# LAB Assignment (4.3)

Name:M.Rohan

H.NO :2403A51545

Batch:08

Task Description#1

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

```
def is_leap(year):
  """
  Checks if a given year is a leap year.

  Args:
    year: An integer representing the year.

  Returns:
    True if the year is a leap year, False otherwise.
  """
  return (year % 4 == 0 and year % 100 != 0) or (year % 400 == 0)

# Example usage:
print(f"Is 2000 a leap year? {is_leap(2000)}")
print(f"Is 1900 a leap year? {is_leap(1900)}")
print(f"Is 2024 a leap year? {is_leap(2024)}")
print(f"Is 2023 a leap year? {is_leap(2023)}")
```

Observation:

1. A leap year occurs every 4 years to keep our calendar aligned with Earth's revolutions around the sun.
2. The function checks if a year is divisible by 4, but not by 100—unless it's also divisible by 400.
3. This logic ensures that century years like 1900 are not leap years, while 2000 is.

4.The function returns True for leap years and False otherwise, making it easy to use in programs.

5.It’s a handy tool for date calculations, calendar apps.

## Task Description#2

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

```
def cm_to_inches(cm):
  """Converts centimeters to inches."""
  # One-shot example: 10 cm is approximately 3.937 inches
  # Conversion factor: 1 inch = 2.54 cm
  inches = cm / 2.54
  return inches

# Example usage
centimeters = 20
inches = cm_to_inches(centimeters)
print(f"{centimeters} cm is equal to {inches:.2f} inches")
```

```
20 cm is equal to 7.87 inches
```

## Observation:

1. This function helps convert lengths from centimeters to inches using a fixed conversion factor.

2. Since 1 centimeter equals approximately 0.3937 inches, the function multiplies the input by this value.

3. It's useful in applications where measurements need to be displayed in imperial units.

4. The function ensures accurate and consistent conversion for any numeric input.

5. It's a simple yet essential tool for international design, tailoring, or engineering tasks.

## Task Description#3

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

```python
def format_name(full_name):
  """
  Formats a full name as "Last, First".

  Examples:
  - "John Doe" -> "Doe, John"
  - "Alice Wonderland" -> "Wonderland, Alice"
  - "Peter Pan" -> "Pan, Peter"
  """
  name_parts = full_name.split()
  if len(name_parts) > 1:
    # Assume the last part is the last name and the first is the first name
    first_name = name_parts[0]
    last_name = name_parts[-1]
    return f"{last_name}, {first_name}"
  else:
    # If only one part, return it as is (or handle as an error)
    return full_name

# Example usage with the provided examples
print(format_name("John Doe"))
print(format_name("Alice Wonderland"))
print(format_name("Peter Pan"))
```

OUTPUT:

```
Doe, John
Wonderland, Alice
Pan, Peter
```

Observation:

1. This function is designed to reformat names from the standard "First Last" structure into "Last, First", which is commonly used in formal documents, directories, and citations.

2. It works by splitting the input string into two parts and rearranging them using string formatting. The few-shot examples help illustrate the pattern clearly, making it easy for both humans and models to understand the transformation.

3. It's a simple yet effective way to standardize name formats across systems.

4.You could easily extend it to handle middle names or initials with a bit more logic.

## Task Description#4

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

Observation:

- **Generate zero-shot prompt and code**: Create a code cell with a zero-shot prompt and the resulting Python function to count vowels.
- **Generate few-shot prompt and code**: Create a code cell with a few-shot prompt (including examples) and the resulting Python function to count vowels.
- **Compare approaches**: Explain the differences between the zero-shot and few-shot approaches in terms of the prompt structure and how they might influence the generated code.

## Task Description#5

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

```
# Create a dummy text file for demonstration
with open("sample.txt", "w") as f:
  f.write("This is line 1.\n")
  f.write("This is line 2.\n")
  f.write("This is line 3.\n")
```

Gemini

```
def count_lines_in_file(filepath):
  """
  Reads a text file and returns the number of lines.

  Examples:
  - Given a file named 'example1.txt' with content:
    Hello
    World
    The function should return 2.
  - Given a file named 'example2.txt' with content:
    Line A
    Line B
    Line C
    The function should return 3.
  """
  try:
    with open(filepath, 'r') as f:
      lines = f.readlines()
      return len(lines)
  except FileNotFoundError:
    return "Error: File not found."

# Example usage with the dummy file
num_lines = count_lines_in_file("sample.txt")
print(f"The number of lines in sample.txt is: {num_lines}")
```

```
The number of lines in sample.txt is: Error: File not found.
```

Observation:

**1.**This function opens a .txt file in read mode and uses a generator expression to count each line efficiently.

2.The few-shot examples help establish the expected input-output  making it easier for a model—or even a beginner—to infer the logic.

3. It's a clean and memory-friendly approach, especially useful for large files.

4. You can adapt it to count only non-empty lines or lines matching a pattern if needed.