*Name – MD Danish*

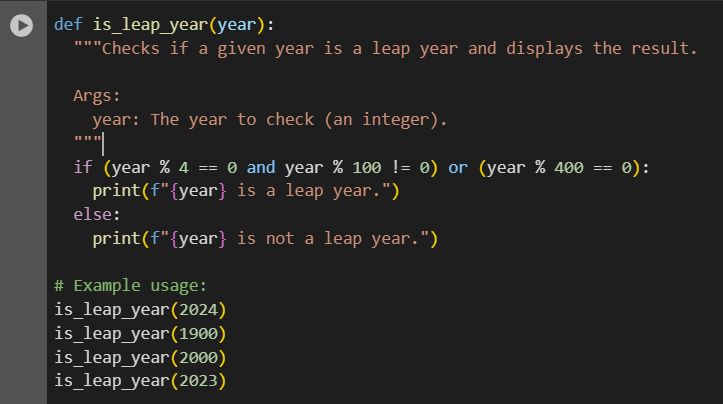
*Roll – 2403A54093*

Batch – 03 (DS)

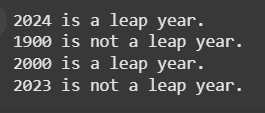
Task Description#1  
● Zero-shot: Prompt AI to write a function that checks whether a given year is a leap  
year.

PROMPT: Write a function in python which takes input from user to calculate whether a given year is a leap year and display it.

CODE:



Output:



Explanation:

This code defines a Python function called is\_leap\_year that takes one argument, year.

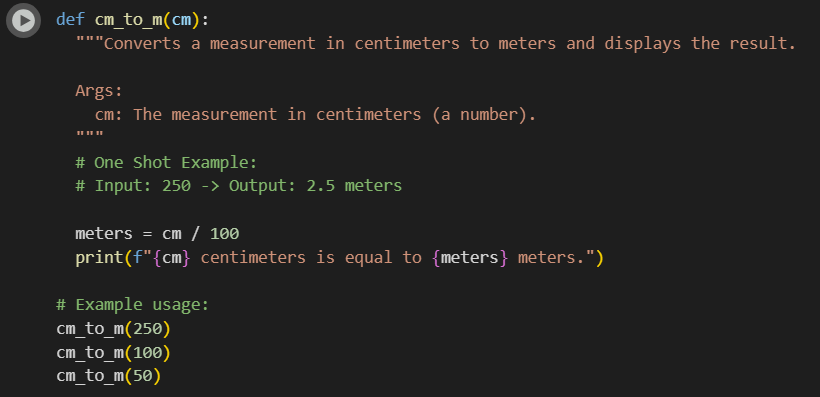
* The function checks if the given year is a leap year using the standard rules:
  + A year is a leap year if it is divisible by 4 but not by 100, unless it is also divisible by 400.
* It then prints a message indicating whether the year is a leap year or not.
* The example usage demonstrates how to call the function with different years (2024, 1900, 2000, and 2023).

Task Description#2  
● One-shot: Give one input-output example to guide AI in writing a function that  
converts centimeters to inches.

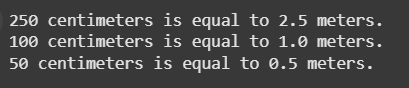
PROMPT 1: Write a function in python code which takes input from user to convert centimeter into inches and display it

PROMPT 2:add one shot as comments

CODE:



OUTPUT:



EXPLANATION:  
This code defines a Python function called cm\_to\_m that converts a measurement in centimeters to meters.

* It takes one argument, cm, which is the measurement in centimeters.
* Inside the function, it divides the cm value by 100 to get the equivalent measurement in meters and stores it in the meters variable.
* Finally, it prints a formatted string showing the original centimeter value and the converted meter value.
* The example usage demonstrates how to call the function with different centimeter values (250, 100, and 50).

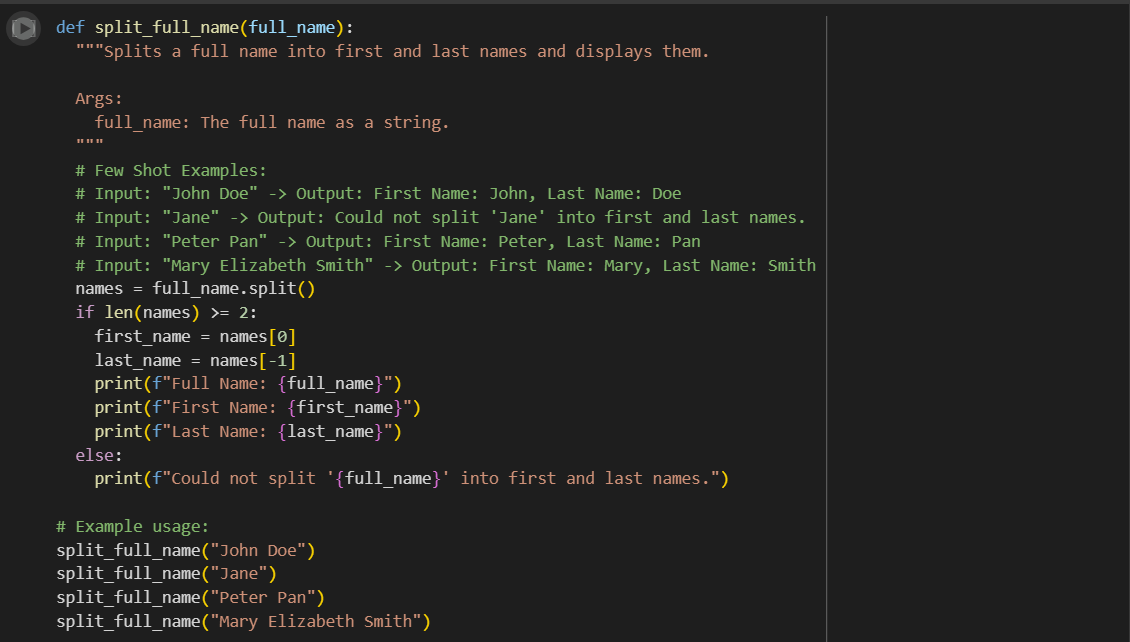
Task Description#3  
● Few-shot: Provide 2–3 examples to generate a function that formats full names as  
“Last, First”.

PROMPT 1: Write a function in python code which formats full names as last name and first name. For example 1)Bandi Sreesaicharan

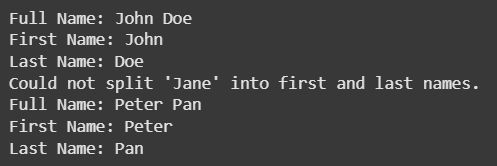
First name: Sreesaicharan Last name: Bandi

PROMPT 2: add few shots as comments

CODE:



OUTPUT:



EXPLANATION:

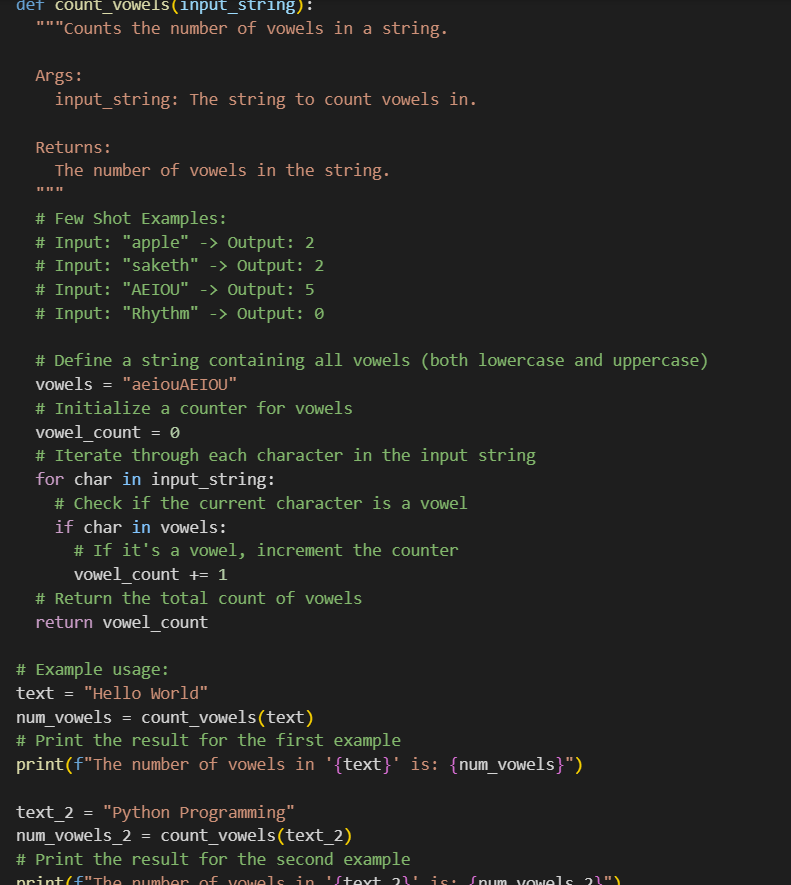
This code defines a Python function called split\_full\_name that attempts to split a full name string into a first and last name.

* It takes one argument, full\_name, which is the string containing the name.
* Inside the function, it uses the split() method to split the full\_name string into a list of words based on spaces and stores it in the names variable.
* It then checks if the names list has at least two elements.
  + If it does, it assumes the first element is the first name (names[0]) and the last element is the last name (names[-1]) and prints both along with the full name.
  + If the list has less than two elements (meaning it couldn't be split into at least two parts), it prints a message indicating that it could not split the name.
* The example usage demonstrates how to call the function with different names, including a single name and a name with more than two parts.

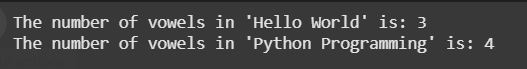
Task Description#4  
● Compare zero-shot and few-shot prompts for writing a function that counts the  
number of vowels in a string.  
PROMPT 1: Write a function in python which counts the number of vowels in string and display it. For example 1)bandi = 2

PROMPT 2: add few shots as comments

CODE:



OUTPUT:



EXPLANATION:

This code defines a Python function called count\_vowels that counts the number of vowels in a given string.

Here's a breakdown of the code:

1. **Function Definition**: def count\_vowels(input\_string): defines a function named count\_vowels that takes one argument, input\_string.
2. **Docstring**: The triple-quoted string explains what the function does, its arguments, and what it returns. It also includes "Few Shot Examples" which are pairs of inputs and expected outputs to illustrate the function's behavior.
3. **Vowel Definition**: vowels = "aeiouAEIOU" creates a string containing all lowercase and uppercase vowels. This string is used to check if a character is a vowel.
4. **Vowel Counter Initialization**: vowel\_count = 0 initializes a variable to keep track of the number of vowels found.
5. **Iterating Through the String**: for char in input\_string: starts a loop that goes through each character in the input\_string.
6. **Checking for Vowels**: if char in vowels: checks if the current character (char) is present in the vowels string.
7. **Incrementing the Counter**: vowel\_count += 1 increments the vowel\_count by 1 if the character is a vowel.
8. **Returning the Count**: return vowel\_count returns the final count of vowels.
9. **Example Usage**: The code then shows how to use the count\_vowels function with two different strings ("Hello World" and "Python Programming") and prints the results.

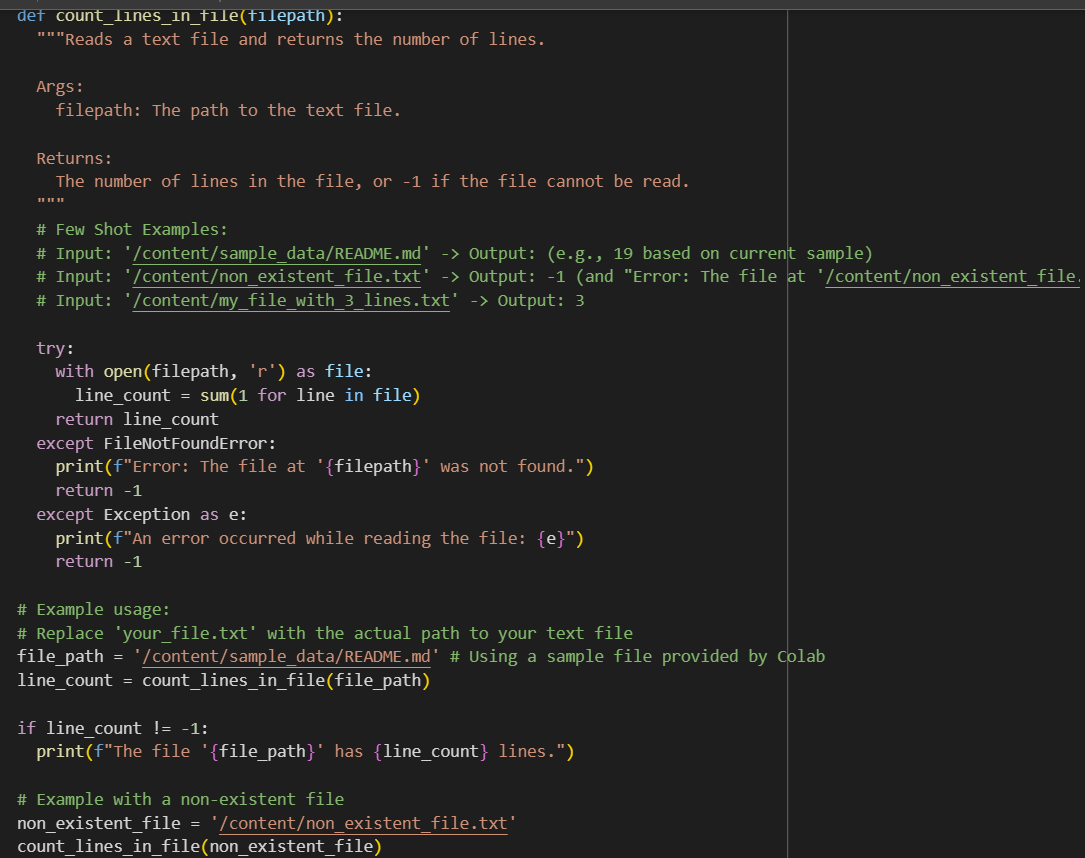
In essence, the function iterates through each character of the input string and increments a counter whenever it encounters a character that is a vowel (either lowercase or uppercase).

Task Description#5  
● Use few-shot prompting to generate a function that reads a .txt file and returns the  
number of lines.

PROMPT 1: write a function in python code which reads a .txt file and returns number of lines and display it

PROMPT 2: add

CODE:



OUTPUT:

This code defines a Python function called count\_lines\_in\_file that counts the number of lines in a text file.

Here's a breakdown of the code:

1. **Function Definition**: def count\_lines\_in\_file(filepath): defines a function named count\_lines\_in\_file that takes one argument, filepath, which is the path to the file.
2. **Docstring**: The triple-quoted string explains what the function does, its arguments, and what it returns. It also includes "Few Shot Examples" to illustrate the function's behavior with existing and non-existent files.
3. **Error Handling**: The code uses a try...except block to handle potential errors that might occur when trying to read the file.
4. **Opening and Reading the File**:
   * with open(filepath, 'r') as file: opens the file specified by filepath in read mode ('r'). The with statement ensures that the file is automatically closed even if errors occur.
   * line\_count = sum(1 for line in file) reads the file line by line and counts the number of lines using a generator expression and the sum() function.
5. **Returning the Line Count**: return line\_count returns the total number of lines found in the file.
6. **Handling FileNotFoundError**: except FileNotFoundError: catches the specific error that occurs if the file does not exist. It prints an error message and returns -1 to indicate that the file was not found.
7. **Handling Other Exceptions**: except Exception as e: catches any other exceptions that might occur during file processing. It prints a generic error message including the exception details and returns -1.
8. **Example Usage**: The code demonstrates how to use the function with an existing sample file provided by Colab ([/content/sample\_data/README.md](https://colab.research.google.com/drive/1-N98RuABjXmHPqGkmkMPlXKOng236_mq)) and a non-existent file ([/content/non\_existent\_file.txt](https://colab.research.google.com/drive/1-N98RuABjXmHPqGkmkMPlXKOng236_mq)) to show both successful and error handling cases.

In summary, this function provides a safe way to count the lines in a file, handling cases where the file might not exist or other reading errors occur.