# AI ASSISTED CODING ASSIGNMENT – 4

2303A51098

Shivani

BATCH-02

**Question1. Zero-Shot Prompting (Basic Lab Task)**

**Task:**

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

```
#Write a python program which takes input a statement and checks whether it is a spam or not and give output accordingly
def is_spam(statement):
    spam_keywords = ["buy now", "free", "click here", "subscribe", "limited time offer", "winner", "cash prize", "urgent", "act now",
    statement_lower = statement.lower()
    for keyword in spam_keywords:
        if keyword in statement_lower:
            return "Spam"
    return "Not Spam"
user_input = input("Enter a statement: ")
print(is_spam(user_input))
```

**Question2. One-Shot Prompting (Emotion detection)**

**Task:**

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

```
#input i am happy output happy (emotions= [‘happy', 'sad', 'angry', 'excited', 'nervous', ’neutral’])
def detect_emotion(statement):
    emotions = ['happy', 'sad', 'angry', 'excited', 'nervous', 'neutral']
    statement_words = statement.lower().split()
    detected_emotions = [word for word in statement_words if word in emotions]
    if detected_emotions:
        return f"Detected emotions: {', '.join(detected_emotions)}"
    else:
        return "No emotions detected"
user_input = input("Enter a statement: ")
print(detect_emotion(user_input))
```

**Question3. Few-Shot Prompting (Student Grading Based on Marks)**

Task:

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

```
'''
input 90 to 100 output A
input 80 to 89 output B
input 70 to 79 output C
input 60 to 69 output D
input below 60 output F'''
def grade_from_score(score):
    if not isinstance(score, (int, float)):
        return "Invalid score"
    elif score < 0 or score > 100:
        return "Invalid score"
    elif 90 <= score <= 100:
        return "Grade: A"
    elif 80 <= score < 90:
        return "Grade: B"
    elif 70 <= score < 80:
        return "Grade: C"
    elif 60 <= score < 70:
        return "Grade: D"
    else:
        return "Grade: F"
try:
    score = float(input("Enter the score (0-100): "))
    print(grade_from_score(score))
except ValueError:
    print("Invalid score")
```

**Question4. Multi-Shot Prompting (Indian Zodiac Sign Prediction using Month Name)**

**Task:**

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

```
'''input March  output Mesha
input April  output Vrishabha
input May  output Mithuna
input June  output Karka
input July  output Simha
input August  output Kanya
input September  output Tula
input October  output Vrischika
input November  output Dhanu
input December  output Makara
input January  output Kumbha
input February  output Meena'''

def zodiac_sign(month):
    month = month.lower()
    zodiac_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 zodiac_dict.get(month, "Invalid month")
user_input = input("Enter a month: ")
print(zodiac_sign(user_input))
```

**Question5. Result Analysis Based on Marks**

**Task:** 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']

```
'''
read the input a number and number should be from 0 to 100 if it is in range
check whether the number is greater than or eqaul to 35 or not ,if it is greater than or equal to 35 print 'Pass'
else print 'Fail'
if number is out of range print 'Check the number once again'
'''
def check_pass_fail(number):
    if not isinstance(number, (int, float)):
        return "Check the number once again"
    elif number < 0 or number > 100:
        return "Check the number once again"
    elif number >= 35:
        return "Pass"
    else:
        return "Fail"

try:
    number = float(input("Enter a number (0-100): "))
    print(check_pass_fail(number))
except ValueError:
    print("Check the number once again")
```

**Question6. Voting Eligibility Check (Chain-of-Thought Prompting)**

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

```
'''read input a number and check whether the number is >=18 or not
if it is >=18 print 'Eligible to vote'
else print 'Not eligible to vote' '''

def check_voting_eligibility(age):
    if not isinstance(age, int):
        return "Invalid age"
    elif age < 0:
        return "Invalid age"
    elif age >= 18:
        return "Eligible to vote"
    else:
        return "Not eligible to vote"
try:
    age = int(input("Enter your age: "))
    print(check_voting_eligibility(age))
except ValueError:
    print("Invalid age")
```

**Q7 Prompt Chaining (String Processing – Palindrome Names)**

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

```
'''read the student names from input and store them in a list
iterate the list and check whether the name palindrome or not
if it is palindrome store it in a new list
then print the new list'''
def filter_palindrome_names(names):
    palindrome_names = [name for name in names if name.lower() == name.lower()[::-1]]
    return palindrome_names
user_input = input("Enter student names separated by commas: ")
names_list = [name.strip() for name in user_input.split(',')]
result = filter_palindrome_names(names_list)
print(result)
```

**Q8 Prompt Chaining (String Processing – Word Length Analysis)**

**Task:** 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

```
'''
Use prompt chaining to analyze a list of words.

Step 1: Generate a list of at least 5 meaningful English words.

Step 2: Using the output from Step 1, traverse the list and calculate the length of each word.

Step 3: Using the output from Step 2, classify each word as:

Short (length < 5)

Long (length ≥ 5)

Finally, display each word along with its length and classification.

Show each step clearly and use the output of the previous step as input for the next step.
'''
def generate_word_list():
    return ["apple", "cat", "elephant", "dog", "banana"]
def calculate_word_lengths(word_list):
    return {word: len(word) for word in word_list}
def classify_words(word_lengths):
    classification = {}
    for word, length in word_lengths.items():
        if length < 5:
            classification[word] = (length, "Short")
        else:
            classification[word] = (length, "Long")
    return classification
word_list = generate_word_list()
word_lengths = calculate_word_lengths(word_list)
classified_words = classify_words(word_lengths)
for word, (length, category) in classified_words.items():
    print(f"Word: {word}, Length: {length}, Classification: {category}")
```