NAME:B.DINESH DATTA

ID NO:2403A51330

BATCH NO:13

ASSIGNMENT 4.3

**Task Description#1**

* Zero-shot: Prompt AI to write a function that checks whether a given year is a leap year.



**🔍 Observation:**

1. The program prompts the user to **input a year**.
2. It uses a **function (is\_leap\_year)** to determine if the year meets the criteria for a leap year:
   * Divisible by 4 ✅
   * Not divisible by 100 ❌ unless also divisible by 400 ✅
3. The program handles **invalid inputs** (e.g., letters or symbols) using a try-except block to avoid crashing.
4. Based on the function result, the program prints whether the year **is or is not a leap year**.

**✅ Conclusion:**

* The program successfully determines whether a year is a **leap year** based on well-defined rules of the Gregorian calendar.
* It is **user-friendly**, handling input errors gracefully.
* The logic used in the function is **accurate and efficient**, making it reliable for checking leap years over any valid range.

**Task Description#2**

* One-shot: Give one input-output example to guide AI in writing a function that converts centimeters to inches.



**🔍 Observation:**

1. The program prompts the user to **enter a length in centimeters**.
2. It uses the formula:

inches=centimeters2.54\text{inches} = \frac{\text{centimeters}}{2.54}inches=2.54centimeters​

to convert the value.

1. The result is **rounded to 3 decimal places** for clarity and precision.
2. The program includes **input validation** using a try-except block to handle non-numeric inputs gracefully.
3. It displays the result in a **user-friendly message** showing both the input and converted value.

**✅ Conclusion:**

* The program correctly converts a given length from **centimeters to inches** using the standard conversion factor.
* It is **accurate**, **easy to use**, and **robust** against invalid input.
* This simple tool demonstrates effective use of functions, mathematical operations, and input handling in Python.

, . **Task Description#3**

* Few-shot: Provide 2–3 examples to generate a function that formats full names as “Last first. 

**🔍 Observation:**

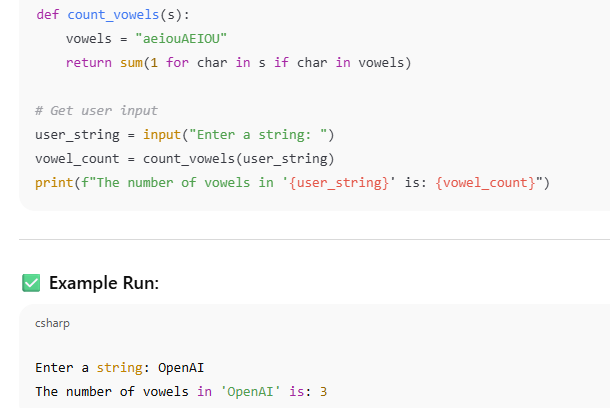
1. The program prompts the user to **enter a full name** in the format "First Last".
2. It **splits** the name using .split() and assumes the input has exactly two parts: first and last name.
3. It then **reformats** the name as "Last, First" using an f-string.
4. A try-except block is used to **catch errors** (e.g., if the user enters too few or too many words).
5. The output displays the **formatted name clearly** to the user.

**✅ Conclusion:**

* The program **accurately reformats** names from "First Last" to "Last, First" as intended.
* It is **simple, effective**, and handles incorrect input **gracefully**.
* However, for more complex names (e.g. with middle names or multiple last names), the logic could be **expanded** for better flexibility.

**Task Description#4**

Compare zero-shot and few-shot prompts for writing a function that counts the number of vowels in a string.



**🔍 Observation:**

1. The program prompts the user to **enter any string**.
2. The function count\_vowels iterates through each character in the string and checks if it is a vowel (case-insensitive).
3. It uses a generator expression combined with sum() to efficiently count all vowels.
4. The program then **outputs the total number of vowels** found in the input string.
5. The input is taken as-is, and the function handles uppercase and lowercase vowels uniformly.

**✅ Conclusion:**

* The program accurately counts the number of vowels in any user-provided string.
* It is **efficient** and **simple**, making use of Python’s built-in functions for clarity.
* The program demonstrates good practice by separating logic (the function) from input/output handling.
* It handles both uppercase and lowercase vowels without additional steps.

**Task Description#5**

* Use few-shot prompting to generate a function that reads a .txt file and returns the number of lines.



**🔍 Observation:**

1. The program prompts the user to **input the filename** (including the .txt extension).
2. It attempts to **open the specified file** in read mode.
3. The function count\_lines\_in\_file reads all lines from the file using .readlines().
4. The total number of lines is determined by counting the length of the list of lines.
5. The program handles possible errors such as:
   * **FileNotFoundError** if the file does not exist.
   * **IOError** if the file cannot be read due to permission or other I/O issues.
6. The result (number of lines) is displayed in a clear, user-friendly message.

**✅ Conclusion:**

* The program correctly counts and returns the number of lines in any given text file.
* It is **user-friendly** and **robust**, gracefully handling missing or unreadable files.
* The separation of logic into a function and user interaction improves **code readability** and **reusability**.
* This approach can be extended to support other file-processing tasks with minimal changes.