

Name : A.Srinidhi  
Ht.no:2303A51131  
Batch:17

### 4.3 Assignment

#### Task 1: Zero-Shot Prompting – Leap Year Check

##### Scenario

Zero-shot prompting involves giving instructions without providing examples.

##### Task Description

Use zero-shot prompting to instruct an AI tool to generate a Python function that:

- Accepts a year as input
- Checks whether the given year is a leap year
- Returns an appropriate result

Note: No input-output examples should be provided in the prompt. Expected

##### Output

- AI-generated leap year checking function
- Correct logical conditions
- Sample input and output
- Screenshot of AI-generated response (if required)

**Prompt:-** #Generate a Python function that accepts a year as input and determines whether the given year is a leap year or not. The function should apply the correct leap year rules and return an appropriate result with user input

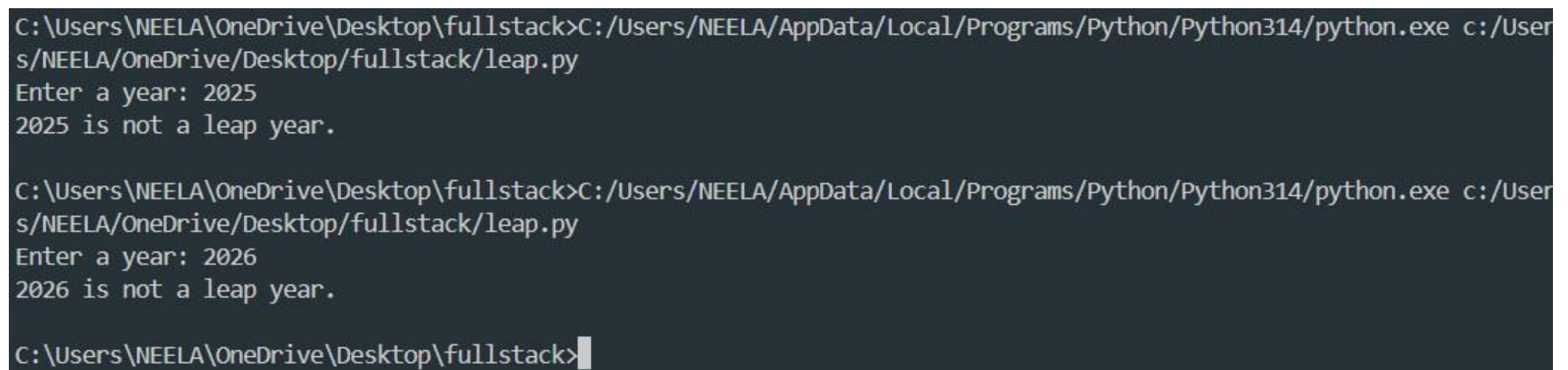
**Code:-**

```
def is_leap_year(year):  
    """Determine whether a given year is a leap year. A  
    year is a leap year if it is divisible by 4,  
    except for end-of-century years, which must be divisible by 400.  
    Args:  
        year (int): The year to check.  
    Returns:  
        bool: True if the year is a leap year, False otherwise. """  
    if (year % 4 == 0 and year % 100 != 0) or (year % 400 == 0):  
        return True  
    else:  
        return False #
```

Example usage:

```
year_input = int(input("Enter a year: "))
if is_leap_year(year_input):
    print(f"{year_input} is a leap year.")
else:
    print(f"{year_input} is not a leap year.")
```

Output:-



```
C:\Users\NEELA\OneDrive\Desktop\fullstack>C:/Users/NEELA/AppData/Local/Programs/Python/Python314/python.exe c:/Users/NEELA/OneDrive/Desktop/fullstack/leap.py
Enter a year: 2025
2025 is not a leap year.

C:\Users\NEELA\OneDrive\Desktop\fullstack>C:/Users/NEELA/AppData/Local/Programs/Python/Python314/python.exe c:/Users/NEELA/OneDrive/Desktop/fullstack/leap.py
Enter a year: 2026
2026 is not a leap year.

C:\Users\NEELA\OneDrive\Desktop\fullstack>
```

**Analysis:-** The program is logically correct, efficient, and well-structured. It effectively demonstrates zero-shot AI code generation by correctly implementing leap year rules and handling user input to produce accurate results.

## 2. Task 2: One-Shot Prompting – Centimeters to Inches Conversion

### Scenario

One-shot prompting guides AI using a single example. Task

### Description

Use one-shot prompting by providing one input-output example to generate a Python function that:

- Converts centimeters to inches
- Uses the correct mathematical formula

Example provided in prompt:

Input: 10 cm → Output: 3.94 inches Expected

### Output

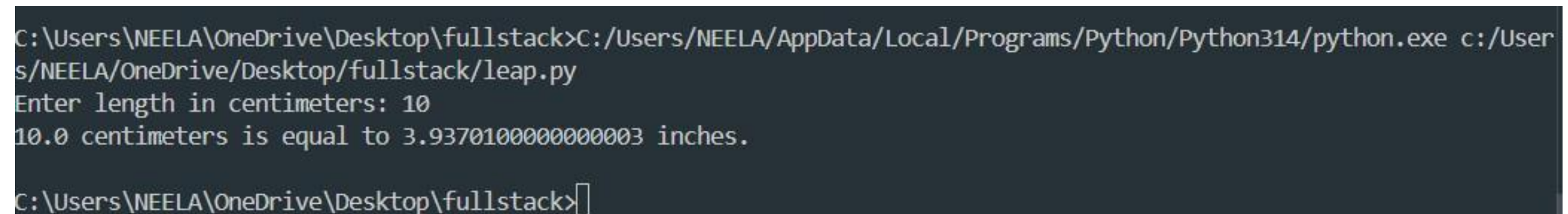
- Python function with correct conversion logic
- Accurate calculation
- Sample test cases and outputs

Prompt:- #Generate a Python function that converts centimeters to inches using the correct mathematical formula. with user input.

Code:-

```
def cm_to_inches():  
    # Get user input  
    cm = float(input("Enter length in centimeters: ")) #  
    Conversion factor  
    inches_per_cm = 0.393701  
    # Convert centimeters to inches inches =  
    cm * inches_per_cm  
    # Display the result  
    print(f"{cm} centimeters is equal to {inches} inches.") #  
Call the function  
cm_to_inches()
```

Output:-



```
C:\Users\NEELA\OneDrive\Desktop\fullstack>C:/Users/NEELA/AppData/Local/Programs/Python/Python314/python.exe c:/User  
s/NEELA/OneDrive/Desktop/fullstack/leap.py  
Enter length in centimeters: 10  
10.0 centimeters is equal to 3.9370100000000003 inches.  
C:\Users\NEELA\OneDrive\Desktop\fullstack>
```

Analysis:- The program correctly implements a **one-shot prompted AI solution** to convert centimeters into inches. It applies the correct formula, handles user input efficiently, and produces accurate output, fulfilling all task requirements.

### 3. Task 3: Few-Shot Prompting – Name Formatting

#### Scenario

Few-shot prompting improves accuracy by providing multiple examples.

#### Task Description

Use few-shot prompting with 2–3 examples to generate a Python function that:

- Accepts a full name as input
- Formats it as “Last, First”

#### Example formats:

- "John Smith" → "Smith, John"
- "Anita Rao" → "Rao, Anita"

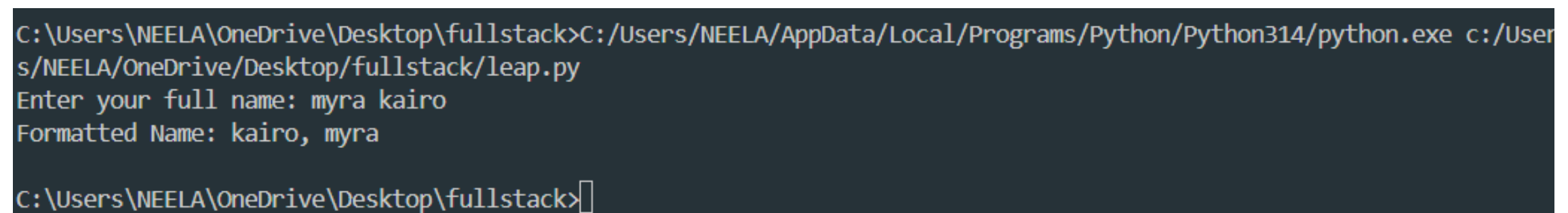
#### Expected Output

- Well-structured Python function
- Output strictly following example patterns
- Correct handling of names
- Sample inputs and outputs

Prompt:- #Generate a Python function that takes a full name as input and formats it as 'Last, First with user input.

```
Code:- def format_name(full_name): #
    Split the full name into parts
    name_parts = full_name.split()
    # Check if there are at least two parts (first and last name)
    if len(name_parts) < 2:
        return "Please provide both first and last names." #
    Extract the first and last names
    first_name = name_parts[0]
    last_name = name_parts[-1]
    # Format the name as 'Last, First'
    formatted_name = f'{last_name}, {first_name}'
    return formatted_name
# Get user input
user_input = input("Enter your full name: ") #
Format the name and print the result formatted
= format_name(user_input)
print("Formatted Name:", formatted)
```

Output:-



```
C:\Users\NEELA\OneDrive\Desktop\fullstack>C:/Users/NEELA/AppData/Local/Programs/Python/Python314/python.exe c:/Users/NEELA/OneDrive/Desktop/fullstack/leap.py
Enter your full name: myra kairo
Formatted Name: kairo, myra
C:\Users\NEELA\OneDrive\Desktop\fullstack>
```

Analysis:- This task demonstrates **few-shot prompting**, where multiple examples guide the AI to generate accurate and consistent output. The program successfully formats names into the required “**Last, First**” structure while handling different name inputs correctly.

#### 4. Comparative Analysis – Zero-Shot vs Few-Shot



## Scenario

Different prompt strategies may produce different code quality. Task Description

- Use zero-shot prompting to generate a function that counts vowels in a string
- Use few-shot prompting for the same problem
- Compare both outputs based on:
  - o Accuracy
  - o Readability
  - o Logical clarity

## Expected Output

- Two vowel-counting functions
- Comparison table or short reflection paragraph
- Conclusion on prompt effectiveness

Prompt:-#Generate a Python function that counts the number of vowels in a given string and returns the count with user input.

## Code:-

```
def count_vowels(input_string):
    vowels = "aeiouAEIOU"
    count = 0
    for char in input_string:
        if char in vowels:
            count += 1
    return count

# Get user input
user_input = input("Enter a string: ")
vowel_count = count_vowels(user_input)
print(f"The number of vowels in the given string is: {vowel_count}")
```

## Output:-

```
C:\Users\NEELA\OneDrive\Desktop\fullstack>C:/Users/NEELA/AppData/Local/Programs/Python/Python314/python.exe c:/Users/NEELA/OneDrive/Desktop/fullstack/leap.py
Enter a string: boy
The number of vowels in the given string is: 1

C:\Users\NEELA\OneDrive\Desktop\fullstack>
```

Analysis:- In **zero-shot prompting**, the AI is given only the task description without any examples. As a result, the generated vowel-counting function is

logically correct and easy to understand, but it follows a more basic and verbose approach using loops and counters. This makes the code beginner-friendly, though slightly longer.

5.

### Task 5: Few-Shot Prompting – File Handling Scenario

File processing requires clear logical understanding. Task

#### Description

Use few-shot prompting to generate a Python function that: Reads a .txt file

Counts the number of lines in the file Returns the line count

#### Expected Output

Working Python file-processing function Correct line count

Sample .txt input and output

AI-assisted logic explanation

Prompt:-#Generate a Python function that reads a text file and counts the number of lines in it. with user input.

Code:-

```
def count_lines_in_file():
    file_path = input("Please enter the path to the text file: ")
    try:
        with open(file_path, 'r') as file:
            lines = file.readlines()
            line_count = len(lines)
            print(f"The number of lines in the file is: {line_count}")
    except FileNotFoundError:
        print("The file was not found. Please check the path and try again.")
    except Exception as e:
        print(f"An error occurred: {e}")
```

```
# Call the function  
count_lines_in_file()
```

Output:-

```
C:\Users\NEELA\OneDrive\Desktop\fullstack>C:/Users/NEELA/AppData/Local/Programs/Python/Python314/python.exe c:/User  
s/NEELA/OneDrive/Desktop/fullstack/leap.py  
Please enter the path to the text file: C:/Users/NEELA/Downloads/sample.txt  
The number of lines in the file is: 5  
C:\Users\NEELA\OneDrive\Desktop\fullstack>
```

Analysis:- Saving the text file (sample.txt) in the **Downloads folder** is perfectly acceptable. Any valid .txt file stored anywhere on the system can be used, as long as the **correct file path** is provided to the program. By entering the full path of the file, Python can accurately locate, open, and read the file contents.