

# **AI Assisted Coding**

**Week2 – Wednesday**

**Name:** Lalu prasad Aroori

**Batch:** 07

**Roll No.:** 2303A51948

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

**Techniques**

**Lab Objectives**

- 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

Week2 -

Wednesday

**Lab Outcomes (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 different prompting strategies

## 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)

```
# Write a Python function that accepts a year as input and checks whether the given year is a leap year.
# The function should return an appropriate result.

def is_leap_year(year):
    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:
year_input = int(input("Enter a year: "))
result = is_leap_year(year_input)
print(result)

#generate remarks for Zero-Shot Prompting - Leap Year Check

# Remarks:
# This function accurately determines whether a given year is a leap year based on the established rules:
# 1. A year is a leap year if it is divisible by 4.
# 2. However, if the year is divisible by 100, it is not a leap year, unless it is also divisible by 400.
# 3. The function takes an integer input representing the year and returns a string indicating whether it is a leap year or not. The example usage demonstrates how to use the function by prompting the user for input
```

Enter a year: 2023  
2023 is not a leap year.  
PS D:\3\_2\_SEMAI\_assisted\_coding\Lab\_4\_3>

## 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

The screenshot shows a code editor interface with a Python file named "One-Shot Prompting - Centimeters to Inches Conversion.py". The code defines a function `cm\_to\_inches` that converts centimeters to inches using the formula `cm / 2.54`. It includes an example usage where the user is prompted for input and the output is displayed. The terminal below shows the execution of the script and its output for two different input values.

```
#Write a Python function to convert centimeters to inches.
#Example:
# Input: 10 cm → Output: 3.94 inches
#
def cm_to_inches(cm):
    inches = cm / 2.54
    return f"{cm} cm is equal to {inches:.2f} inches."
#
# Example usage:
cm_input = float(input("Enter length in centimeters: "))
result = cm_to_inches(cm_input)
print(result)
#
#generate remarks for One-Shot Prompting - Centimeters to Inches Conversion
#
# Remarks:
# This function converts a given length from centimeters to inches using the conversion factor of 1 inch = 2.54 cm.
# It takes a floating-point number as input representing the length in centimeters and returns a formatted string
# indicating the equivalent length in inches rounded to two decimal places. The example usage demonstrates how to
# use the function by prompting the user for input and displaying the result.

PS D:\3_2_SEM\AI_assisted_coding\Lab_4_3> & C:/Python314/python.exe "d:/3_2_SEM\AI_assisted_coding\Lab_4_3\One-Shot Prompting - Centimeters to Inches Conversion.py"
Enter length in centimeters: 10
10.0 cm is equal to 3.94 inches.
PS D:\3_2_SEM\AI_assisted_coding\Lab_4_3> & C:/Python314/python.exe "d:/3_2_SEM\AI_assisted_coding\Lab_4_3\One-Shot Prompting - Centimeters to Inches Conversion.py"
Enter length in centimeters: 25
25.0 cm is equal to 9.84 inches.
```

## 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

```
File Edit Selection View Go Run ... ← → O Lab_4_3
Welcome Zero-Shot Prompting - Leap Year Check.py One-Shot Prompting - Centimeters to Inches Conversion.py Few-Shot Prompting - Name Formatting.py ...
Few-Shot Prompting - Name Formatting.py > ...
2 #Examples:
3 "#John Smith" + "Smith, John"
4 "#Anita Rao" + "Rao, Anita"
5 def format_name(full_name):
6     parts = full_name.split()
7     if len(parts) >= 2:
8         return parts[-1] + ", " + parts[0]
9     else:
10        return "Invalid Name"
11 # Example usage:
12 name_input = input("Enter full name (First Last): ")
13 result = format_name(name_input)
14 print(result)
15 #generate remarks for Few-Shot Prompting - Name Formatting
16 # Remarks:
17 # This function takes a full name as input in the format "First Last" and splits it
18 # into first and last names. It then returns a formatted string in the "Last, First" format.
19 # The example usage demonstrates how to use the function by prompting the user for a full name
20 # and displaying the reformatted result.

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTMAN CONSOLE Python + v ⚡ ... | ☰ x
PS D:\3_2_SEM\AI_assisted_coding\Lab_4_3> & C:/Python314/python.exe "d:/3_2_SEM\AI_assisted_coding\Lab_4_3\Few-Shot Prompting - Name Formatting.py"
Enter full name (First Last): John Smith
Smith, John
PS D:\3_2_SEM\AI_assisted_coding\Lab_4_3> Rahul
Rahul : The term 'Rahul' is not recognized as the name of a cmdlet, function, script file, or operable program. Check the spelling
of the name, or if a path was included, verify that the path is correct and try again.
At line:1 char:1
+ Rahul
Ln 15, Col 1 Spaces: 4 UTF-8 CRLF {} Python ⚡ 3.14.2 ⓘ Go Live ☰
```

## 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

The screenshot shows a code editor interface with two open files:

- Comparative Analysis - Zero-Shot vs Few-Shot (Vowel Count).py**: This file contains a Python function named `count_vowels` which iterates through each character of an input string and counts the number of vowels (both uppercase and lowercase) using a generator expression. It also includes a usage example where it prompts the user for a string and prints the result.
- Few-Shot Prompting - Name Formatting.py**: This file contains a few-shot prompt template for generating a Python function. It includes examples for "Hello World" (output 3) and "Python Programming" (output 4), followed by the function definition and its logic.

In the terminal pane at the bottom, the command `python Comparative Analysis - Zero-Shot vs Few-Shot (Vowel Count).py` is run, and the output shows the program prompting for a string ("Enter a string: shiva") and then printing the count of vowels (2).

The screenshot shows a code editor interface with the following details:

- File Bar:** File, Edit, Selection, View, Go, Run, ...
- Title Bar:** Lab\_4\_3
- Explorer:** Shows files like Welcome, Zero-Shot Prompt..., One-Shot Prompt..., Few-Shot Prompt..., Comparative Analysis - Zero-Shot vs Few-Shot (Vowel Count).py, and LAB\_4\_3.
- Editor Area:** Displays a Python script titled "Comparative Analysis - Zero-Shot vs Few-Shot (Vowel Count).py". The script compares two implementations of vowel counting. It includes comments explaining accuracy, readability, and logical clarity. The code uses generator expressions and loops to count vowels in a string.
- Terminal:** Shows command-line output from running the script. It shows the script being run, user input "shiva", and the output "The number of vowels in the given string is: 2".
- Status Bar:** Includes Ln 32, Col 14, Spaces: 4, UTF-8, CRLF, Python 3.14.2, and Go Live.

## 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

The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows a tree view of files and folders. In the **LAB\_4\_3** folder, there is a **demo.txt** file and a **Few-Shot Prompting - File Handling (Line Count).py** script.
- Editor:** The **Few-Shot Prompting - File Handling (Line Count).py** script is open. It contains a function `count_lines_in_file` that reads a file and returns the number of lines. A terminal window at the bottom shows the script being run and outputting the line count for `demo.txt`.
- Terminal:** The terminal shows the command `PS D:\3_2_SEM\AI_assisted_coding\Lab_4_3> & C:/Python314/python.exe "d:/3_2_SEM/AI_assisted_coding/Lab_4_3/Few-Shot Prompting - File Handling (Line Count).py"`. It then prompts for a file path, receives `D:\3_2_SEM\AI_assisted_coding\Lab_4_3\demo.txt`, and outputs the result: `The number of lines in the file is: 3`.

This screenshot is identical to the one above, but the `count_lines_in_file` function now includes detailed comments explaining its purpose and how it handles errors.

```
1 # Write a Python function that reads a .txt file and counts the number of lines.
2 # Examples:
3 #   # File content:
4 #     # Hello
5 #     # World
6 #   # → Output: 2 lines
7 #   # File content:
8 #     # Python
9 #     # AI
10 #   # Lab
11 #   # → Output: 3 lines
12 def count_lines_in_file(file_path):
13     try:
14         with open(file_path, 'r') as file:
15             lines = file.readlines()
16             return f"The number of lines in the file is: {len(lines)}"
17     except FileNotFoundError:
18         return "File not found. Please check the file path."
19 # Example usage:
20 file_path_input = input("Enter the path to the .txt file: ")
21 result = count_lines_in_file(file_path_input)
22 print(result)
23 # generate remarks for Few-Shot Prompting - File Handling (Line Count)
24 # Remarks:
25 # This function reads a specified .txt file and counts the number of lines it contains.
26 # It uses a try-except block to handle potential file not found errors gracefully.
27 # The function opens the file in read mode, reads all lines into a list, and returns the count of lines.
28 # The example usage demonstrates how to use the function by prompting the user for the file path
29 # and displaying the line count result. The provided examples illustrate the expected output for
30 # different file contents.
31
32
```