

## AI Assisted Coding

### ASSIGNMENT:4

Name:K.Rupaas shyni

2303A52417

Batch:32

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 given text into one of the following labels only: Spam or Not Spam.

Do not explain your answer. Take Input From User.

Code:

```
'''
Write a Python function that classifies a given text as Spam or Not
Spam using zero-shot prompting.
'''

def classify_text(text):
    spam_keywords = ["win", "free", "price", "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:

```
PS C:\Users\rupaa\OneDrive\Desktop\AIAC> & C:/Users/rupaa/anaconda3/python.exe c:/U
p/AIAC/assig-4.py
Enter the text to classify: "Congratulations! You have won a free lottery ticket."
Spam
PS C:\Users\rupaa\OneDrive\Desktop\AIAC> █
```

Explanation:

This function checks if any of the predefined spam keywords are present in the input text.

If any keyword is found, it classifies the text as "Spam"; otherwise, it classifies it 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 given sentence.

Possible emotions are: happy, sad, angry, excited, nervous, neutral.

Example:

Sentence: "I am feeling very joyful today"

Emotion: happy

Now classify the emotion of this sentence:

"I am worried about my exam results"

Print only the predicted emotion.

Take Input From User.

Code:

```
def detect_emotion(sentence):  
    sentence = sentence.lower()  
  
    if "happy" in sentence or "joy" in sentence or "glad" in sentence:  
        return "happy"  
    elif "sad" in sentence or "sorrow" in sentence or "unhappy" in sentence:  
        return "sad"  
    elif "angry" in sentence or "mad" in sentence or "furious" in sentence:  
        return "angry"  
    else:  
        return "Neutral"  
  
# Take input from user  
sentence = input("Enter a sentence to detect emotion: ")  
print(detect_emotion(sentence))
```

Output:

```
top/AIAC/assig-4.py  
Enter a sentence to detect emotion: iam feeling very glad to meet you  
happy
```

Explanation:

The code defines a function `detect_emotion` that takes a sentence as input and returns the detected emotion based on specific keywords. It converts the sentence to lowercase and checks for the presence of keywords associated with different emotions such as happy, sad, angry, excited, and nervous. If none of these keywords are found, it returns "neutral". The user is prompted to enter a sentence, and the detected emotion is printed.

### 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 grade based on marks.

Possible grades are: A, B, C, D, F.

Examples:

Marks: 95

Grade: A

Marks: 82

Grade: B

Marks: 74

Grade: C

Marks: 61

Grade: D

Marks: 45

Grade: F

Print only the grade.

Take Input From User.

Code:

```
Write a Python program that predicts a student's grade based on marks using few-shot prompting.
'''
def predict_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 = float(input("Enter the student's marks: "))
print(f"The predicted grade is: {predict_grade(marks)}")
```

Output:

```
Enter the student's marks: 90
The predicted grade is: A
```

Explanation:

This code defines a function `get_grade` that takes an integer input `marks` and returns a corresponding grade based on predefined thresholds.

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:

```
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.
'''
def predict_rashi(month):
    month = month.lower()
    rashi_dict = {
        "january": "Capricorn",
        "february": "Aquarius",
        "march": "Pisces",
        "april": "Aries",
        "may": "Taurus",
        "june": "Gemini",
        "july": "Cancer",
        "august": "Leo",
        "september": "Virgo",
        "october": "Libra",
        "november": "Scorpio",
        "december": "Sagittarius"
    }
    return rashi_dict.get(month, "Invalid month")
# Take input from user
month = input("Enter the month of birth: ")
print(f"The predicted Rashi is: {predict_rashi(month)}")
```

Output:

```
PS C:\Users\rupaa\OneDrive\Desktop\AIAC
top/AIAC/assig-4.py
Enter the month of birth: january
The predicted Rashi is: Capricorn
```

Explanation:

This code defines a dictionary that maps each month of the year to its corresponding Indian Zodiac sign (Rashi). It then prompts the user to input their birth month, retrieves the associated Rashi from the dictionary, and prints it. If the user enters an invalid month, it returns "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:

Determine whether the student Passes or Fails based on marks.

Think step by step before giving the final answer.

At the end, print only Pass or Fail. Take Input From User.

Code:

```
Write a Python program that determines whether a student
Passes or Fails based on marks using Chain-of-Thought (CoT)
prompting. '''
def pass_fail(marks):
    if marks >= 40:
        return "Pass"
    else:
        return "Fail"
# Take input from user
marks = float(input("Enter the student's marks: "))
print(f"The student has: {pass_fail(marks)}")
```

Output:

```
Enter the student's marks: 50
The student has: Pass
```

Explanation:

This code takes the marks obtained by a student as input and checks if the student has passed or failed based on a predefined passing criteria (40 marks). If the marks are greater than or equal to 40, it prints "Pass"; otherwise, it prints "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:

Determine whether a person is eligible to vote.

The minimum voting age is 18.

Reason step by step and then give the final answer.

Print only Eligible or Not Eligible. Take Input From User.

Code:

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

def check_voting_eligibility(age):
    if age >= 18:
        return "Eligible to vote"
    else:
        return "Not eligible to vote"
# Take input from user
age = int(input("Enter your age: "))
print(check_voting_eligibility(age))
```

Output:

```
Enter your age: 17
Not eligible to vote
```

Explanation:

This code snippet checks if a user is eligible to vote based on their age. It prompts the user to enter their age and then evaluates whether the age is 18 or older. If the condition is met, it prints "Eligible"; otherwise, it prints "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:

1. Given the following list of student names, prepare them for palindrome checking:

Names: ["madam", "arjun", "level", "rahul", "radar"]

2. Using the processed list from the previous step, check each name and identify whether it is a palindrome.
3. From the previous results, display only the names that are palindromes.

Code:

```
# Given list of student names
names = ["madam", "arjun", "level", "rahul", "radar"]

# Step 1: Prepare names for palindrome checking (convert to lowercase)
processed_names = [name.lower() for name in names]

# Step 2: Check each name and identify palindromes
palindrome_results = []
for name in processed_names:
    is_palindrome = name == name[::-1]
    palindrome_results.append((name, is_palindrome))

# Step 3: Display only palindromic names
palindromes = [name for name, is_palindrome in palindrome_results if is_palindrome]

print("Original names:", names)
print("\nPalindrome check results:")
for name, is_palindrome in palindrome_results:
    print(f"{name}: {'Palindrome' if is_palindrome else 'Not Palindrome'}")

print("\nPalindromic names:", palindromes)
```

Output:

```
Palindrome check results:
madam: Palindrome
arjun: Not Palindrome
level: Palindrome
rahul: Not Palindrome
radar: Palindrome

Palindromic names: ['madam', 'level', 'radar']
PS C:\Users\rupaa\OneDrive\Desktop\AIAC> █
```

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:

1. From the previous results, display only the names that are palindromes.
2. Using the generated list of words, calculate the length of each word.
3. Using the word lengths from the previous step, classify each word as:

Short if length is less than 5

Long if length is greater than or equal to 5

Display each word with its classification.

Code:

```

def filter_palindromes(names):
    """Filter names that are palindromes"""
    palindromes = [name for name in names if name.lower() == name.lower()[::-1]]
    return palindromes

def classify_word_length(words):
    """Classify words as Short or Long based on length"""
    classifications = []
    for word in words:
        length = len(word)
        classification = "Short" if length < 5 else "Long"
        classifications.append((word, length, classification))
    return classifications

# Example usage
names = ["anna", "bob", "racecar", "john", "level", "sarah"]
palindromes = filter_palindromes(names)

print("Palindromes:", palindromes)

# Classify word lengths
word_classifications = classify_word_length(palindromes)
print("\nWord Length Classifications:")
for word, length, classification in word_classifications:
    print(f"{word} - Length: {length} - {classification}")

```

Output:

```

Palindromes: ['anna', 'bob', 'racecar', 'level']

Word Length Classifications:
anna - Length: 4 - Short
bob - Length: 3 - Short
racecar - Length: 7 - Long
level - Length: 5 - Long

```

