unit  
    converted_value = value_in_meters / conversion_factors_to_meters[unit_out]  
    return converted_value
```

```
[18] ✓ Os
value_in_meters = value * conversion_factors_to_meters[unit_in]
# Convert from meters to the desired output unit
converted_value = value_in_meters / conversion_factors_to_meters[unit_out]
return converted_value

def kilometers_to_miles(kilometers):
    # 1 kilometer is approximately 0.621371 miles
    miles = kilometers * 0.621371
    return miles

# Example usage of the dedicated function:
km_value = 10
miles_result = kilometers_to_miles(km_value)
print(f"{km_value} kilometers is equal to {miles_result:.2f} miles (using dedicated function).")

# Example usage with the general convert_distance function:
converted_miles = convert_distance(km_value, 'km', 'mi')
print(f"{km_value} kilometers is equal to {converted_miles:.2f} miles (using general converter).")

...
10 kilometers is equal to 6.21 miles (using dedicated function).
10 kilometers is equal to 6.21 miles (using general converter).
```

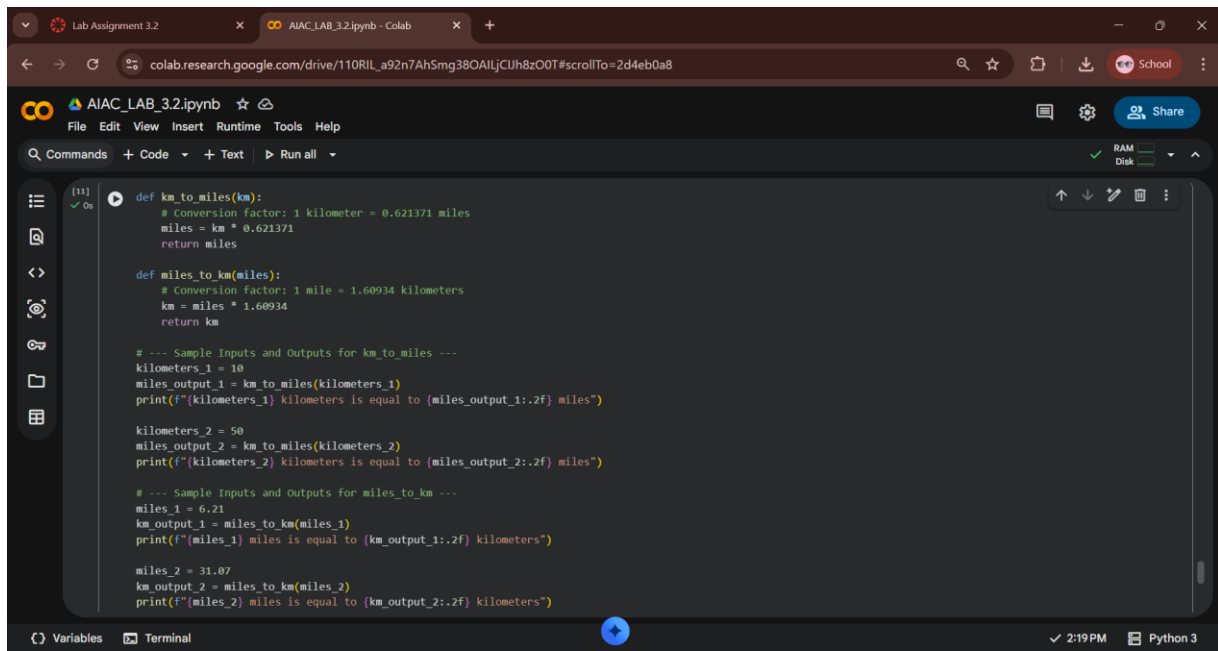
**Prompt – 03 :** Write two Python functions:

1. km\_to\_miles(km)
2. miles\_to\_km(miles)

Use correct conversion factors, add comments, and provide sample inputs and outputs.

**Code :**





The screenshot shows a Google Colab notebook interface. The browser tabs at the top include 'Lab Assignment 3.2' and 'AIAC\_LAB\_3.2.ipynb - Colab'. The address bar shows a Google Drive link. The notebook's menu bar includes 'File', 'Edit', 'View', 'Insert', 'Runtime', 'Tools', and 'Help'. Below the menu is a toolbar with 'Commands', '+ Code', '+ Text', and 'Run all'. On the right, there are icons for RAM and Disk usage, and a 'Share' button. The main code editor contains the following Python code:

```
[11] def km_to_miles(km):  
    # Conversion factor: 1 kilometer = 0.621371 miles  
    miles = km * 0.621371  
    return miles  
  
def miles_to_km(miles):  
    # Conversion factor: 1 mile = 1.60934 kilometers  
    km = miles * 1.60934  
    return km  
  
# --- Sample Inputs and Outputs for km_to_miles ---  
kilometers_1 = 10  
miles_output_1 = km_to_miles(kilometers_1)  
print(f"{kilometers_1} kilometers is equal to {miles_output_1:.2f} miles")  
  
kilometers_2 = 50  
miles_output_2 = km_to_miles(kilometers_2)  
print(f"{kilometers_2} kilometers is equal to {miles_output_2:.2f} miles")  
  
# --- Sample Inputs and Outputs for miles_to_km ---  
miles_1 = 6.21  
km_output_1 = miles_to_km(miles_1)  
print(f"{miles_1} miles is equal to {km_output_1:.2f} kilometers")  
  
miles_2 = 31.07  
km_output_2 = miles_to_km(miles_2)  
print(f"{miles_2} miles is equal to {km_output_2:.2f} kilometers")
```

At the bottom of the interface, there are tabs for 'Variables' and 'Terminal', a status bar showing '2:19 PM' and 'Python 3', and a blue circular icon.

THANK YOU!!