

Name – Narige Rohith

Roll – 2403a54101

Batch – 04 (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:

```
def is_leap_year(year):
    """Checks if a given year is a leap year and displays the result.

    Args:
        year: The year to check (an integer).
    """
    if (year % 4 == 0 and year % 100 != 0) or (year % 400 == 0):
        print(f"{year} is a leap year.")
    else:
        print(f"{year} is not a leap year.")

# Example usage:
is_leap_year(2024)
is_leap_year(1900)
is_leap_year(2000)
is_leap_year(2023)
```

Output:

```
2024 is a leap year.
1900 is not a leap year.
2000 is a leap year.
2023 is not a leap year.
```

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:



```
def cm_to_m(cm):
    """Converts a measurement in centimeters to meters and displays the result.

    Args:
        cm: The measurement in centimeters (a number).

    """
    # One Shot Example:
    # Input: 250 -> Output: 2.5 meters

    meters = cm / 100
    print(f"{cm} centimeters is equal to {meters} meters.")

    # Example usage:
    cm_to_m(250)
    cm_to_m(100)
    cm_to_m(50)
```

OUTPUT:

```
250 centimeters is equal to 2.5 meters.
100 centimeters is equal to 1.0 meters.
50 centimeters is equal to 0.5 meters.
```

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:



```
def split_full_name(full_name):
    """Splits a full name into first and last names and displays them.

    Args:
        full_name: The full name as a string.
    """

    # Few Shot Examples:
    # Input: "John Doe" -> Output: First Name: John, Last Name: Doe
    # Input: "Jane" -> Output: Could not split 'Jane' into first and last names.
    # Input: "Peter Pan" -> Output: First Name: Peter, Last Name: Pan
    # Input: "Mary Elizabeth Smith" -> Output: First Name: Mary, Last Name: Smith
    names = full_name.split()
    if len(names) >= 2:
        first_name = names[0]
        last_name = names[-1]
        print(f"Full Name: {full_name}")
        print(f"First Name: {first_name}")
        print(f"Last Name: {last_name}")
    else:
        print(f"Could not split '{full_name}' into first and last names.")

    # Example usage:
    split_full_name("John Doe")
    split_full_name("Jane")
    split_full_name("Peter Pan")
    split_full_name("Mary Elizabeth Smith")
```

OUTPUT:

```
Full Name: John Doe
First Name: John
Last Name: Doe
Could not split 'Jane' into first and last names.
Full Name: Peter Pan
First Name: Peter
Last Name: Pan
```

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:

```

def count_vowels(input_string):
    """Counts the number of vowels in a string.

    Args:
        input_string: The string to count vowels in.

    Returns:
        The number of vowels in the string.
    """

    # Few Shot Examples:
    # Input: "apple" -> Output: 2
    # Input: "saketh" -> Output: 2
    # Input: "AEIOU" -> Output: 5
    # Input: "Rhythm" -> Output: 0

    # Define a string containing all vowels (both lowercase and uppercase)
    vowels = "aeiouAEIOU"
    # Initialize a counter for vowels
    vowel_count = 0
    # Iterate through each character in the input string
    for char in input_string:
        # Check if the current character is a vowel
        if char in vowels:
            # If it's a vowel, increment the counter
            vowel_count += 1
    # Return the total count of vowels
    return vowel_count

# Example usage:
text = "Hello World"
num_vowels = count_vowels(text)
# Print the result for the first example
print(f"The number of vowels in '{text}' is: {num_vowels}")

text_2 = "Python Programming"
num_vowels_2 = count_vowels(text_2)
# Print the result for the second example
print(f"The number of vowels in '{text_2}' is: {num_vowels_2}")

```

OUTPUT:

```

The number of vowels in 'Hello World' is: 3
The number of vowels in 'Python Programming' is: 4

```

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:

```

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

    Args:
        filepath: The path to the text file.

    Returns:
        The number of lines in the file, or -1 if the file cannot be read.
    """
    # Few Shot Examples:
    # Input: '/content/sample_data/README.md' -> Output: (e.g., 19 based on current sample)
    # Input: '/content/non_existent_file.txt' -> Output: -1 (and "Error: The file at '/content/non_e
    # Input: '/content/my_file_with_3_lines.txt' -> Output: 3

    try:
        with open(filepath, 'r') as file:
            line_count = sum(1 for line in file)
        return line_count
    except FileNotFoundError:
        print(f"Error: The file at '{filepath}' was not found.")
        return -1
    except Exception as e:
        print(f"An error occurred while reading the file: {e}")
        return -1

    # Example usage:
    # Replace 'your_file.txt' with the actual path to your text file
    file_path = '/content/sample_data/README.md' # Using a sample file provided by Colab
    line_count = count_lines_in_file(file_path)

    if line_count != -1:
        print(f"The file '{file_path}' has {line_count} lines.")

    # Example with a non-existent file
    non_existent_file = '/content/non_existent_file.txt'
    count_lines_in_file(non_existent_file)

```

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:

- 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:**

- o 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.
- o 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](#)) and a non-existent file ([/content/non_existent_file.txt](#)) 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.