AI ASSISTED CODING

LAB ASSIGNMENT – 4

Student Name: Tejas Kumar

Hall Ticket Number: 2303A51489

Course: B.Tech

Task 1: Zero-Shot Prompting – Leap Year Check

Prompt Used

Write a Python function to check whether a given year is a leap year.

Detailed Explanation

Zero-shot prompting provides no examples to the AI. The AI relies purely on its prior knowledge to apply leap year rules.

Implementation Steps

Step 1: Accept the year as integer input.

Step 2: Check divisibility by 4.

Step 3: Exclude years divisible by 100.

Step 4: Include years divisible by 400.

Step 5: Combine conditions logically.

Step 6: Return the final result.

AI-Generated Code

```
def is_leap_year(year):
  if (year % 4 == 0 and year % 100 != 0) or (year % 400 == 0):
    return True
  return False
```

Testing (Input → Output)

Input: 2024 → Output: True

Input: 1900 → Output: False

Input: 2000 → Output: True

Observations

The logic works correctly for leap year identification.

Conclusion

Zero-shot prompting gives a correct but basic solution.

Task 2: One-Shot Prompting – Centimeters to Inches Conversion

Prompt Used

Convert centimeters to inches. Example: 10 cm → 3.94 inches.

Detailed Explanation

The given example helps the AI understand the expected conversion logic clearly.

Implementation Steps

Step 1: Accept centimeter value.

Step 2: Use conversion factor 2.54.

Step 3: Perform division.

Step 4: Store result.

Step 5: Return output.

Step 6: Verify using test inputs.

AI-Generated Code

```
def cm_to_inches(cm):
  return cm / 2.54
```

Testing (Input → Output)

Input: 10 → Output: 3.94

Input: 25 → Output: 9.84

Observations

The function is concise and accurate.

Conclusion

One-shot prompting improves clarity and precision.

Task 3: Few-Shot Prompting – Name Formatting

Prompt Used

Format a full name as 'Last, First' using examples.

Detailed Explanation

Multiple examples allow the AI to infer the correct string manipulation pattern.

Implementation Steps

Step 1: Accept name string.

Step 2: Split into parts.

Step 3: Assign first and last names.

Step 4: Rearrange order.

Step 5: Add comma separator.

Step 6: Return formatted name.

AI-Generated Code

```
def format_name(name):
  first, last = name.split()
  return f"{last}, {first}"
```

Testing (Input → Output)

Input: John Smith → Output: Smith, John

Input: Anita Rao → Output: Rao, Anita

Observations

Works correctly for two-part names.

Conclusion

Few-shot prompting improves formatting accuracy.

Task 4: Comparative Analysis – Zero-Shot vs Few-Shot (Vowel Count)

Prompt Used

Count the number of vowels in a string using different prompting techniques.

Detailed Explanation

This task compares how examples influence code clarity and structure.

Implementation Steps

Step 1: Accept string input.

Step 2: Define vowels.

Step 3: Initialize counter.

Step 4: Traverse characters.

Step 5: Increment count on match.

Step 6: Return total count.

AI-Generated Code

```
def count_vowels(text):
  vowels = 'aeiouAEIOU'
  count = 0
  for c in text:
    if c in vowels:
      count += 1
  return count
```

Testing (Input → Output)

Input: hello → Output: 2

Input: education → Output: 5

Observations

Few-shot code is cleaner and more readable.

Conclusion

Examples significantly improve AI-generated output.

Task 5: Few-Shot Prompting – File Handling (Line Count)

Prompt Used

Count the number of lines in a text file using examples.

Detailed Explanation

Few-shot prompting helps the AI understand file-processing logic correctly.

Implementation Steps

Step 1: Accept filename.

Step 2: Open file in read mode.

Step 3: Read all lines.

Step 4: Count lines.

Step 5: Return count.

Step 6: Display result.

AI-Generated Code

```
def count_lines(filename):
  with open(filename, 'r') as f:
    return len(f.readlines())
```

Testing (Input → Output)

File with 3 lines → Output: 3

Empty file → Output: 0

Observations

The logic works correctly but assumes file exists.

Conclusion

Few-shot prompting produces reliable file-handling code.