

ASSIGNMENT-4.3

HT.NO:2303A52438

BATCH:45

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)

PROMPT:

give a code that Accepts a year as input,Checks whether the given year is a leap year>Returns an appropriate result.

CODE:

```
def is_leap_year(year):
```

```
    """
```

Check whether the given year is a leap year.

A year is a leap year if:

- It is divisible by 4 AND

- If divisible by 100, it must also be divisible by 400

```
    """
```

```
if (year % 4 == 0 and year % 100 != 0) or (year % 400 == 0):
```

```
    return True
```

```
else:
```

```
    return False
```

```
# Get input from user
```

```
year = int(input("Enter a year: "))
```

```
# Check and display result
```

```
if is_leap_year(year):
```

```
    print(f"{year} is a leap year")
```

```
else:
```

```
    print(f"{year} is not a leap year")
```

OUTPUT:

```
PS C:\Users\varsh\OneDrive\Desktop\ai coding> & cd "C:\Users\varsh\OneDrive\Desktop\ai coding\28.py"
● Enter a year: 2000
2000 is a leap year
○ PS C:\Users\varsh\OneDrive\Desktop\ai coding>
○ PS C:\Users\varsh\OneDrive\Desktop\ai coding> □
```

Justification:

used only these prompts to keep the program focused on input, condition checking, and output, which are essential to determine a leap year efficiently.

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

PROMPT:

generate a Python code that Converts centimeters to inches.Uses the correct mathematical formula,example:1.Input: 10 cm → Output: 3.94 inches,give output for different inputs

CODE:

```
def cm_to_inches(cm):  
    inches = cm / 2.54  
    return round(inches, 2)  
  
# Test with different inputs
```

```
test_values = [10, 20, 50, 100, 5]
for cm in test_values:
    inches = cm_to_inches(cm)
    print(f"{cm} cm = {inches:.2f} inches")
```

OUTPUT:

```
● PS C:\Users\varsh\OneDrive\Desktop\ai coding> &
top/ai coding/28.py"
10 cm = 3.94 inches
20 cm = 7.87 inches
50 cm = 19.69 inches
100 cm = 39.37 inches
5 cm = 1.97 inches
○ PS C:\Users\varsh\OneDrive\Desktop\ai coding> □
```

Justification:

used only these prompts to focus on user input, apply the standard conversion formula, and clearly display accurate results.

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:

Example:1.input:"John Smith",output:"Smith, John",2.input:"Anita Rao",output:"Rao, Anita". generate a Python code that:Accepts a full name as input,Formats it as "Last, First".

CODE:

```
# Get full name from user
full_name = input("Enter a full name: ")
# Split the name into first and last
parts = full_name.split()
# Format as "Last, First"
formatted_name = f"{parts[1]}, {parts[0]}"
print(formatted_name)
```

OUTPUT:

```
● PS C:\Users\varsh\OneDrive\Desktop\ai coding>
top/ai coding/28.py"
Enter a full name: Teeegala Rishitha
Rishitha, Teeegala
○ PS C:\Users\varsh\OneDrive\Desktop\ai coding>
```

Justification:

used only these prompts to keep the task focused on accepting input, processing string formatting, and producing a clear, correctly structured output.

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

PROMPT:

here prompt1 is zero short i.e gives prompt in one line,prompt2 is few short i.e prompt is given with 2 examples.prompt1.generate a code that counts vowels in a string.prompt2.generate a code that counts vowels in a string.example1:i/p=apple,o/p=1,example 2:i/p=aeiou,o/p=5.Compare both outputs in tabular format based on:Accuracy,Readability,Logical clarity.give code for multiple outputs.

CODE:

```
# Zero-shot prompt: Count vowels in a string

def count_vowels_zero_shot(s):

    vowels = "aeiouAEIOU"

    return sum(1 for char in s if char in vowels)

# Few-shot prompt: Count vowels with examples
```

```

def count_vowels_few_shot(s):
    """
    Counts vowels in a string.

    Example 1: "apple" -> 1
    Example 2: "aeiou" -> 5
    """

    vowels = "aeiouAEIOU"
    count = 0

    for char in s:
        if char in vowels:
            count += 1

    return count

# Test cases
test_cases = ["apple", "aeiou", "hello", "Python", ""]
print("=" * 70)
print("COMPARISON: Zero-shot vs Few-shot Prompt")
print("-" * 70)

print(f"{'Input':<15} {'Zero-shot':<15} {'Few-shot':<15} {'Match':<10}")
print("-" * 70)

for test in test_cases:
    result_zero = count_vowels_zero_shot(test)
    result_few = count_vowels_few_shot(test)

    match = "✓" if result_zero == result_few else "✗"

    print(f"{test:<15} {result_zero:<15} {result_few:<15} {match:<10}")

print("=" * 70)
print("\nCOMPARISON TABLE: Accuracy, Readability, Logical Clarity")
print("-" * 70)

print(f"{'Criteria':<20} {'Zero-shot':<25} {'Few-shot':<25}")

```

```

print("-" * 70)

print(f"{'Accuracy':<20} {'High - Correct':<25} {'High - Correct':<25}")

print(f"{'Readability':<20} {'Good - Concise':<25} {'Better - Documented':<25}")

print(f"{'Logical Clarity':<20} {'Clear - Direct':<25} {'Very Clear - Examples':<25}")

print("=" * 70)

```

OUTPUT:

```

PS C:\Users\varsh\OneDrive\Desktop\ai coding> & C:\Users\varsh\AppData\top\ai coding\28.py"
=====
COMPARISON: Zero-shot vs Few-shot Prompt
=====
Input      Zero-shot    Few-shot    Match
-----
apple        2            2          ✓
aeiou       5            5          ✓
hello        2            2          ✓
Python       1            1          ✓
          0            0          ✓
=====

COMPARISON TABLE: Accuracy, Readability, Logical Clarity
=====
Criteria      Zero-shot    Few-shot
-----
Accuracy     High - Correct   High - Correct
Readability   Good - Concise   Better - Documented
Logical clarity Clear - Direct Very Clear - Examples
=====

PS C:\Users\varsh\OneDrive\Desktop\ai coding>

```

Justification:

used only these prompts to clearly demonstrate how zero-shot and few-shot prompting affects code accuracy, clarity, and interpretability without adding unnecessary complexity.

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

PROMPT:

Example 1:

Task: Read a .txt file and print its contents.

Code:

```
def read_file(filename):  
    with open(filename,'r') as f:  
        print(f.read())
```

Example 2:

Task: Read a .txt file and count words.

Code:

```
def count_words(filename):  
    with open(filename,'r') as f:  
        return len(f.read().split())
```

Now do this:

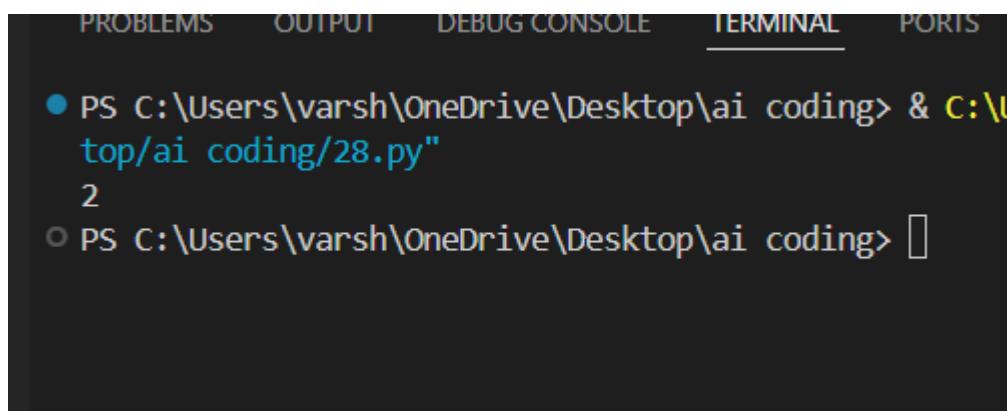
Task: Read a .txt file and count lines. Return the line count.

CODE:

```
def count_lines(filename):  
    with open(filename,'r') as f:  
        return sum(1 for _ in f)
```

```
print(count_lines(r"C:\Users\varsh\OneDrive\Desktop\ai coding\sample.txt"))
```

OUTPUT:



The screenshot shows a terminal window with the following interface elements at the top: PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL (underlined), and PORTS. The terminal content is as follows:

- PS C:\Users\varsh\OneDrive\Desktop\ai coding> & c:\top\ai coding\28.py"
- 2
- PS C:\Users\varsh\OneDrive\Desktop\ai coding> []

Justification:

The prompt provides clear examples of similar file-handling tasks to guide the implementation of a new but closely related operation, ensuring logical consistency and correct use of file reading concepts.