

AI Assisted Coding

Assignment 4.3

Name:D.udayan

Hall.no:2303A54017

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:

```
#generate a python function to accepts a year as a in put and if the given year  
is a leap year if it is not a leap year give it is not a leap year
```

AI generated:

```
1  #generate a python function to accepts a year as a in put and if the given year is a leap year if it is not a leap year give it is not a leap year
2
3  def is_leap_year(year):
4      """
5      Return True if 'year' is a leap year, False otherwise.
6      Accepts an int or a string convertible to int.
7      """
8      try:
9          y = int(year)
10     except (TypeError, ValueError):
11         raise ValueError("year must be an integer or string of digits")
12     return (y % 4 == 0) and (y % 100 != 0 or y % 400 == 0)
13
14 # Example usage:
15 if __name__ == "__main__":
16     year_input = input("Enter a year: ")
17     if is_leap_year(year_input):
18         print(f"{int(year_input)} is a leap year")
19     else:
20         print(f"{int(year_input)} is not a leap year")
```

Output:

```
Enter a year: 2003
2003 is not a leap year
PS C:\Users\MYPC\OneDrive\Desktop\ai assisted coding> 
```

Explanation:

Using a leap year logic

If the given year is divisible by 4 and 400 it is a leap year

If the give year is not divisible by 100 it is not a 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

Prompt:

#Generate a python function code that converts centimeters to inches by using formula

AI generated:

```
def cm_to_inches(cm, decimals=2):  
    """  
    Convert centimeters to inches.  
    Accepts int, float, or string convertible to float.  
    Returns a float rounded to `decimals`.  
    """  
    try:  
        c = float(cm)  
    except (TypeError, ValueError):  
        raise ValueError("cm must be a number or numeric string")  
    inches = c * 0.3937007874015748  
    return round(inches, decimals)  
  
if __name__ == "__main__":  
    # example: 10 cm -> 3.94 inches  
    s = input("Enter length in centimeters (e.g. 10): ")  
    try:  
        c = float(s)  
    except (TypeError, ValueError):  
        print("Invalid input")  
    else:  
        print(f"{c} cm -> {cm_to_inches(c, 2)} inches")
```

Output:

```
Enter length in centimeters (e.g. 10): 12  
12.0 cm -> 4.72 inches  
PS C:\Users\MYPC\OneDrive\Desktop\ai assisted coding> 
```

Explanation:

Using a formula $\text{cm} \times 0.3937007874015748 = \text{inches}$ converting centimeters in to 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

Prompt:

```
Format a full name as "last, first".  
Accepts a string with first and last name separated by space.
```

AI generated:

```
def format_name(full_name):  
    """  
    Format a full name as "last, first".  
    Accepts a string with first and last name separated by space.  
    """  
    try:  
        parts = full_name.strip().split()  
        if len(parts) < 2:  
            raise ValueError("Please provide both first and last name")  
        first_name = parts[0]  
        last_name = parts[-1]  
        return f"{last_name}, {first_name}"  
    except AttributeError:  
        raise ValueError("full_name must be a string")  
  
if __name__ == "__main__":  
    name_input = input("Enter a full name: ")  
    try:  
        formatted = format_name(name_input)  
        print(f"{name_input} -> {formatted}")  
    except ValueError as e:  
        print(f"Error: {e}")
```

Output:

```
Enter a full name: duddukuri udayan
duddukuri udayan -> udayan, duddukuri
```

Explanation:

Splits the name into half (first and last)

Takes first and last name and formats it as last and first

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:

Generate a python function code that counts vowels in a string using zero-shot and Generate same a python function code using few-shot and compare the both output on accuracy,readability,logical clarity

AI generated:

```
# Zero-shot: Count vowels in a string
def count_vowels_zero_shot(text):
    """
    Count vowels in a string (a, e, i, o, u).
    Case-insensitive.
    """
    vowels = "aeiouAEIOU"
    return sum(1 for char in text if char in vowels)

# Few-shot: Count vowels with examples
def count_vowels_few_shot(text):
    """
    Count vowels in a string.
    Examples:
        "hello" -> 2 (e, o)
        "Python" -> 1 (o)
        "aeiou" -> 5
    """
    vowel_list = ['a', 'e', 'i', 'o', 'u']
    count = 0
    for char in text.lower():
        if char in vowel_list:
            count += 1
    return count

# Comparison
if __name__ == "__main__":
    test_strings = ["Hello World", "Python", "aeiou", "xyz"]

    print("Comparison of Zero-shot vs Few-shot:\n")
    for test in test_strings:
        zero = count_vowels_zero_shot(test)
        few = count_vowels_few_shot(test)
        print(f"{test}: Zero-shot={zero}, Few-shot={few}, Match={zero==few}")

    print("\nAccuracy: Both are identical (100%)")
    print("Readability: Few-shot is more explicit with loop")
    print("Logical Clarity: Zero-shot is more concise with comprehension")
```

Output:

```
'Python': Zero-shot=1, Few-shot=1, Match=True
'aeiou': Zero-shot=5, Few-shot=5, Match=True
'aeiou': Zero-shot=5, Few-shot=5, Match=True
'xyz': Zero-shot=0, Few-shot=0, Match=True
'xyz': Zero-shot=0, Few-shot=0, Match=True

Accuracy: Both are identical (100%)
Readability: Few-shot is more explicit with loop
Logical Clarity: Zero-shot is more concise with comprehension
```

Explanation:

Calls both functions and compares their outputs

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:

Generate a python function code that reads the text files and counts the number of lines in the files and returns the line count

AI generated:

```
def count_lines_in_file(file_path):  
    """  
    Read a text file and count the number of lines.  
    Returns the line count as an integer.  
    """  
    try:  
        with open(file_path, 'r', encoding='utf-8') as file:  
            line_count = sum(1 for line in file)  
            return line_count  
    except FileNotFoundError:  
        raise FileNotFoundError(f"File '{file_path}' not found")  
    except IOError as e:  
        raise IOError(f"Error reading file: {e}")  
  
if __name__ == "__main__":  
    file_input = input("Enter the file path: ")  
    try:  
        count = count_lines_in_file(file_input)  
        print(f"'{file_input}' has {count} lines")  
    except (FileNotFoundError, IOError) as e:  
        print(f"Error: {e}")
```

Output:

```
Enter the file path: /home/uday/Documents/notes.txt  
Error: File '/home/uday/Documents/notes.txt' not found
```

Explanation:

Program reads a text file

Counts total number of lines

Uses `with open()` for safe file handling

Uses `sum()` for efficient counting

Handles file errors gracefully