AI Assisted Coding

ASSIGNMENT 4

Name: B Vrindha
HT No: 2303A52003
Batch: 31

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.

Question 1: Zero-Shot Prompting (Basic Lab Task)

Task 1:

Write a Python function that classifies a given text as Spam or Not Spam using zero-shot prompting.
Steps:
1. Construct a prompt without any examples.
2. Clearly specify the output labels.
3. Display only the predicted label.
Input: "Congratulations! You have won a free lottery ticket."
Expected Output: Spam

Prompt:

```
'''
Classify the text as Spam or Not Spam.
Return only the label.

Text: "Congratulations! You have won a free lottery ticket."
'''
```

Code:

```python
def classify_text(text):
    spam_keywords = ["win", "free", "prize", "money", "offer", "click", "buy now"]
    for keyword in spam_keywords:
        if keyword in text.lower():
            return "Spam"
    return "Not Spam"

# Take input from user
user_input = input("Enter the text to classify: ")
print(classify_text(user_input))
```

Output(O/P)

```
Enter the text to classify: "Congratulations! You have won a free lottery ticket."
Spam
```

Explanation:

- The function scans the input text for predefined spam-related keywords.
- If at least one spam keyword is found, the text is classified as Spam.
- If no spam keywords are found, the text is classified as Not Spam.

Question 2: One-Shot Prompting (Emotion detection)

Task 2:

Write a Python program that detects the emotion of a sentence using one-shot prompting.
Emotions: [‘happy', 'sad', 'angry', 'excited', 'nervous', ’neutral’]
Steps:
1. Provide one labeled example inside the prompt.
2. Take a sentence as input.
3. Print the predicted emotion

Prompt:

```
'''
Detect the emotion of the sentence.
Return only one label from: happy, sad, angry, excited, nervous, neutral.
'''
```

Code:

```
def detect_emotion(sentence):
    happy_keywords = ["joy", "happy", "pleased", "delighted", "glad"]
    sad_keywords = ["sad", "unhappy", "sorrow", "depressed", "down"]
    angry_keywords = ["angry", "mad", "furious", "irate", "annoyed"]
    excited_keywords = ["excited", "thrilled", "eager", "enthusiastic", "elated"]
    nervous_keywords = ["nervous", "anxious", "worried", "tense", "apprehensive"]

    sentence_lower = sentence.lower()

    for keyword in happy_keywords:
        if keyword in sentence_lower:
            return "happy"
    for keyword in sad_keywords:
        if keyword in sentence_lower:
            return "sad"
    for keyword in angry_keywords:
        if keyword in sentence_lower:
            return "angry"
    for keyword in excited_keywords:
        if keyword in sentence_lower:
            return "excited"
    for keyword in nervous_keywords:
        if keyword in sentence_lower:
            return "nervous"

    return "neutral"
print(detect_emotion("I am so thrilled about my new job!"))
```

Output:

```
[Running] python -u "c:\Users\PC\OneDrive\Desktop\AIAC\oneshot.python"
excited
```

Explanation:

- One labeled example is provided to show how emotion classification works.
- The model uses this single example to understand the task.
- A new sentence is given as input.
- The model predicts and outputs only one emotion label from the given list.

Question 3: Few-Shot Prompting (Student Grading Based on Marks)

Task 3:

Write a Python program that predicts a student's grade based on marks using few-shot prompting.
Grades: ['A', 'B', 'C', 'D', 'F']
Grading Criteria (to be inferred from examples):
• 90–100 → A
• 80–89 → B
• 70–79 → C
• 60–69 → D
• Below 60 → F

Prompt:

```
'''
Predict the student's grade based on marks.
Possible grades: A, B, C, D, F.
Examples:
Marks: 95
Grade: A
Marks: 85
Grade: B
Marks: 75
Grade: C
Marks: 65
Grade: D
Marks: 45
Grade: F
Now predict:
Marks: 92
Grade:
'''
```

Code:

```python
# Function to determine grade based on marks
def get_grade(marks):
    if marks >= 90:
        return "A"
    elif marks >= 80:
        return "B"
    elif marks >= 70:
        return "C"
    elif marks >= 60:
        return "D"
    else:
        return "F"

# Take input from user
marks = int(input("Enter marks: "))
print(get_grade(marks))
```

Output:

```
Enter marks: 90
A
```

Explanation:

- Multiple examples with marks and grades are provided.
- The model learns the grading pattern from these examples.
- New marks are given as input without explicitly stating rules.
- The model predicts the appropriate grade based on inferred criteria.

Question 4:Multi-Shot Prompting (Indian Zodiac Sign Prediction using Month Name)

Task 4:

Write a Python program that predicts a person's Indian Zodiac sign (Rashi) based on the month of birth (month name) using multi-shot prompting.

Indian Zodiac Order (Simplified Month-Based Model):

The Indian Zodiac cycle starts in March with Mesha and follows this order:

March → Mesha
April → Vrishabha
May → Mithuna
June → Karka
July → Simha
August → Kanya
September → Tula
October → Vrischika
November → Dhanu
December → Makara

January → Kumbha
February → Meena

Prompt:

```
'''
Predict the Indian Zodiac sign (Rashi) based on the birth month name.
Examples:
March → Mesha
April → Vrishabha
May → Mithuna
June → Karka
July → Simha
August → Kanya
September → Tula
October → Vrischika
November → Dhanu
December → Makara
January → Kumbha
February → Meena
Print only the Rashi name.
Take Input From User.
'''
```

Code:

```
def predict_rashi(month):
    """
    Predict the Indian Zodiac sign (Rashi) based on the birth month name.

    Parameters:
    month (str): The birth month name.

    Returns:
    str: The corresponding Rashi name.
    """
    rashi_dict = {
        'March': 'Mesha',
        'April': 'Vrishabha',
        'May': 'Mithuna',
        'June': 'Karka',
        'July': 'Simha',
        'August': 'Kanya',
        'September': 'Tula',
        'October': 'Vrischika',
        'November': 'Dhanu',
        'December': 'Makara',
        'January': 'Kumbha',
        'February': 'Meena'
    }
    return rashi_dict.get(month, "Invalid month name")  # Return Rashi or an error message

# Take input from user
birth_month = input("Enter your birth month: ")
rashi = predict_rashi(birth_month)
print(rashi)
```

Output:

```
Enter your birth month: January
Kumbha
```

Explanation:

- The code stores each month and its Indian Zodiac (Rashi) in a dictionary.
- It asks the user to enter their birth month.
- It retrieves the corresponding Rashi from the dictionary.
- It prints the Rashi to the user.
- If the month is invalid, it prints "Invalid month".

Question 5: Result Analysis Based on Marks

Task 5:

Write a Python program that determines whether a student Passes or Fails based on marks using Chain-of-Thought (CoT) prompting.

Result Categories: ['Pass', 'Fail']

Prompt:

```
'''
You are a teacher. Your task is to decide whether a student Passes or Fails based on their marks.
Think step by step and explain your reasoning before giving the result.
generate python code using COT (Chain of Thought) prompting technique.

'''
```

Code:

```python
# Step 1: Take input from the user
marks = float(input("Enter the marks obtained by the student: "))

# Step 2: Define the passing criteria
passing_marks = 40.0  # Assuming 40 is the passing mark

# Step 3: Determine Pass or Fail
if marks >= passing_marks:
    result = "Pass"
else:
    result = "Fail"

# Step 4: Print the result
print(result)
```

Output:

```
Enter the marks obtained by the student: 40
Pass
```

Explanation:

- Chain-of-Thought (CoT) means thinking step by step before giving the answer.
- The prompt first sorts marks and the reasoning for Pass/Fail.
- The reasoning is written as “CoT: First check then decide” before the output.
- The prompt asks the AI to apply the same reasoning to new marks.
- The AI returns both the step-by-step reasoning and the final result (Pass or Fail).

Question 6: Voting Eligibility Check (Chain-of-Thought Prompting)

Task 6:

Write a Python program that determines whether a person is eligible to vote using Chain-of-Thought (CoT) prompting.

Prompt:

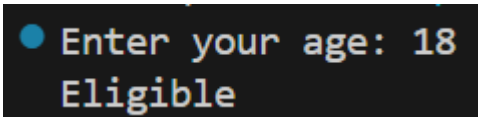
```
'''
Write a Python program that determines whether a person is eligible to vote using Chain-of-Thought reasoning.
The program should:
Ask the user for their age.
Explain step by step why the person is eligible or not.
Print the final result as "Eligible" or "Not Eligible".
A person is eligible to vote if their age is 18 or above.
'''
```

Code:

```
def check_voting_eligibility(age):
    # Step 1: Define the legal voting age
    legal_voting_age = 18

    # Step 2: Compare the user's age with the legal voting age
    if age >= legal_voting_age:
        # Step 3: If age is greater than or equal to legal voting age, they are eligible
        return "Eligible"
    else:
        # Step 4: If age is less than legal voting age, they are not eligible
        return "Not Eligible"
# Take input from user
user_age = int(input("Enter your age: "))
eligibility = check_voting_eligibility(user_age)
print(eligibility)
```

Output:



Explanation:

- The program asks the user for their age.
- It checks step by step if the age is 18 or more
- Prints each reasoning step to explain eligibility.
- Shows the final result: "Eligible" or "Not Eligible".

Question 7: Prompt Chaining (String Processing – Palindrome Names)

Task 7:

Write a Python program that uses the prompt chaining technique to identify palindrome names from a list of student names.

Prompt:

```
'''
Write a Python program that identifies palindrome names from a list of student names using prompt chaining.
The program should:
Take a list of names as input.
Check each name step by step to see if it reads the same forwards and backwards.
Collect all palindrome names into a new list.
Print the list of palindrome names.
'''
```

Code:

```
def is_palindrome(name):
    # Step 1: Normalize the name by converting it to lowercase
    normalized_name = name.lower()
    # Step 2: Check if the normalized name reads the same forwards and backwards
    return normalized_name == normalized_name[::-1]
def find_palindrome_names(names):
    # Step 3: Initialize an empty list to collect palindrome names
    palindrome_names = []
    # Step 4: Iterate through each name in the input list
    for name in names:
        # Step 5: Check if the current name is a palindrome
        if is_palindrome(name):
            # Step 6: If it is a palindrome, add it to the palindrome names list
            palindrome_names.append(name)
    return palindrome_names
# Example usage
student_names = ["Anna", "Bob", "Cathy", "David", "Eve", "Hannah", "John", "Level"]
palindrome_names = find_palindrome_names(student_names)
print("Palindrome Names:")
print(palindrome_names)
```

Output:

```
[Running] python -u "c:\Users\PC\OneDrive\Desktop\AIAC\chainPrompting.python"
Palindrome Names:
['Anna', 'Bob', 'Eve', 'Hannah', 'Level']
```

Explanation:

- Prompt Chaining breaks a task into smaller sequential steps.
- Take a list of student names as input.
- Check each name step by step to see if it is a palindrome.
- Collect all palindrome names into a new list.
- Print the list of palindrome names.
- This approach makes the program organized, easy to follow, and modular.

Question 8: Prompt Chaining (String Processing – Word Length Analysis)

Task 8:

Write a Python program that uses prompt chaining to analyze a list of words. In the first prompt, generate a list of words. In the second prompt, traverse the list and calculate the length of each word. In the third prompt, use the output of the previous step to determine whether each word is Short (length less than 5) or Long (length greater than or equal to 5), and display the result for each word

Prompt:

```
'''
Write a Python program that analyzes a list of words using prompt chaining.
The program should:
 Generate a list of words.
 Traverse the list and calculate the length of each word.
 Determine whether each word is Short (length < 5) or Long (length >= 5) based on the lengths
 Display the result for each word.
'''
```

Code:

```
def analyze_word_length(words):
    # Step 1: Initialize an empty list to store the results
    results = []
    # Step 2: Iterate through each word in the input list
    for word in words:
        # Step 3: Calculate the length of the current word
        length = len(word)
        # Step 4: Determine if the word is Short or Long based on its length
        if length < 5:
            results.append((word, "Short"))
        else:
            results.append((word, "Long"))
    return results
# Example usage
word_list = ["apple", "bat", "cat", "dolphin", "egg", "fish", "grape"]
word_analysis = analyze_word_length(word_list)
print("Word Length Analysis:")
for word, length_type in word_analysis:
    print(f"{word}: {length_type}")
```

Output:

```
Word Length Analysis:
apple: Long
bat: Short
cat: Short
dolphin: Long
egg: Short
fish: Short
grape: Long
```

Explanation:

- Break the task into 3 sequential steps.
- Generate the list of words.
- Find the length of each word.
- Classify words as Short or Long and display results.
- Makes the program modular, clear, and easy to follow.