SCHOOL OF COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE			DEPARTMENT OF COMPUTER SCIENCE ENGINEERING
Program Name: B. Tech		**Assignment Type:** Lab	**Academic Year:**2025-2026
CourseCode	23CS002PC304	**Course Title**	AI Assisted Coding
Year/Sem	III/II	**Regulation**	R23
Date and Day of Assignment	**Week1 – Monday**	**Batch**	23CSBTB47B
Name	Yashaswini Mittapalli	**Hall Ticket No**	2303A54049
Assignment Number:1.3(Present assignment number)/**24**(Total number of assignments)			

Q. No.	Question	*Expected Time to complete*
1	Lab 2: Exploring Additional AI Coding Tools beyond Copilot – Gemini (Colab) and Cursor AI **Lab Objectives:** ❖ To explore and evaluate the functionality of Google Gemini for AI-assisted coding within Google Colab. ❖ To understand and use Cursor AI for code generation, explanation, and refactoring. ❖ To compare outputs and usability between Gemini, GitHub Copilot, and Cursor AI. ❖ To perform code optimization and documentation using AI tools. **Lab Outcomes (LOs):** After completing this lab, students will be able to: ❖ Generate Python code using Google Gemini in Google Colab. ❖ Analyze the effectiveness of code explanations and suggestions by Gemini. ❖ Set up and use Cursor AI for AI-powered coding assistance. ❖ Evaluate and refactor code using Cursor AI features. ❖ Compare AI tool behavior and code quality across different platforms. **Task 1: Statistical Summary for Survey Data** ❖ **Scenario:** You are a **data analyst intern** working with survey responses stored as numerical lists. ❖ **Task:** Use **Google Gemini in Colab** to generate a Python function that reads a list of numbers and calculates the **mean, minimum, and maximum** values. ➢ **Prompt :** You are an AI coding assistant. Write a Python function in Google Colab that: Accepts a list of numbers as input Calculates the mean, minimum, and maximum values	Week1 - Monday

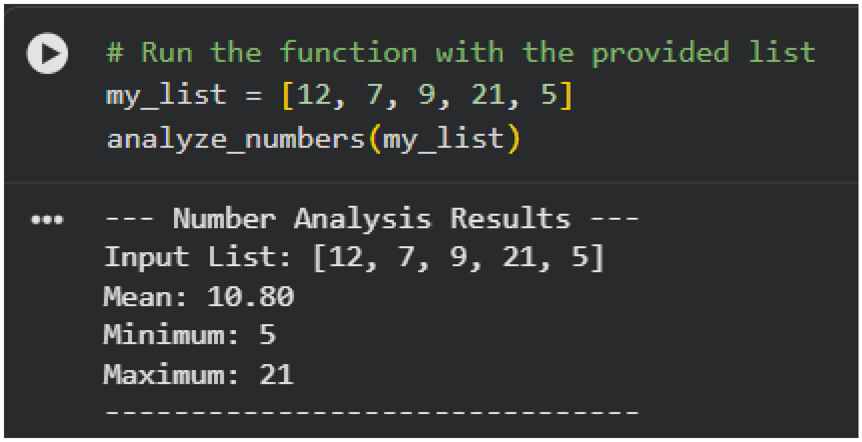
Prints the results clearly

Then run the function on the list [12, 7, 9, 21, 5] and show the output.

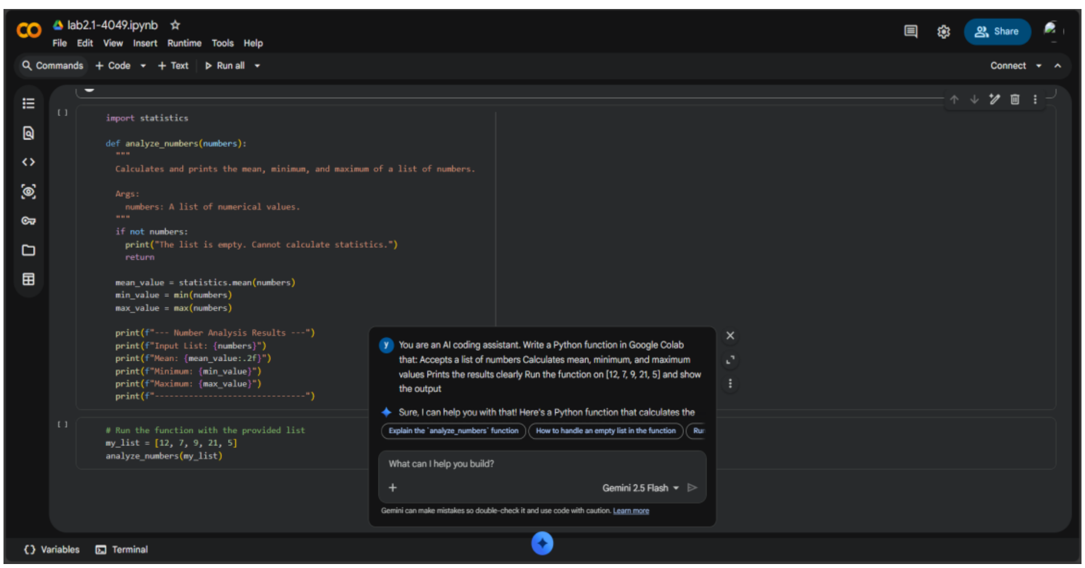
❖ **Expected Output:**

Input List: [12, 7, 9, 21, 5]
Mean: 10.80
Minimum: 5
Maximum: 21

➢ Output shown in Colab



➢ Screenshot of Gemini prompt and result



Task 2: Armstrong Number – AI Comparison

❖ **Scenario:**

You are evaluating AI tools for numeric validation logic.

❖ **Task:**

Generate an **Armstrong number checker** using **Gemini** and **GitHub Copilot**.

Compare their outputs, logic style, and clarity.

❖ **Expected Output:**

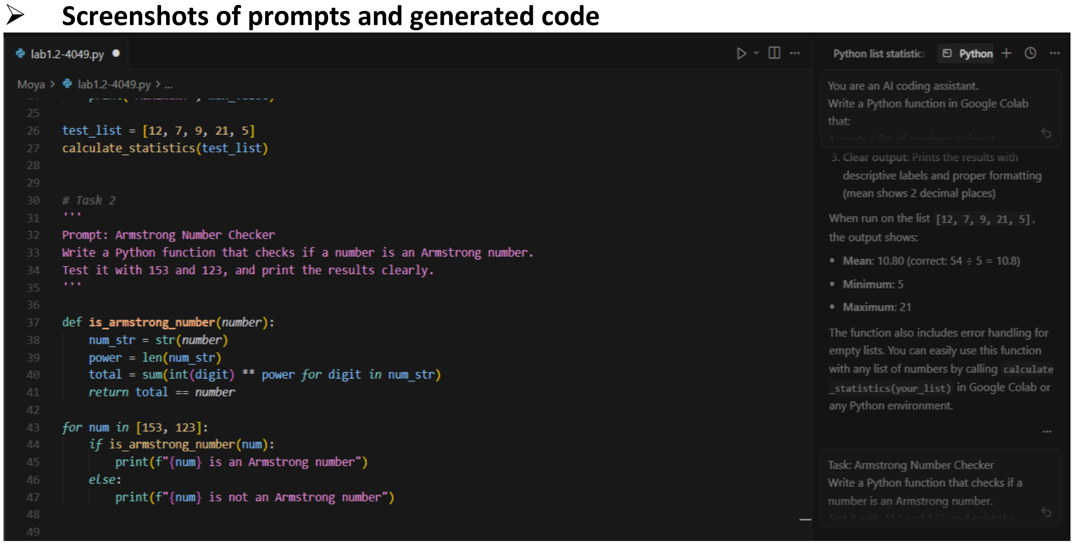
153 IS an Armstrong number.

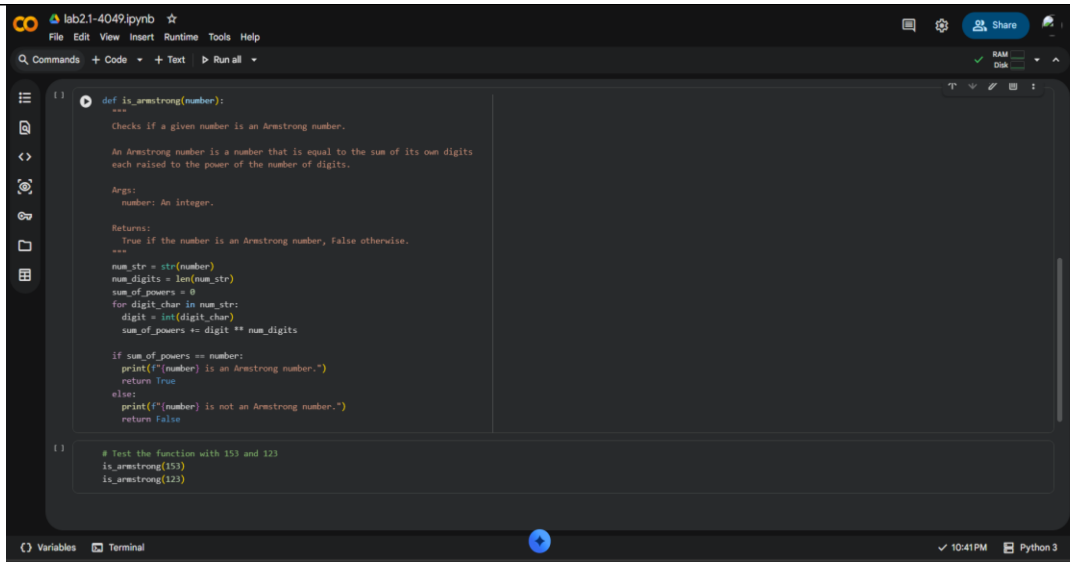
123 IS not an Armstrong number.

Side-by-side comparison table

Point	Google Colab Version	Code (Return-based)
Function ending	print("153 is an Armstrong number")	Same both
Function usage	Can’t reuse result in other logic	Can reuse in if, loops, tests
Main logic location	Logic + output mixed together	Logic inside function, output outside

➢ **Screenshots of prompts and generated code**





Task 3: Leap Year Validation Using Cursor AI

❖ **Scenario:**

You are validating a calendar module for a backend system.

❖ **Task:**

Use **Cursor AI** to generate a Python program that checks whether a given year is a leap year.
Use **at least two different prompts** and observe changes in code.

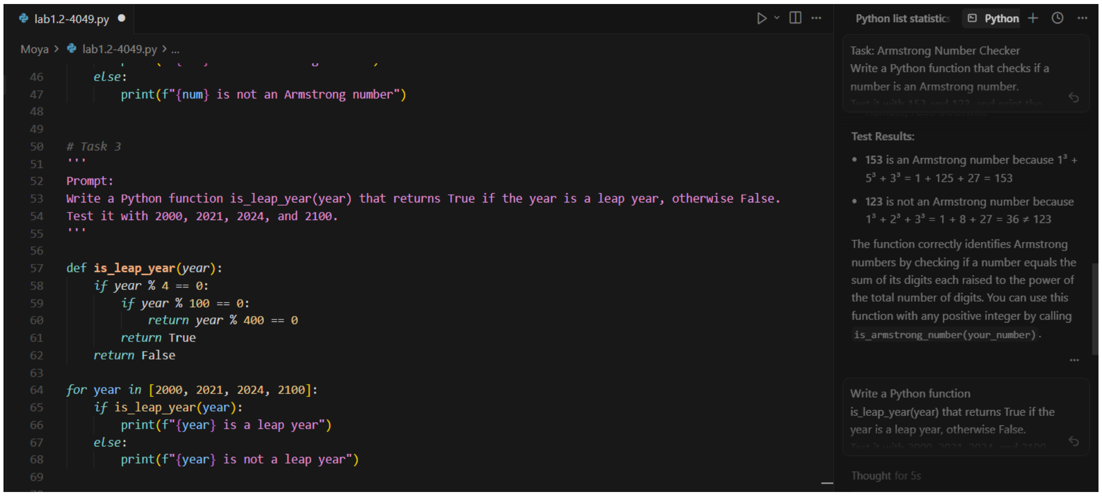
Input:
1900
Output:
1900 is not a leap year

Input:
2000
Output:
2000 is a leap year

Input:
2024
Output:
2024 is a leap year

Input:
2100
Output:
2100 is not a leap year

- ➢ Two versions of code

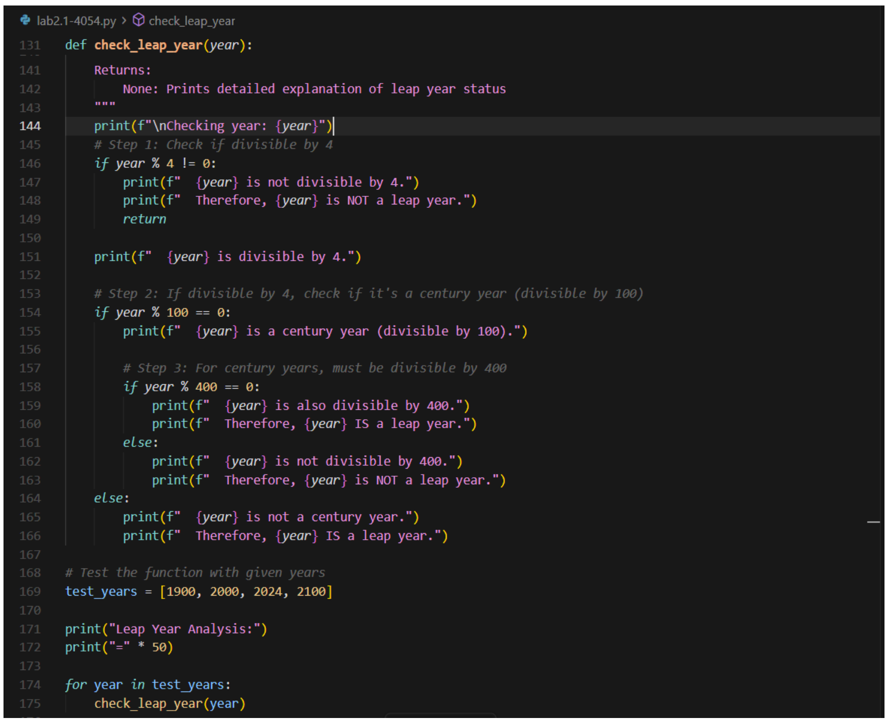


```python
    else:
        print(f"{num} is not an Armstrong number")


# Task 3
'''
Prompt:
Write a Python function is_leap_year(year) that returns True if the year is a leap year, otherwise False.
Test it with 2000, 2021, 2024, and 2100.
'''

def is_leap_year(year):
    if year % 4 == 0:
        if year % 100 == 0:
            return year % 400 == 0
        return True
    return False

for year in [2000, 2021, 2024, 2100]:
    if is_leap_year(year):
        print(f"{year} is a leap year")
    else:
        print(f"{year} is not a leap year")
```

```python
def check_leap_year(year):
    Returns:
        None: Prints detailed explanation of leap year status
    """
    print(f"\nChecking year: {year}")
    # Step 1: Check if divisible by 4
    if year % 4 != 0:
        print(f"  {year} is not divisible by 4.")
        print(f"  Therefore, {year} is NOT a leap year.")
        return

    print(f"  {year} is divisible by 4.")

    # Step 2: If divisible by 4, check if it's a century year (divisible by 100)
    if year % 100 == 0:
        print(f"  {year} is a century year (divisible by 100).")

        # Step 3: For century years, must be divisible by 400
        if year % 400 == 0:
            print(f"  {year} is also divisible by 400.")
            print(f"  Therefore, {year} IS a leap year.")
        else:
            print(f"  {year} is not divisible by 400.")
            print(f"  Therefore, {year} is NOT a leap year.")
    else:
        print(f"  {year} is not a century year.")
        print(f"  Therefore, {year} IS a leap year.")

# Test the function with given years
test_years = [1900, 2000, 2024, 2100]

print("Leap Year Analysis:")
print("=" * 50)

for year in test_years:
    check_leap_year(year)
```


- ➢ Brief comparison :

Key Difference in Code Style

```
if year % 4 == 0:
   if year % 100 == 0:
      return year % 400 == 0
   else:
      return True
else:
   return False
```

Compact style

return (year % 4 == 0 and year % 100 != 0) or (year % 400 == 0)

Task 4: Student Logic + AI Refactoring (Odd/Even Sum)

❖ **Scenario:**

Company policy requires developers to write logic before using AI.

❖ **Task:**

Write a Python program that calculates the **sum of odd and even numbers in a tuple**, then refactor it using any AI tool.

Prompt:

Write a Python program to find the sum of odd numbers and sum of even numbers in a tuple using basic logic.

Then refactor the code using an AI tool to make it simpler.

Show:

1. Original code

```python
# Original code
def calculate_sums_original(numbers):
    even_sum = 0
    odd_sum = 0

    for num in numbers:
        if num % 2 == 0:
            even_sum += num
        else:
            odd_sum += num

    return even_sum, odd_sum
```

2. Refactored code

```python
# Refactored code (AI Version)
def calculate_sums_refactored(numbers):
    even_sum = sum(num for num in numbers if num % 2 == 0)
    odd_sum (variable) even_sum: int ers if num % 2 != 0)
    return even_sum, odd_sum

test_tuple = (1, 2, 3, 4, 5, 6)
```

3. Short explanation of changes

Input: (1, 2, 3, 4, 5, 6)

❖ **Expected Output:**

```
Original Code Output:
Even sum: 12
Odd sum: 9

Refactored Code Output:
Even sum: 12
Odd sum: 9
```

➢ **Explanation of improvements**

1. Code is **simpler and easier to read**
2. No need to write loops manually
3. Uses Python’s built-in sum() function for clarity