NAME: MOHAMMED NIZAMUDDIN ROLL NO: 2503B05144 COURSE: MTECH(CSE)

Lab 4: Advanced Prompt Engineering – Zero-shot, One-shot, and Few-shot Techniques

LabObjectives:

- 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.

LabOutcomes(LOs):

After completing this lab, students will be able to:

- 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 the three prompting strategies

Task Description#1

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

Expected Output#1

- AI-generated function with no examples provided

Prompt-1: write a function that checks whether a given year is a leap year.

CODE:

```
# ...existing code...
def is_leap_year(year: int) -> bool:
    """
    Return True if `year` is a leap year using Gregorian rules:
    - divisible by 4, except years divisible by 100 unless also divisible by 400.2013
    """
    if not isinstance(year, int):
        raise TypeError("year must be an integer")
    return (year % 4 == 0) and (year % 100 != 0 or year % 400 == 0)


if __name__ == "__main__":
    import sys
    try:
        y = int(sys.argv[1]) if len(sys.argv) > 1 else int(input("Year: "))
    except Exception as e:
        print("Invalid year:", e)
        sys.exit(1)
    print(f"{y} is a leap year" if is_leap_year(y) else f"{y} is not a leap year")
# ...existing code...
```

OUTPUT:

```
PS C:\Users\moham\Desktop\Python> & "C:/Program Files/Python314/python.exe" c:/Users/moham/Desktop/Python/Code.py
Year: 2004
2004 is a leap year
PS C:\Users\moham\Desktop\Python> & "C:/Program Files/Python314/python.exe" c:/Users/moham/Desktop/Python/Code.py
Year: 2025
2025 is not a leap year
```

Task Description#2

● **One-shot**: Give one input-output example to guide AI in writing a function that converts centimeters to inches.

Expected Output#2

● Function with correct conversion logic

Prompt-1: Give a python code that converts centimeters to inches

Prompt-2: Take input from user

```python
# ...existing code...
def cm_to_inches(cm: float) -> float:
    """
    Convert centimeters to inches (1 inch = 2.54 cm) and print a conversion log.

    One-shot example:
    input: 30
    printed output: Converted 30.0 cm to 11.8110 inches
    """
    if not isinstance(cm, (int, float)):
        raise TypeError("cm must be a number")
    inches = cm / 2.54
    print(f"Converted {float(cm)} cm to {inches:.4f} inches")
    return inches
# ...existing code...

if __name__ == "__main__":
    import sys
    try:
        # Accept cm from command-line or prompt the user
        cm = float(sys.argv[1]) if len(sys.argv) > 1 else float(input("Centimeters: "))
    except Exception as e:
        print("Invalid input:", e)
        sys.exit(1)

    cm_to_inches(cm)
```

OUTPUT:

```
PS C:\Users\moham\Desktop\Python> & "C:/Program Files/Python314/python.exe" c:/Users/moham/Desktop/Python/Code.py
Centimeters: 30
Converted 30.0 cm to 11.8110 inches
PS C:\Users\moham\Desktop\Python> & "C:/Program Files/Python314/python.exe" c:/Users/moham/Desktop/Python/Code.py
Centimeters: 1
Converted 1.0 cm to 0.3937 inches
```

Task Description#3

● Few-shot: Provide 2–3 examples to generate a function that formats full names as "Last, First**".**

Expected Output#3

● Well-structured function respecting the examples

Prompt-1: to generate a function that formats full names as "Last, First**".**

Prompt-2: Modify the given function to accept input from the user.

CODE:

```python
def format_name(full_name: str) -> str:
    if not isinstance(full_name, str):
        raise TypeError("full_name must be a string")
    s = full_name.strip()
    if not s:
        return ""
    if "," in s:
        parts = [p.strip() for p in s.split(",", 1)]
        first = parts[1] if len(parts) > 1 else ""
        return f"{parts[0]}, {first}".strip(", ")
    tokens = [t for t in s.split() if t]
    if len(tokens) == 1:
        return tokens[0]
    last = tokens[-1]
    first = " ".join(tokens[:-1])
    return f"{last}, {first}"

if __name__ == "__main__":
    import sys
    try:
        name = input("Enter full name: ")
    except Exception:
        sys.exit(1)
    print(format_name(name))
```

OUTPUT:

```
PS C:\Users\moham\Desktop\Python> & "C:/Program Files/Python314/python.exe" c:/Users/moham/Desktop/Python/Code.py
Enter full name: Mohammed Nizamuddin
Nizamuddin, Mohammed
PS C:\Users\moham\Desktop\Python> & "C:/Program Files/Python314/python.exe" c:/Users/moham/Desktop/Python/Code.py
Enter full name: John Cena
Cena, John
```

Task Description#4

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

Expected Output#4

● Functional output and comparative reflection

Prompt-1: Compare zero-shot and few-shot prompts for writing a function that counts the number of vowels in a string**.**

CODE:

```python
# Zero-shot approach
def count_vowels_zero(text: str) -> int:
    """Count vowels in given string."""
    if not isinstance(text, str):
        raise TypeError("Input must be a string")
    return sum(1 for c in text.lower() if c in 'aeiou')

# Few-shot approach with examples
def count_vowels_few(text: str) -> int:
    """Count vowels in text.
    Examples:
    - "hello" → 2 (e,o)
    - "PYTHON" → 1 (o)
    - "sky" → 0
    """
    if not isinstance(text, str):
        raise TypeError("Input must be a string")
    return sum(c in 'aeiou' for c in text.lower())

if __name__ == "__main__":
    try:
        text = input("Enter text to count vowels: ")
        print(f"Zero-shot result: {count_vowels_zero(text)} vowels")
        print(f"Few-shot result: {count_vowels_few(text)} vowels")
    except Exception as e:
        print(f"Error: {e}")
```

OUTPUT:

```
PS C:\Users\moham\Desktop\Python> & "C:/Program Files/Python314/python.exe" c:/Users/moham/Desktop/Python/Code.py
Enter text to count vowels: Apple
Zero-shot result: 2 vowels
Few-shot result: 2 vowels
PS C:\Users\moham\Desktop\Python> & "C:/Program Files/Python314/python.exe" c:/Users/moham/Desktop/Python/Code.py
Enter text to count vowels: Banana
Zero-shot result: 3 vowels
Few-shot result: 3 vowels
```

Comparative Reflection

1. **Zero-shot vs Few-shot Differences:**

- Zero-shot implementations are more concise but lack context
- Few-shot versions include examples that clarify edge cases
- Both approaches produce identical results

2. **Documentation Quality:**
    - Few-shot docstrings are more helpful with concrete examples
    - Zero-shot documentation is minimal but sufficient
3. **Code Complexity:**
    - Implementation complexity is similar
    - Few-shot approach leads to more robust error handling due to example-driven development

Task Description#5
● Use few-shot prompting to generate a function that reads a .txt file and returns the number of lines.
Expected Output#5
● Working file-processing function with AI-guided logic

Prompt-1: generate a function that reads a .txt file and return number of lines

Prompt-2: Write a working file-processing Python function that reads a .txt file and returns the number of lines.

```python
def count_lines(filename: str) -> int:
    """
    Count the number of lines in a text file.

    Examples:
    - "empty.txt" (empty file) → 0 lines
    - "single.txt" (one line) → 1 line
    - "multiline.txt" (multiple lines with empty lines) → actual line count

    Returns:
    - int: Number of lines in file
    """
    try:
        with open(filename, 'r', encoding='utf-8') as file:
            return sum(1 for _ in file)
    except FileNotFoundError:
        raise FileNotFoundError(f"File '{filename}' not found")
    except Exception as e:
        raise Exception(f"Error reading file: {e}")

if __name__ == "__main__":
    try:
        filename = input("Enter the path to your text file: ")
        line_count = count_lines(filename)
        print(f"Number of lines in {filename}: {line_count}")
    except Exception as e:
        print(f"Error: {e}")
```

(variable) line_count: int

OUTPUT:

```
PS C:\Users\moham\Desktop\Python>
& "C:/Program Files/Python314/python.exe" c:/Users/moham/Desktop/Python/Code.py
Enter the path to your text file: C:\Users\moham\Desktop\Python\Test2.txt
Number of lines in C:\Users\moham\Desktop\Python\Test2.txt: 2
PS C:\Users\moham\Desktop\Python> & "C:/Program Files/Python314/python.exe" c:/Users/moham/Desktop/Python/Code.py
Enter the path to your text file: C:\Users\moham\Desktop\Python\Test1.txt
Number of lines in C:\Users\moham\Desktop\Python\Test1.txt : 0
```

How to Test

1. **Create sample text files in the same directory:**
   - Test1.txt (empty file)
   - Test2.txt (with some content)
2. **Run in VS Code Terminal:**

   python "c:/Users/moham/Desktop/Python/Code.py"

3. **Enter the file path when prompted:**

Enter the path to your text file: Test2.txt

Features:

- Error handling for missing files
- UTF-8 encoding support
- Memory efficient (streams file)
- Example-driven implementation
- User-friendly interface

Note: Report should be submitted a word document for all tasks in a single document with prompts, comments & code explanation, and output and if required, screenshots

Evaluation Criteria:

Criteria Max	**Marks**
Zero Shot (Task #1)	2.5
One Shot (Task#2)	2.5
Few Shot (Task#3 & Task #5)	2.5
Comparison (Task#4)	2.5
Total 10 Marks	