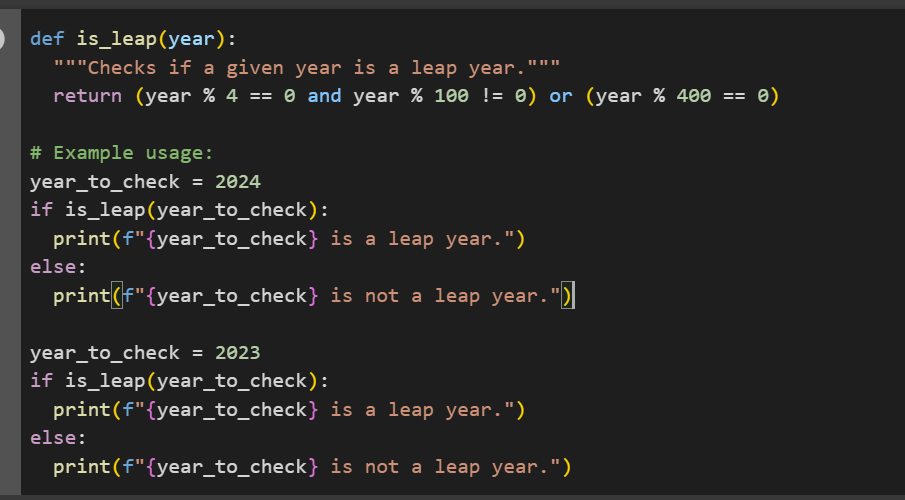
**ASSIGNMENT-4.3**

**TASK-1:**

Write a python code to check the whether it is leap year or not



**OUTPUT:**

A black background with white text

AI-generated content may be incorrect.

**EXPLANATION:**

This code defines a function called is\_leap that takes a year as input and determines if it's a leap year based on the following rules:

* A year is a leap year if it is divisible by 4.
* However, if the year is divisible by 100, it is **not** a leap year, unless...
* The year is also divisible by 400. In that case, it **is** a leap year.

**TASK-2:**

**GIVE ME A CODE OF INPUT-OUTPUT ONE SHOT EXAMPLE GUIDING AI I WRITING A FUNCTION TO CONVERTS CENTIMETERS TO INCHES FUNCTION WITH CORRECT CONVERSION LOGIC**

**CODE:**

A screenshot of a computer code

AI-generated content may be incorrect.

**OUTPUT:**

A black text on a white background

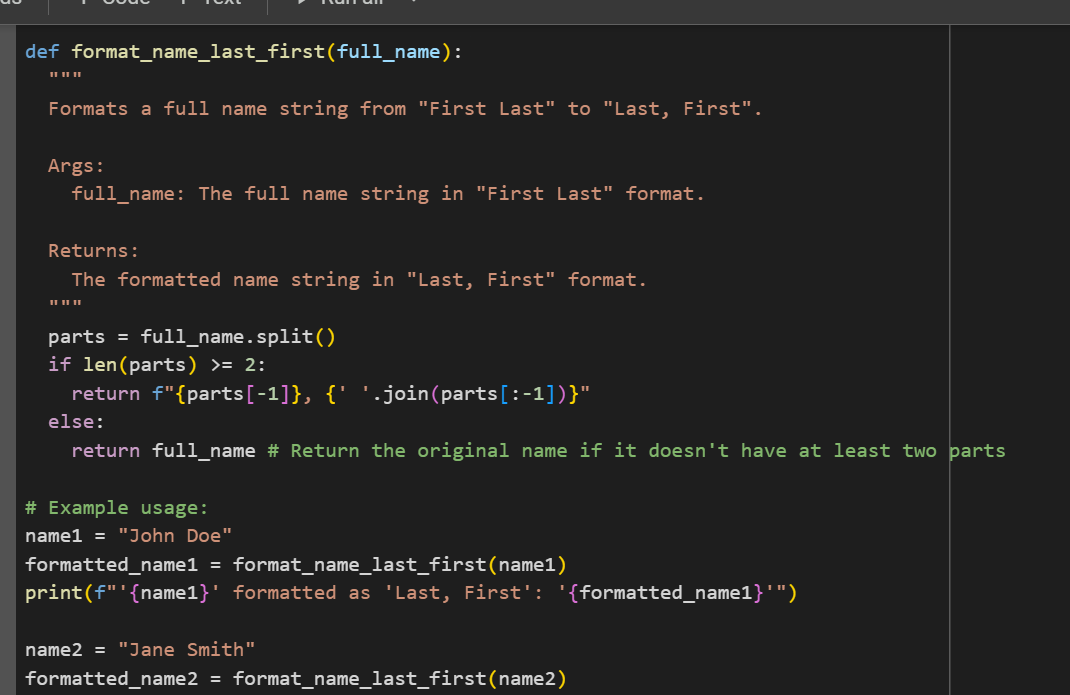
AI-generated content may be incorrect.

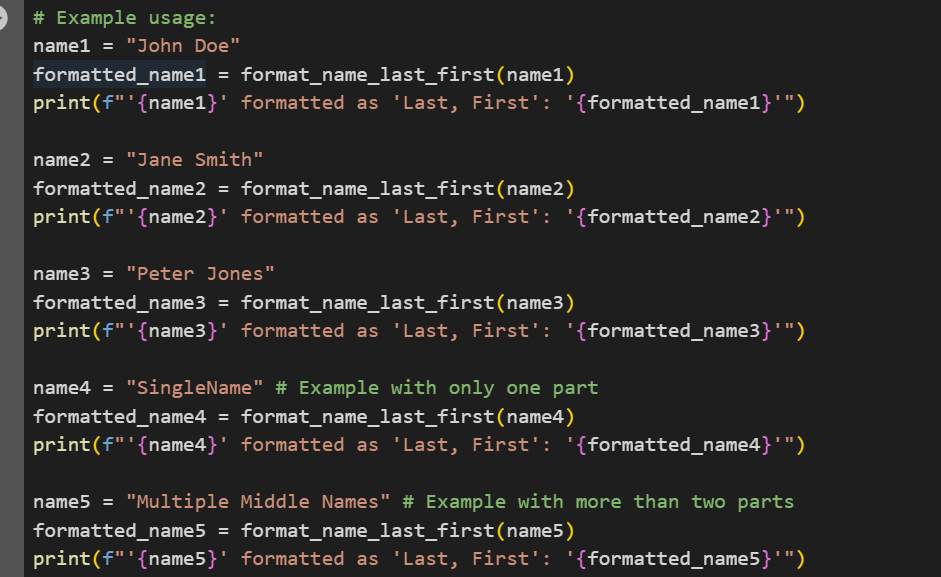
EXPLANATION:

This code defines a function centimeters\_to\_inches that takes a value in centimeters and converts it to inches using the conversion factor 0.393701. It then shows an example converting 100 centimeters to inches

**TASK-3:**

Write a function that formats names as 'Last, First' using 2–3 examples.





OUTPUT:

A screenshot of a computer

AI-generated content may be incorrect.

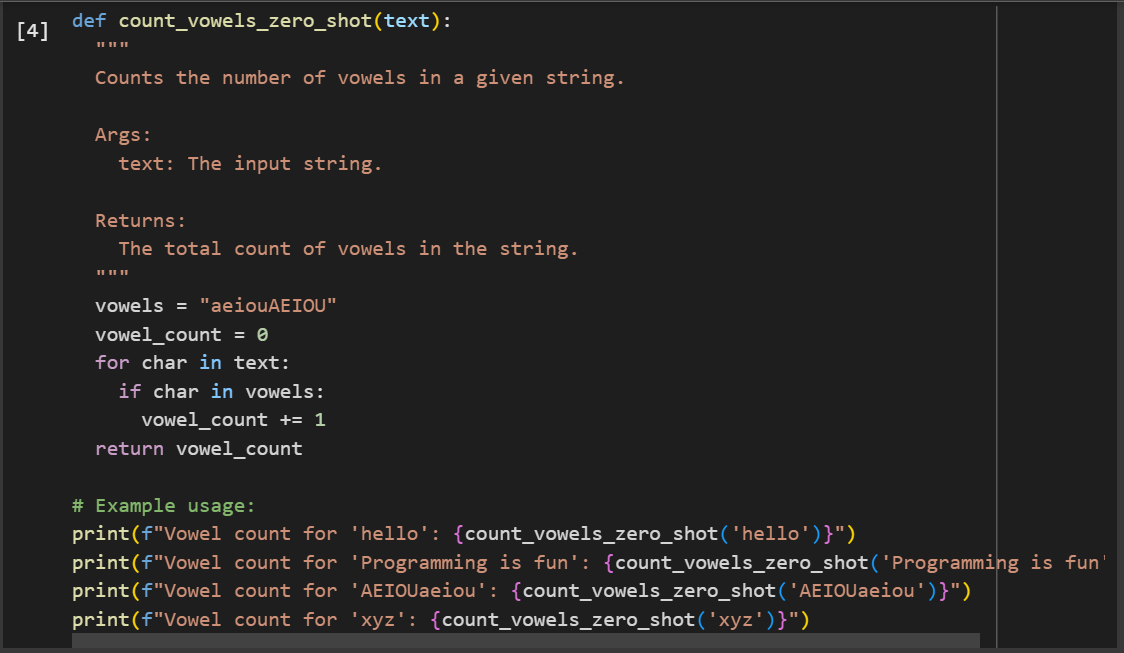
EXPLANATION:

The code defines a Python function called format\_name\_last\_first that takes a full\_name string as input and attempts to reformat it as "Last, First".

Here's a breakdown of how it works:

1. **Splitting the name:** It first splits the full\_name string into a list of parts using spaces as delimiters (full\_name.split()).
2. **Checking for at least two parts:** It then checks if the list of parts has at least two elements. This is to ensure there's at least a first and a last name.
3. **Formatting:**
   * If there are two or more parts, it takes the last element of the list (parts[-1]) as the last name and joins the remaining elements (parts[:-1]) with spaces to form the first name (handling potential middle names). It then returns a formatted string in the "Last, First" format.
   * If there are fewer than two parts (e.g., a single name), it returns the original full\_name as is.
4. **Examples:** The code then shows several examples of how to use the function with different types of names, including names with a single part and names with multiple middle names, and prints the results.

**TASK-4:** **Write code to compare zero-shot vs. few-shot prompts for a function that counts vowels in a string, with outputs and reflection."**



Output:

A screen shot of a computer

AI-generated content may be incorrect.

**EXPLANATION:**

1. **Define vowels:** It initializes a string vowels containing all lowercase and uppercase vowels ("aeiouAEIOU").
2. **Initialize count:** It initializes a variable vowel\_count to 0 to keep track of the number of vowels found.
3. **Iterate through characters:** It loops through each character in the input text string.
4. **Check if vowel:** Inside the loop, for each char, it checks if that char is present in the vowels string.
5. **Increment count:** If the character is a vowel, it increments the vowel\_count by 1.
6. **Return count:** After iterating through all characters, the function returns the final vowel\_count

TASK-5: "Generate a Python function (few-shot) to read a .txt file and return its line count."

Code:

A screenshot of a computer screen

AI-generated content may be incorrect.

A computer screen shot of text

AI-generated content may be incorrect.

**OUTPUT:**

A close up of a text

AI-generated content may be incorrect.

**EXPLANATION :**

1. The code defines a Python function called count\_lines\_in\_file that takes a filename as input.

2. It uses a try-except block to handle potential errors, specifically FileNotFoundError.

3. Inside the try block, it opens the specified file in read mode ('r') using a with statement, which ensures the file is automatically closed.

4. It uses a generator expression (1 for line in f) to iterate through each line of the file.

5. sum() is used to count the number of 1s generated by the expression, effectively counting the lines.

6. If the file is not found, the except FileNotFoundError block is executed.

7. In the except block, it returns an error message indicating that the file was not found.

8. The code then demonstrates the function's usage by creating a dummy file named "sample.txt".

9. It calls count\_lines\_in\_file with "sample.txt" and prints the result.

10. Finally, it calls the function with a non-existent file to show how the error handling works.