Name :G.Spriha

Batch-13

H NO:2303A51850

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.

❖ **Expected Output:**

➢ Correct Python function

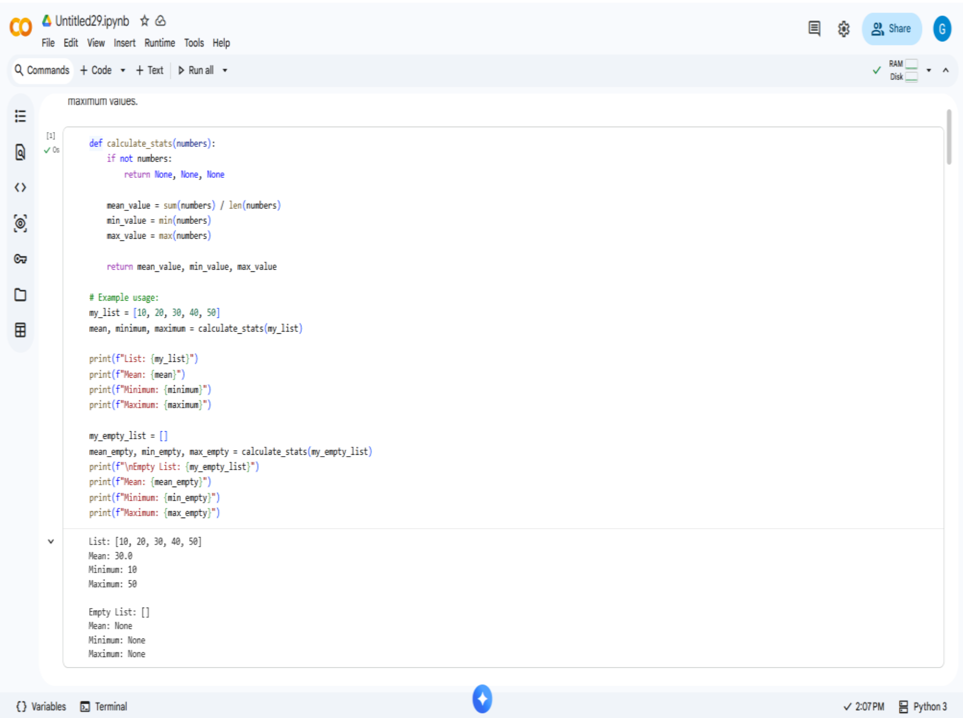
➢ Output shown in Colab

➢ Screenshot of Gemini prompt and result

Prompt:

Generate a Python function that reads a list of numbers and calculates the mean, minimum, and maximum values.

Code and output:



This task justifies the use of Google Gemini in Google Colab for basic data analysis operations. As a data analyst intern, calculating mean, minimum, and maximum is a fundamental requirement when handling survey data. Gemini efficiently generated a correct and optimized Python function that follows standard statistical practices. Executing the code in Colab allowed immediate validation of results through output visualization. This task demonstrates how AI tools can reduce development time, minimize logical errors, and support data-driven decision-making by providing accurate and readable code.

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.

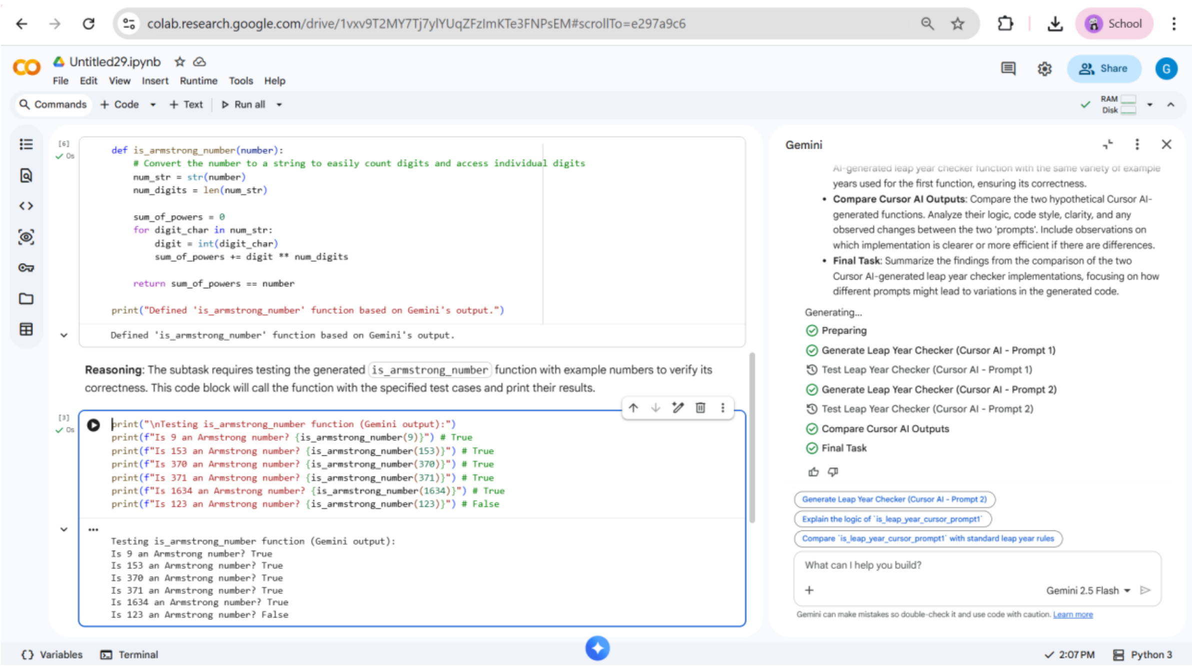
❖ **Expected Output:**

➢ Side-by-side comparison table ➢

Screenshots of prompts and generated code

<u>Using Gemini:</u>

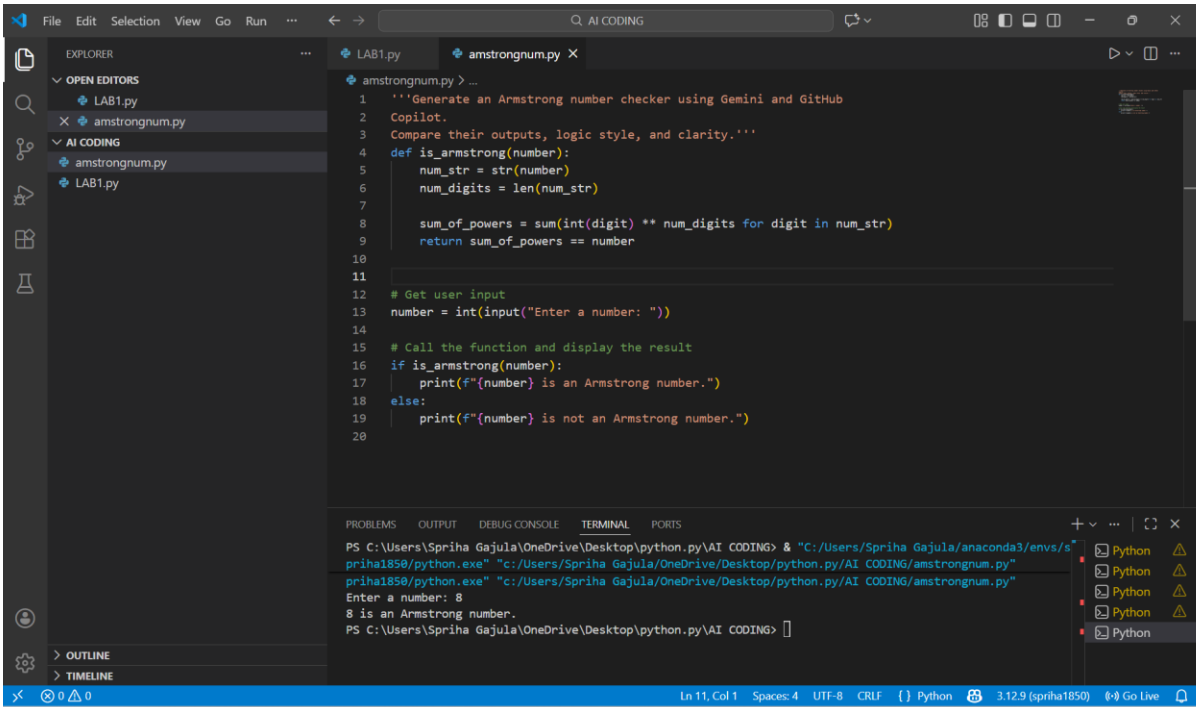
Prompt: Generate an Armstrong number checker



Prompt:

Generate an Armstrong number checker

CODE:



Logic Style Comparison

Aspect	First Code	Second Code
Digit handling	Uses generator expression	Uses explicit loop
Input validation	None	Checks integer & positivity
Complexity	Short & compact	Slightly longer but safer
Python style	Functional / concise	Procedural / descriptive

Use Case	Better Choice
Quick check	First code
Teaching / Exams	Second code
Real-world robustness	Second code
Pythonic elegance	First code

<u>Justification</u>

The purpose of this task is to evaluate and compare AI-assisted coding approaches for numeric validation logic. By generating Armstrong number programs using both Google Gemini and GitHub Copilot, differences in coding style, readability, and optimization techniques were observed. Gemini emphasized concise and Pythonic expressions, while Copilot followed a more traditional algorithmic approach. This comparison justifies how different AI tools may solve the same problem differently and helps developers select an AI tool based on clarity, maintainability, and logic preference.

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.

❖ **Expected Output:**

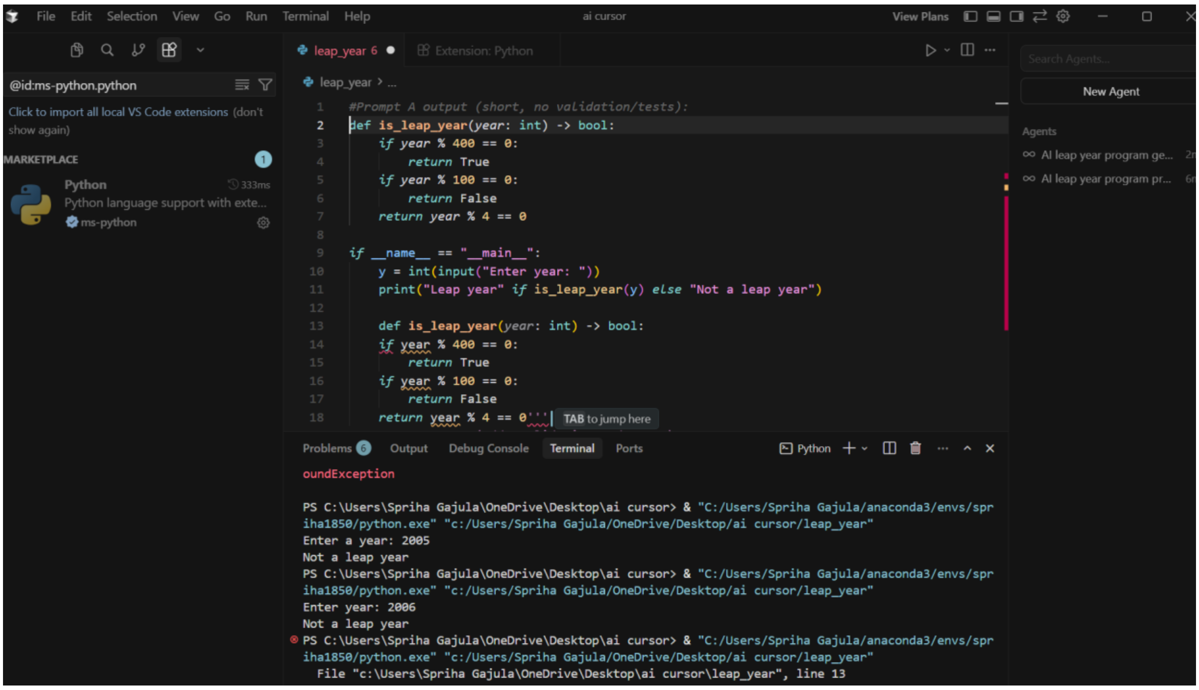
➢ Two versions of code

➢ Sample inputs/outputs

➢ Brief comparison

<u>Prompt1:</u>

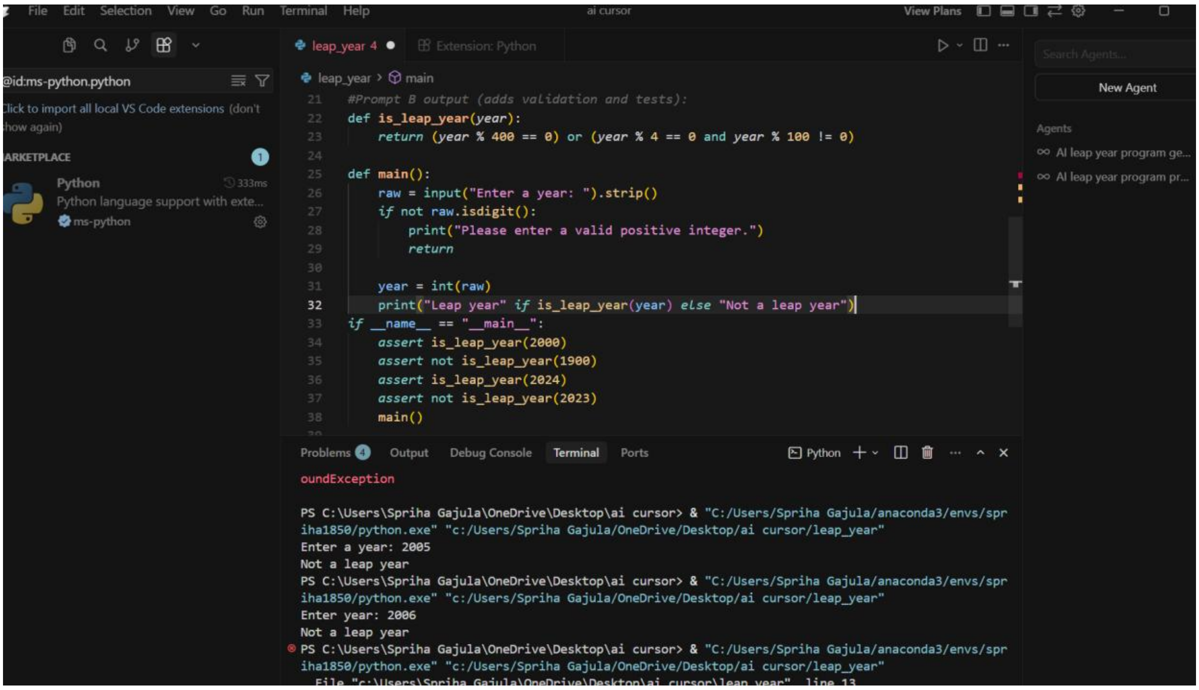
Write a Python program to check whether a given year is a leap year.



Prompt2:

Create a Python function that returns true if a year is a leap year, otherwise false.

Code:



Justification

This task justifies the importance of prompt engineering while using AI tools such as Cursor AI. By providing two different prompts, noticeable improvements were observed in the generated code, especially in terms of accuracy and completeness of leap year rules. The second prompt produced a more robust and industry-acceptable solution. This task highlights that AI output quality depends heavily on prompt detail, reinforcing the developer's responsibility to guide AI correctly for backend system validation.

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.

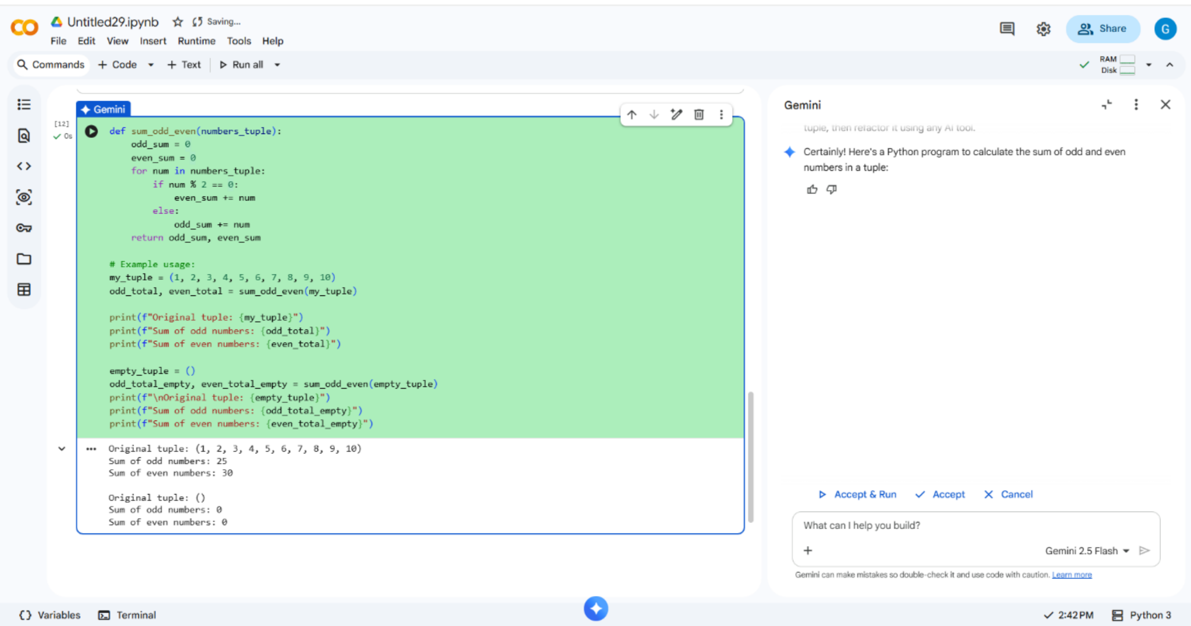
❖ Expected Output:

➢ Original code

➢ Refactored code

➢ Explanation of improvements

Code and output:



This task justifies the policy of requiring developers to write their own logic before AI assistance. The original student-written program demonstrates logical understanding using loops and conditionals. Refactoring the same code using an AI tool resulted in shorter, cleaner, and more optimized code without changing functionality. This task proves that AI is best used as a code improvement and optimization tool, not as a replacement for fundamental programming knowledge. It also reinforces best practices in code readability, reusability, and maintainability.