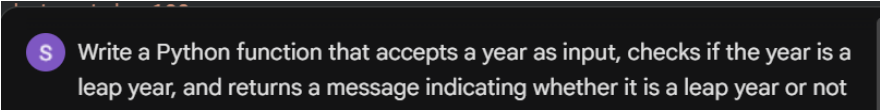


1	<div data-bbox="386 220 748 422"> <p>Gangula Navyasri B-25 Assignment - 4.3 2303A51739</p> </div> <div data-bbox="371 455 1266 483"> <p>Lab 4: Advanced Prompt Engineering – Zero-shot, One-shot, and Few-shot Techniques</p> </div> <div data-bbox="371 512 529 539"> <p>Lab Objectives</p> </div> <div data-bbox="407 543 1302 665"> <ul style="list-style-type: none"> • To explore and apply different levels of prompt examples in AI-assisted code generation • To understand how zero-shot, one-shot, and few-shot prompting affect AI output quality • To evaluate the impact of context richness and example quantity on AI performance • To build awareness of prompt strategy effectiveness for different problem types </div> <div data-bbox="371 695 589 720"> <p>Lab Outcomes (LOs)</p> </div> <div data-bbox="371 722 880 747"> <p>After completing this lab, students will be able to:</p> </div> <div data-bbox="407 751 1190 875"> <ul style="list-style-type: none"> • Use zero-shot prompting to instruct AI with minimal context • Use one-shot prompting with a single example to guide AI code generation • Apply few-shot prompting using multiple examples to improve AI responses • Compare AI outputs across different prompting strategies </div> <hr/> <div data-bbox="371 905 880 930"> <p>Task 1: Zero-Shot Prompting – Leap Year Check</p> </div> <div data-bbox="371 934 464 959"> <p>Scenario</p> </div> <div data-bbox="371 961 1122 987"> <p>Zero-shot prompting involves giving instructions without providing examples.</p> </div> <div data-bbox="371 989 550 1014"> <p>Task Description</p> </div> <div data-bbox="371 1016 1153 1041"> <p>Use zero-shot prompting to instruct an AI tool to generate a Python function that:</p> </div> <div data-bbox="407 1045 881 1136"> <ul style="list-style-type: none"> • Accepts a year as input • Checks whether the given year is a leap year • Returns an appropriate result </div> <div data-bbox="371 1138 1015 1163"> <p>Note: No input-output examples should be provided in the prompt.</p> </div> <div data-bbox="371 1167 550 1192"> <p>Expected Output</p> </div> <div data-bbox="407 1197 930 1346"> <ul style="list-style-type: none"> • AI-generated leap year checking function • Correct logical conditions • Sample input and output • Screenshot of AI-generated response (if required) <p>Prompt :</p> </div> <div data-bbox="407 1373 1282 1482">  <p>The screenshot shows a dark-themed interface with a purple circle containing a white 'S' icon. To the right of the icon, the text reads: "Write a Python function that accepts a year as input, checks if the year is a leap year, and returns a message indicating whether it is a leap year or not".</p> </div> <div data-bbox="407 1514 470 1539"> <p>Code :</p> </div>	Week 2 - Wednesday
---	--	--------------------

```

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

    A year is a leap year if:
    - It is divisible by 4, but not by 100
    - OR it is divisible by 400
    """
    if (year % 4 == 0 and year % 100 != 0) or (year % 400 == 0):
        return f"{year} is a leap year."
    else:
        return f"{year} is not a leap year."

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

```

Output :

```

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

```

Explanation : Checks wheather a year is leap or not by applying the rules: divisible by 4 but not 100, or divisible by 400. If the condition is true, it returns that the year is a leap year; otherwise, it states it is not leap year.

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

S Write a Python function to convert centimeters to inches using inches = cm / 2.54.

Code:

```

def cm_to_inches(cm):
    """
    Converts a given length in centimeters to inches.
    1 inch = 2.54 centimeters.
    """
    inches = cm / 2.54
    return inches

# Example usage:
length_cm = 10
length_inches = cm_to_inches(length_cm)
print(f"{length_cm} centimeters is equal to {length_inches:.2f} inches.")

length_cm = 25.4
length_inches = cm_to_inches(length_cm)
print(f"{length_cm} centimeters is equal to {length_inches:.2f} inches.")

```

Output :

```

10 centimeters is equal to 3.94 inches.
25.4 centimeters is equal to 10.00 inches.

```

Explanation : This function converts centimeters to inches using the formula $\text{inches} = \text{cm} / 2.54$. For example, 10 cm equals 3.94 inches and 25.4 cm equals 10 inches, with results shown to two decimal places

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 :

S Write a Python function that accepts a full name as input and formats it as "Last, First". Examples. Input: "John Smith" → Output: "Smith, John" Input: "Anita Rao" → Output: "Rao, Anita", Input: "Michael Jordan" → Output: "Jordan, Michael" The function should : Split the input into first and last name , Return the formatted string as "Last, First", Include sample test cases with outputs

Code :

```
[3] def format_name_last_first(full_name):
    """
    Formats a full name from 'First Last' to 'Last, First'.

    Args:
        full_name (str): The full name string (e.g., 'John Smith').

    Returns:
        str: The formatted name string (e.g., 'Smith, John').
        Returns 'Invalid Name' if the input doesn't contain a space.
    """
    name_parts = full_name.split(' ')
    if len(name_parts) == 2:
        first_name = name_parts[0]
        last_name = name_parts[1]
        return f"{last_name}, {first_name}"
    else:
        return "Invalid Name: Please provide a first and last name separated by a space."

# Sample test cases:
print(f"Input: 'John Smith' -> Output: '{format_name_last_first('John Smith')}'")
print(f"Input: 'Anita Rao' -> Output: '{format_name_last_first('Anita Rao')}'")
print(f"Input: 'Michael Jordan' -> Output: '{format_name_last_first('Michael Jordan')}'")
print(f"Input: 'SingleName' -> Output: '{format_name_last_first('SingleName')}'")
```

Output:

```
Input: 'John Smith' -> Output: 'Smith, John'
Input: 'Anita Rao' -> Output: 'Rao, Anita'
Input: 'Michael Jordan' -> Output: 'Jordan, Michael'
Input: 'SingleName' -> Output: 'Invalid Name: Please provide a first and last name separated by a space.'
Input: 'Three Part Name' -> Output: 'Invalid Name: Please provide a first and last name separated by a space.'
```

Explanation : This function reformats names from "First Last" to "Last, First." It splits the input by space, works only if there are exactly two parts, and returns an error message for single or multi-part names outside that format.

Task 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:
 - Accuracy
 - Readability
 - Logical clarity

Expected Output

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

S Write two Python functions to count vowels in a string: 1. Zero-shot: no examples, just instructions. 2. Few-shot: include examples like "hello → 2", "Python → 1". Then compare both outputs on accuracy, readability, and logical clarity. Provide both functions, a short comparison, and a conclusion.

Code :

```

def count_vowels_zero_shot(text):
    """
    Counts the number of vowels in a given string (case-insensitive).
    """
    vowels = "aeiouAEIOU"
    vowel_count = 0
    for char in text:
        if char in vowels:
            vowel_count += 1
    return vowel_count

# Example usage:
print(f'Hello World' has {count_vowels_zero_shot('Hello World')} vowels.")
print(f'Python Programming' has {count_vowels_zero_shot('Python Programming')} vowels.")
print(f'AEIOUaeiou' has {count_vowels_zero_shot('AEIOUaeiou')} vowels.")
print(f'Rhythm' has {count_vowels_zero_shot('Rhythm')} vowels.")

... 'Hello World' has 3 vowels.
'Python Programming' has 4 vowels.
'AEIOUaeiou' has 10 vowels.
'Rhythm' has 0 vowels.

```

```

def count_vowels_few_shot(text):
    """
    Counts the number of vowels in a given string (case-insensitive).
    Examples: 'hello' -> 2, 'Python' -> 1.
    """
    vowels = "aeiou"
    vowel_count = 0
    for char in text:
        if char.lower() in vowels:
            vowel_count += 1
    return vowel_count

# Example usage:
print(f'hello' has {count_vowels_few_shot('hello')} vowels. (Expected: 2)")
print(f'Python' has {count_vowels_few_shot('Python')} vowels. (Expected: 1)")
print(f'Artificial Intelligence' has {count_vowels_few_shot('Artificial Intelligence')} vowels.")
print(f'RHYTHM' has {count_vowels_few_shot('RHYTHM')} vowels.")
print(f'OpenAI' has {count_vowels_few_shot('OpenAI')} vowels.")

... 'hello' has 2 vowels. (Expected: 2)
'Python' has 1 vowels. (Expected: 1)
'Artificial Intelligence' has 10 vowels.
'RHYTHM' has 0 vowels.
'OpenAI' has 4 vowels.

```

Explanation : This function counts vowels in a string by checking each character against a set of vowels (both uppercase and lowercase). It increments a counter whenever a vowel is found and returns the total. For example, "Hello World" has 3 vowels, "Python Programming" has 4, "AEIOUaeiou" has 10, and "Rhythm" has none.

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

S Write a Python function to read a .txt file and count its lines. Examples: File content Line 1: Hello World Line 2: Python is fun , Output: 3 File content: Line 1: Data Science Line 2: Machine Learning Output: 2 The function should: Accept filename as input , Return line count - Show sample inputs/outputs , Briefly explain the logic

Code :

```
import os

def count_lines_in_file(filename):
    def count_lines_in_file(filename)
        f lines.
    Reads a text file and counts the number of lines.
    Args:
        filename (str): The path to the text file.
        file.
    Returns:
        int: The total number of lines in the file.
        file.
    """
    line_count = 0
    try:
        with open(filename, 'r') as file:
            for line in file:
                line_count += 1
    except FileNotFoundError:
        print(f"Error: File '{filename}' not found.")
        return -1 # Indicate an error
    return line_count
```

```
 # --- Sample Test Cases ---

# Sample 1: Create a temporary file for testing
file_content_1 = "Hello World\nPython is fun\nThird line"
file_name_1 = "sample_1.txt"
with open(file_name_1, 'w') as f:
    f.write(file_content_1)

print(f"File content '{file_name_1}':")
print(file_content_1)
line_count_1 = count_lines_in_file(file_name_1)
print(f"Output: {line_count_1} (Expected: 3)\n")

# Sample 2: Create another temporary file for testing
file_content_2 = "Data Science\nMachine Learning"
file_name_2 = "sample_2.txt"
with open(file_name_2, 'w') as f:
    f.write(file_content_2)

print(f"File content '{file_name_2}':")
print(file_content_2)
line_count_2 = count_lines_in_file(file_name_2)
print(f"Output: {line_count_2} (Expected: 2)\n")

# Clean up temporary files
os.remove(file_name_1)
os.remove(file_name_2)
```

```
File content 'sample_1.txt':
Hello World
Python is fun
Third line
Output: 3 (Expected: 3)
```

```
File content 'sample_2.txt':
Data Science
Machine Learning
Output: 2 (Expected: 2)
```

Explanation : This program defines a function that reads a text file and counts its lines by iterating through each line. It handles missing files with an error message and returns -1 in that case. The sample tests create temporary files, write content into them, count the lines (3 and 2 respectively), display the results, and then clean up by deleting the files.

--	--	--