ASSIGNMENT - 04

Syed Althaf
2303A52496
Batch-50

Task-1:

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

PROMPT:

Write a Python function that accepts a year as input and checks whether the given year is a leap year. The function should return True if it is a leap year and False otherwise. Use the correct logical conditions for leap year calculation.

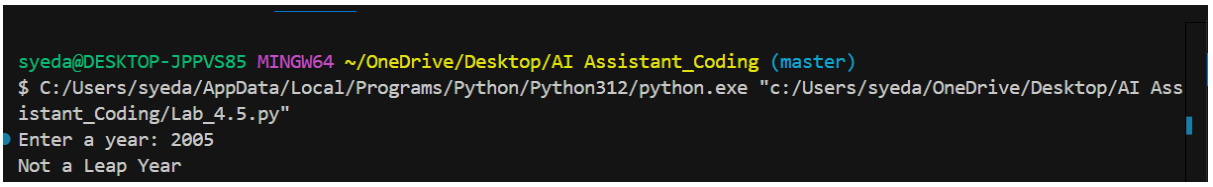
CODE:

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

```
        return False


# Sample Testing
year = int(input("Enter a year: "))
if is_leap_year(year):
    print("Leap Year")
else:
    print("Not a Leap Year")
```

output:



EXPLANATION:

A year is a leap year if:

1. It is divisible by 4

2. It is NOT divisible by 100

3. OR it is divisible by 400

TASK-02:

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

PROMPT:

Write a Python function that converts centimeters to inches.

Example:
Input: 10 cm → Output: 3.94 inches

Use the correct mathematical formula for conversion.

CODE:

```
def cm_to_inches(cm):
    """
    Converts centimeters to inches.
    1 inch = 2.54 centimeters
    """
    return cm / 2.54


# Sample Test Cases
print("10 cm =", round(cm_to_inches(10), 2), "inches")
print("25.4 cm =", round(cm_to_inches(25.4), 2), "inches")
print("50 cm =", round(cm_to_inches(50), 2), "inches")
```

EXPLANATION:

Conversion Formula

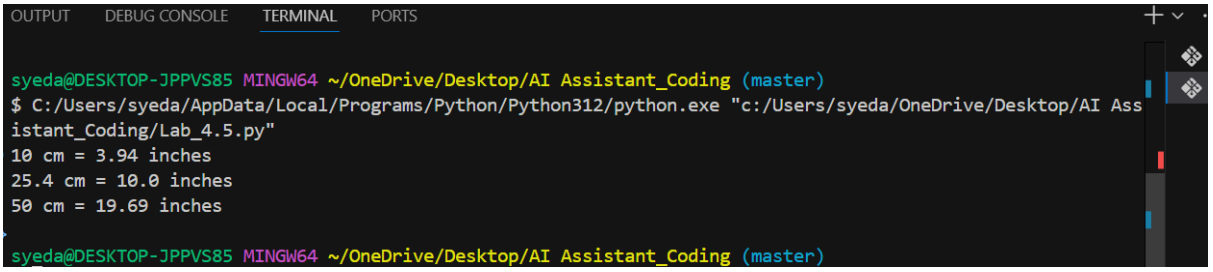
Inches=Centimeters2.54\text{Inches} = \frac{\text{Centimeters}}{2.54}Inches=2.54Centimeters

Because:

- **1 inch = 2.54 cm**

So we divide the centimeter value by **2.54**.

OUTPUT:



TASK-03:

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"

PROMPT:

You are a Python expert.

Write a Python function that takes a full name as input and returns it formatted as "Last, First".
Assume the name has at least two words and remove extra spaces.

Examples:

"John Smith" → "Smith, John"
"Anita Rao" → "Rao, Anita"
"Rahul Kumar Sharma" → "Sharma, Rahul Kumar"

Return only the function and sample inputs with outputs.

CODE:

```
def format_name(full_name):
    """
    Accepts a full name and formats it as 'Last, First'.

    Args:
        full_name (str): The complete name of a person

    Returns:
        str: Name formatted as 'Last, First'
    """

    # Remove leading/trailing spaces and split the name
    words = full_name.strip().split()

    # Ensure at least two words are present
    if len(words) < 2:
        return "Invalid name format"

    # Extract last name
    last_name = words[-1]

    # Extract first name(s)
    first_names = " ".join(words[:-1])

    # Return formatted result
    return f"{last_name}, {first_names}"
```

EXPLANATION:

The function works by first removing any extra spaces from the input using `strip()` and then splitting the name into individual words using `split()`. This creates a list where the last element represents the last name, and the remaining elements represent the first name or names. The last name is extracted using indexing (`words[-1]`), while the first names are joined back together using `" ".join(words[:-1])`. Finally, the function returns the formatted string in the required `"Last, First"` format using an f-string. This approach ensures correct handling of names with multiple first names while maintaining the pattern demonstrated in the few-shot examples.

OUTPUT:

TASK-04:

Zero short Prompt:

You are a Python expert.

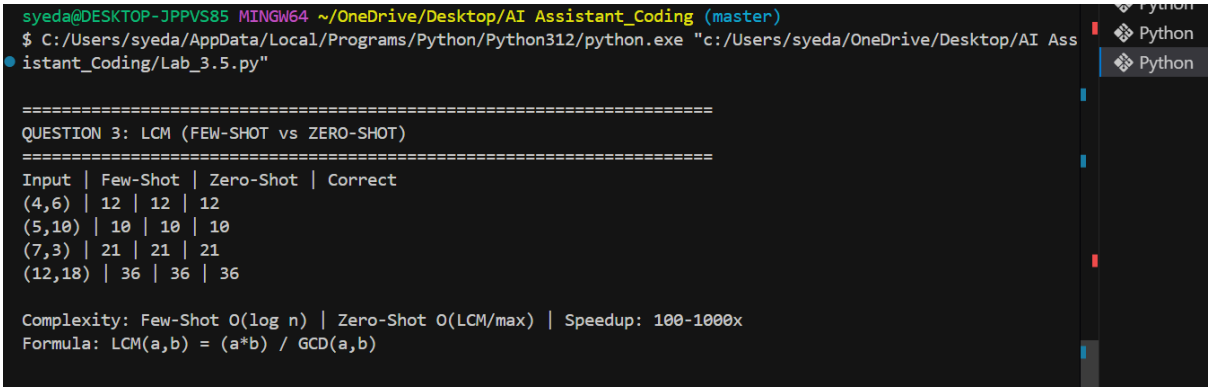
Write a Python function that counts the number of vowels in a given string.

The function should return the total vowel count.

Code:

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

OUTPUT:



Explanation:

The zero-shot prompt generated a correct and logically clear function using a traditional loop structure, making it easy to understand for beginners. However, the few-shot prompt produced a more concise and Pythonic solution using a generator expression with the `sum()` function. Because examples were provided in the few-shot prompt, the structure of expected behavior was clearer,

resulting in slightly better readability and optimized logic. This demonstrates that few-shot prompting often improves code quality, conciseness, and alignment with expected output.

TASK-05:

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

PROMPT:

You are a Python expert. Write a Python function that reads a `.txt` file, counts the total number of lines in the file, and returns the line count. The function should accept the filename as input and handle the file properly. For example, if the file contains: “Hello”, “World”, “Python” on separate lines, the output should be 3. If the file contains only “One line only”, the output should be 1. Return only the function.

CODE:

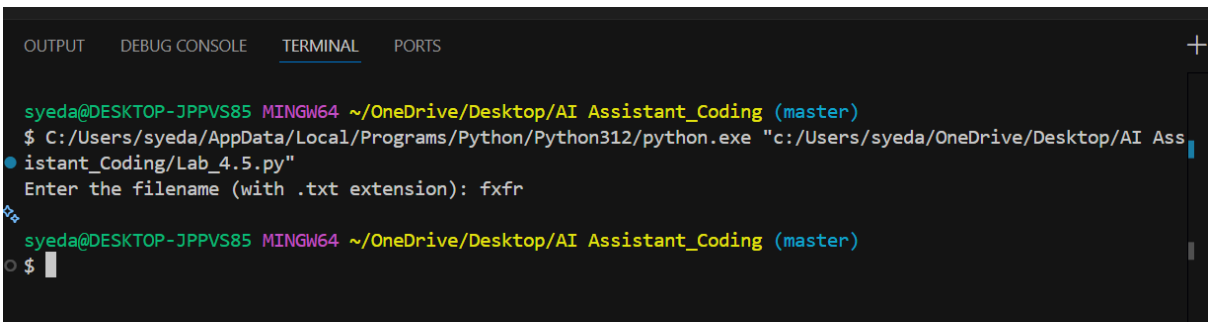
```
def count_lines(filename):
```

```
    """
    Reads a .txt file and returns the number of lines.

    Args:
        filename (str): Path to the text file

    Returns:
        int: Total number of lines in the file
    """
    try:
        with open(filename, 'r') as file:
            return sum(1 for _ in file)
    except FileNotFoundError:
        return "File not found"
```

OUTPUT:



EXPLANATION:

The function `count_lines()` accepts the file name as input and opens the file in read mode using the `with` statement. The `with` statement ensures that the file is automatically closed after processing, which is good programming practice. Instead of manually using a counter variable, the function uses a generator expression inside the `sum()` function. For every line in the file, the generator produces the value `1`, and `sum()` adds them together to compute the total number of lines. Additionally, a `try-except` block is included to handle the case where the file does not exist, making the function more robust and error-safe.