AIAC – 4.1

Batch 32, 2303A52065

Q1. 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.

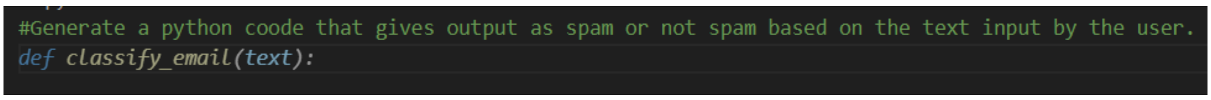
Input:

"Congratulations! You have won a free lottery ticket."

Expected Output:

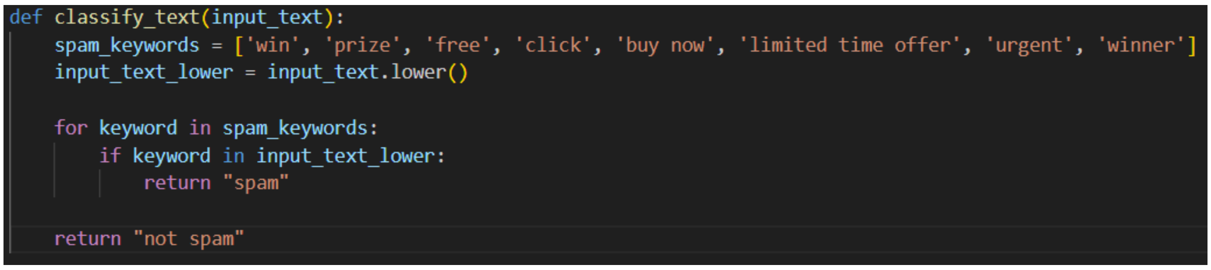
Spam

Prompt:

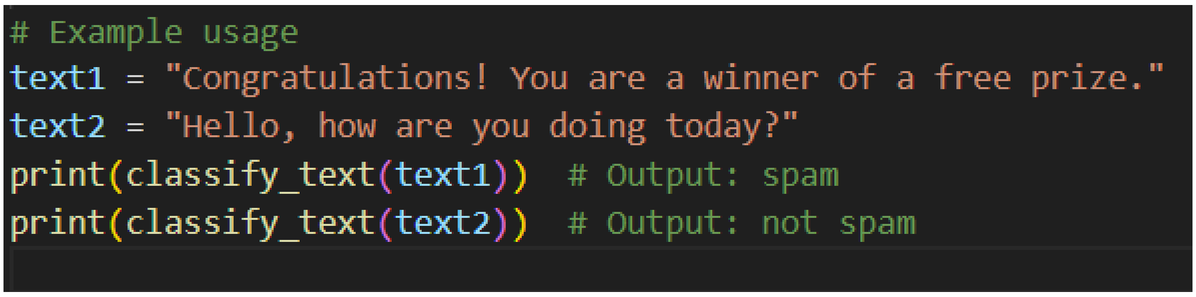


```
#Generate a python coode that gives output as spam or not spam based on the text input by the user.
def classify_email(text):
```

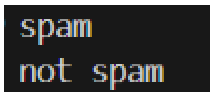
Code:



```
def classify_text(input_text):
    spam_keywords = ['win', 'prize', 'free', 'click', 'buy now', 'limited time offer', 'urgent', 'winner']
    input_text_lower = input_text.lower()

    for keyword in spam_keywords:
        if keyword in input_text_lower:
            return "spam"

    return "not spam"
```



```
# Example usage
text1 = "Congratulations! You are a winner of a free prize."
text2 = "Hello, how are you doing today?"
print(classify_text(text1))  # Output: spam
print(classify_text(text2))  # Output: not spam
```

Output: 

```
spam
not spam
```

Justification:

Since no examples were provided, the AI classified the message using basic keyword-based logic. The generated function correctly identified spam-related words in the input text. The output label was displayed clearly as required. This shows that zero-shot prompting can still produce correct results when the task is simple and well defined.

Q2. 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’]

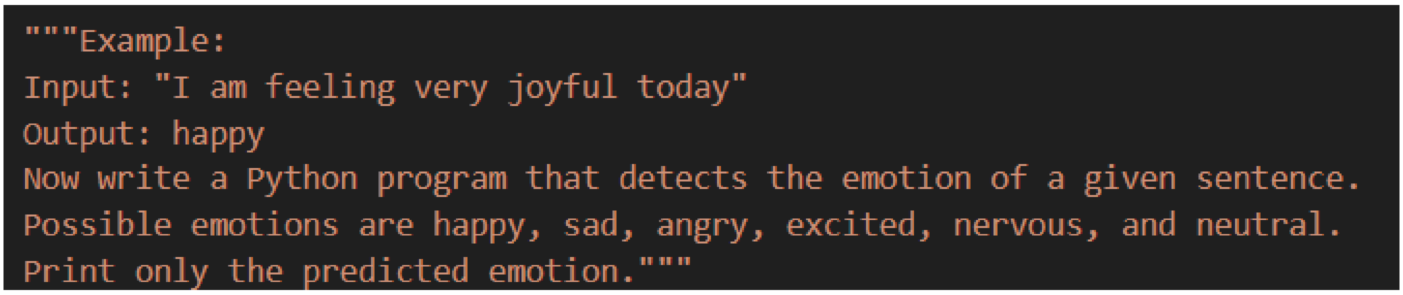
Steps:

1. Provide one labeled example inside the prompt.

2. Take a sentence as input.

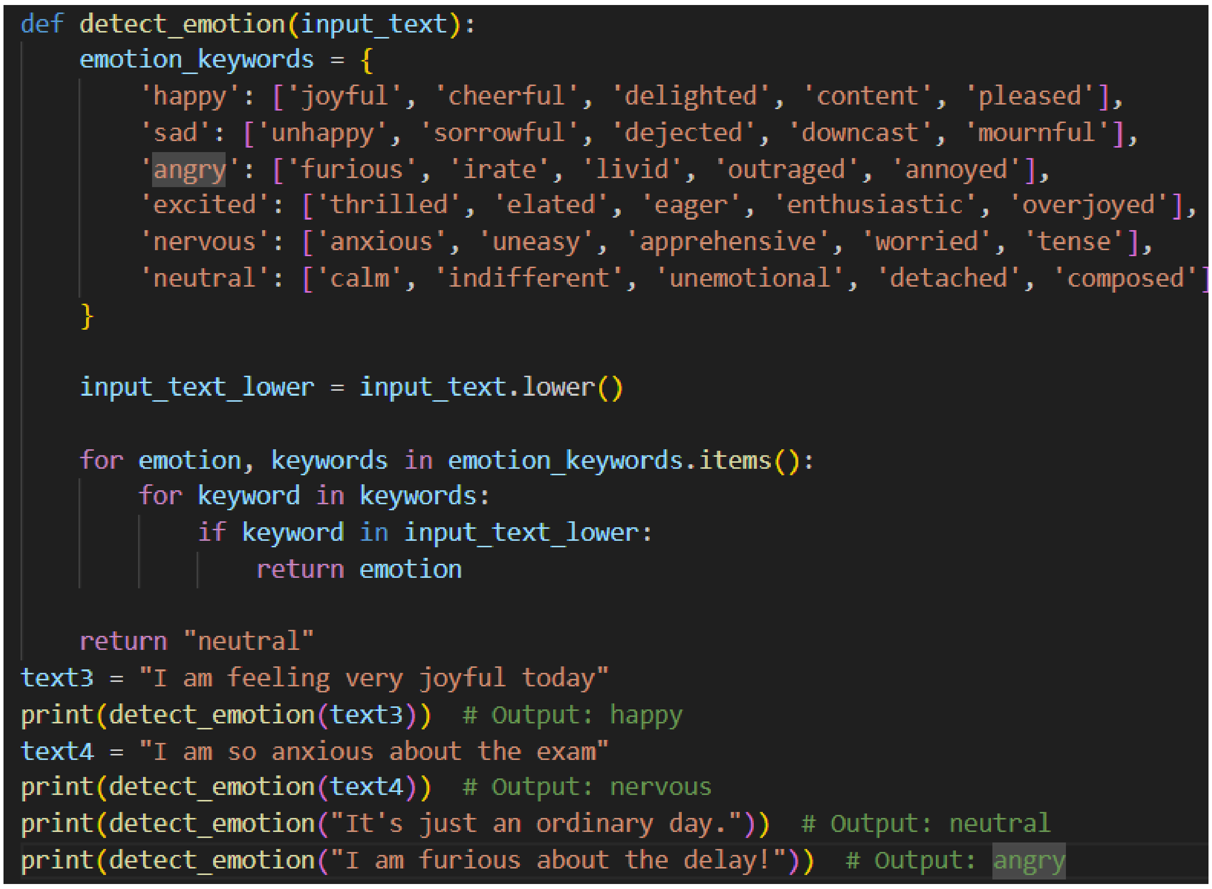
3. Print the predicted emotion

Prompt:



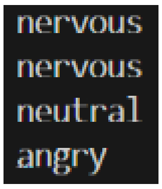
```
"""Example:
Input: "I am feeling very joyful today"
Output: happy
Now write a Python program that detects the emotion of a given sentence.
Possible emotions are happy, sad, angry, excited, nervous, and neutral.
Print only the predicted emotion."""
```

Code:



```python
def detect_emotion(input_text):
    emotion_keywords = {
        'happy': ['joyful', 'cheerful', 'delighted', 'content', 'pleased'],
        'sad': ['unhappy', 'sorrowful', 'dejected', 'downcast', 'mournful'],
        'angry': ['furious', 'irate', 'livid', 'outraged', 'annoyed'],
        'excited': ['thrilled', 'elated', 'eager', 'enthusiastic', 'overjoyed'],
        'nervous': ['anxious', 'uneasy', 'apprehensive', 'worried', 'tense'],
        'neutral': ['calm', 'indifferent', 'unemotional', 'detached', 'composed']
    }

    input_text_lower = input_text.lower()

    for emotion, keywords in emotion_keywords.items():
        for keyword in keywords:
            if keyword in input_text_lower:
                return emotion

    return "neutral"
text3 = "I am feeling very joyful today"
print(detect_emotion(text3))  # Output: happy
text4 = "I am so anxious about the exam"
print(detect_emotion(text4))  # Output: nervous
print(detect_emotion("It's just an ordinary day."))  # Output: neutral
print(detect_emotion("I am furious about the delay!"))  # Output: angry
```

Output:



```
nervous
nervous
neutral
angry
```

Justification:

Providing one labeled example helped the AI understand how emotions should be classified. The generated program correctly matched keywords to emotions and produced clear output. Compared to zero-shot prompting, one-shot prompting improved accuracy and output consistency.

Q3. 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

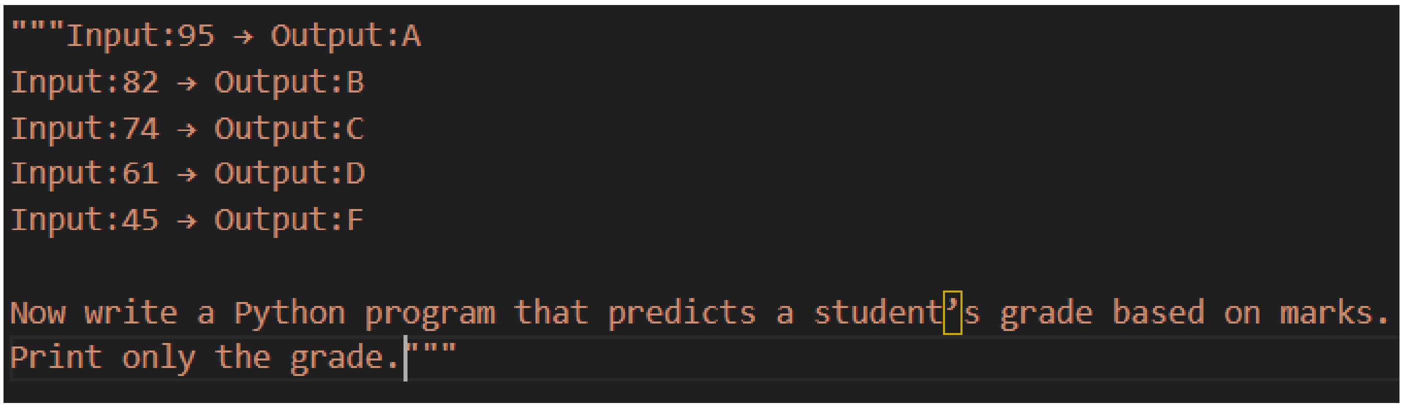
• 80–89 → B

• 70–79 → C

• 60–69 → D

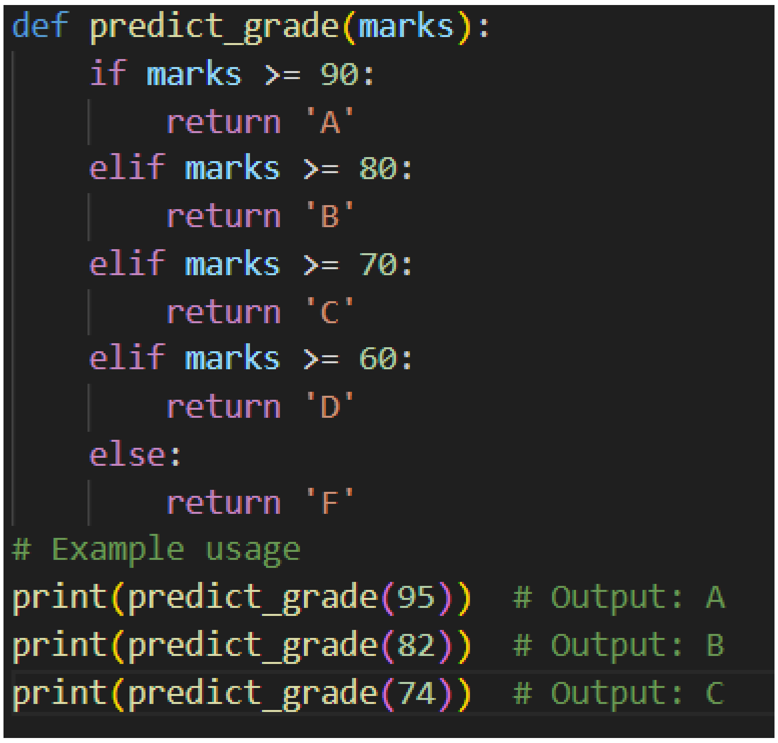
• Below 60 → F

Prompt:



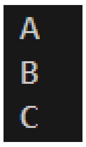
```
"""Input:95 → Output:A
Input:82 → Output:B
Input:74 → Output:C
Input:61 → Output:D
Input:45 → Output:F

Now write a Python program that predicts a student’s grade based on marks.
Print only the grade."""
```

Code:



```python
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'
# Example usage
print(predict_grade(95))  # Output: A
print(predict_grade(82))  # Output: B
print(predict_grade(74))  # Output: C
```

Output:



```
A
B
C
```

Justification:

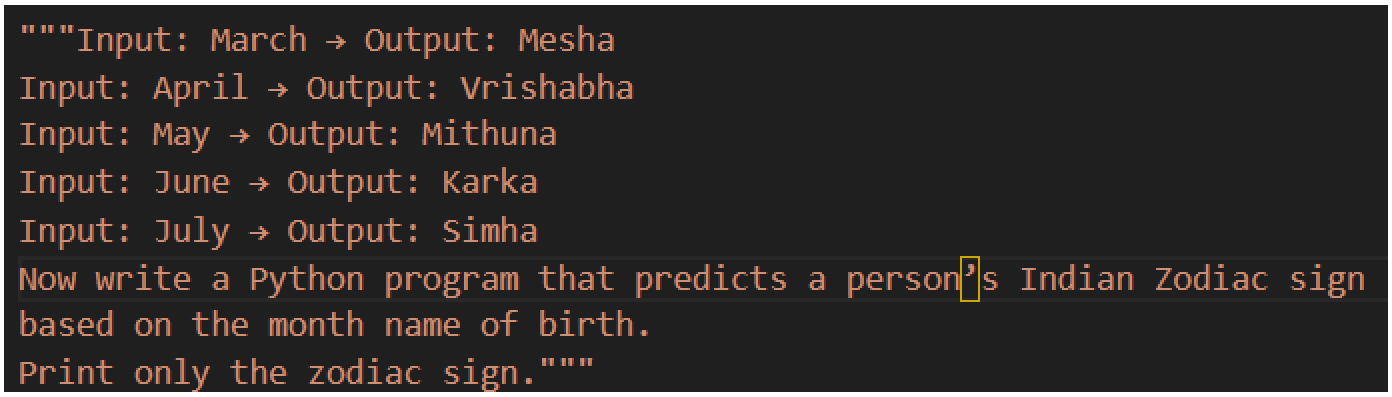
Providing multiple input-output examples helped the AI clearly understand the grading pattern. The generated code correctly inferred the grade ranges without explicitly stating them again. Few-shot prompting improved accuracy and reduced ambiguity compared to zero-shot or one-shot prompts. The output was clear and matched the expected grading criteria.

Q4. 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:

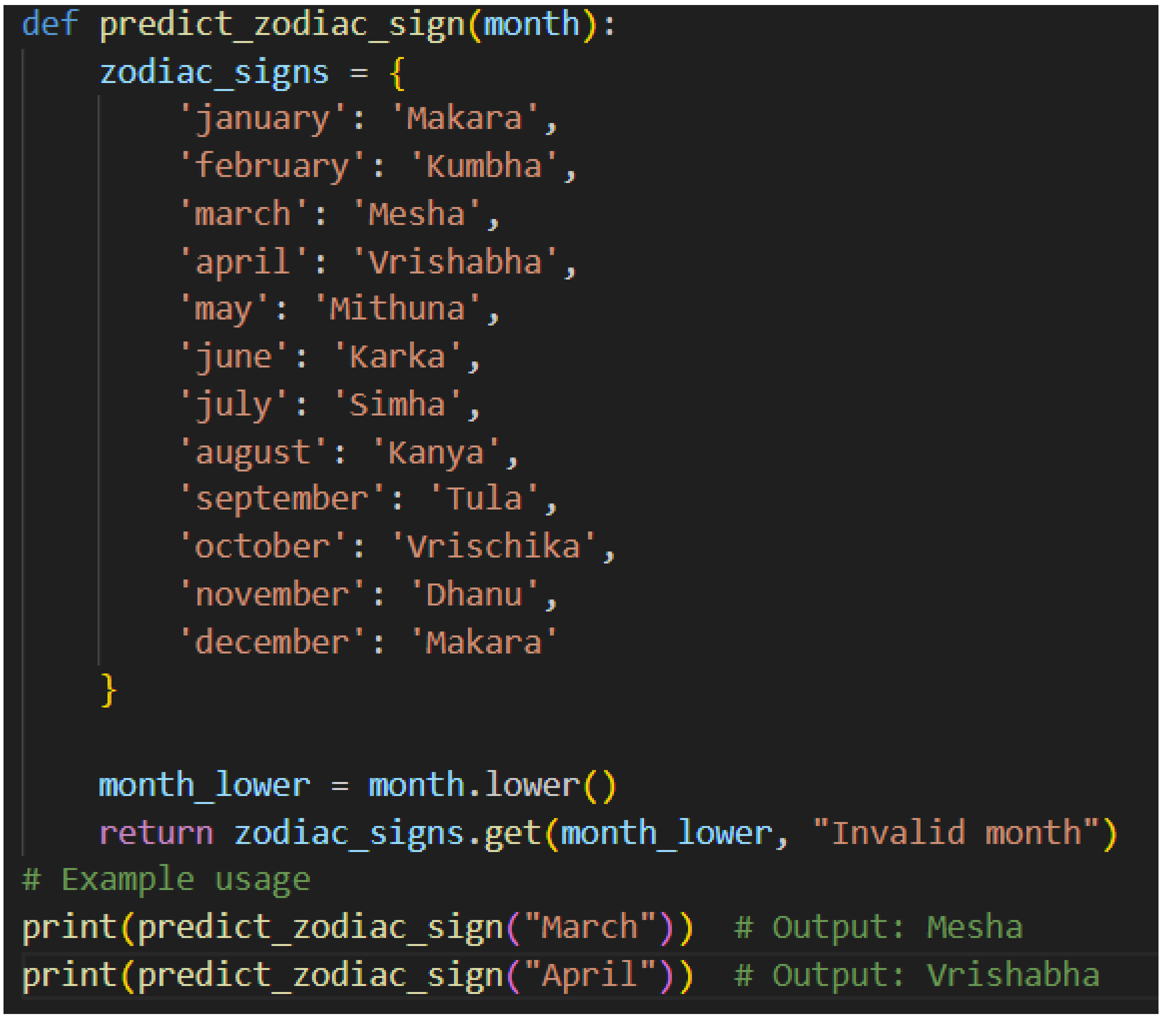
March → Mesha April → Vrishabha May → Mithuna June → Karka July → Simha August → Kanya September → Tula October → Vrischika November → Dhanu December → Makara January → Kumbha February → Meena

Prompt:



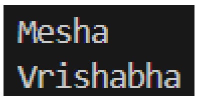
```
"""Input: March → Output: Mesha
Input: April → Output: Vrishabha
Input: May → Output: Mithuna
Input: June → Output: Karka
Input: July → Output: Simha
Now write a Python program that predicts a person’s Indian Zodiac sign
based on the month name of birth.
Print only the zodiac sign."""
```

Code:



```python
def predict_zodiac_sign(month):
    zodiac_signs = {
        'january': 'Makara',
        'february': 'Kumbha',
        'march': 'Mesha',
        'april': 'Vrishabha',
        'may': 'Mithuna',
        'june': 'Karka',
        'july': 'Simha',
        'august': 'Kanya',
        'september': 'Tula',
        'october': 'Vrischika',
        'november': 'Dhanu',
        'december': 'Makara'
    }

    month_lower = month.lower()
    return zodiac_signs.get(month_lower, "Invalid month")
# Example usage
print(predict_zodiac_sign("March"))  # Output: Mesha
print(predict_zodiac_sign("April"))  # Output: Vrishabha
```

Output:



```
Mesha
Vrishabha
```

Justification:

Providing multiple examples helped the AI clearly understand the month-to-rashi mapping pattern. The generated code followed the correct zodiac order starting from January. Multi-shot prompting improved accuracy and reduced confusion compared to fewer examples. The output was clear and matched the expected Indian zodiac sign.

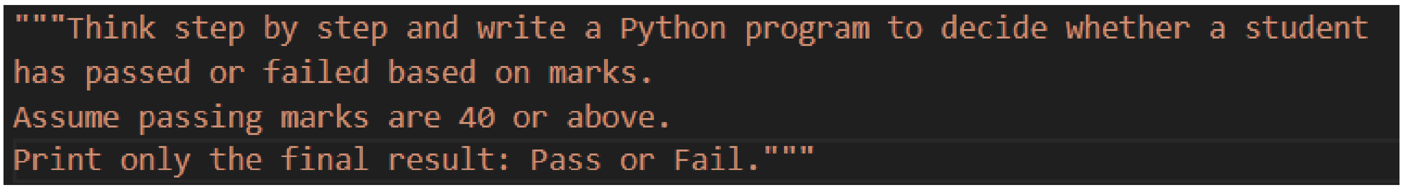
Q5. Result Analysis Based on Marks

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

Result Categories:

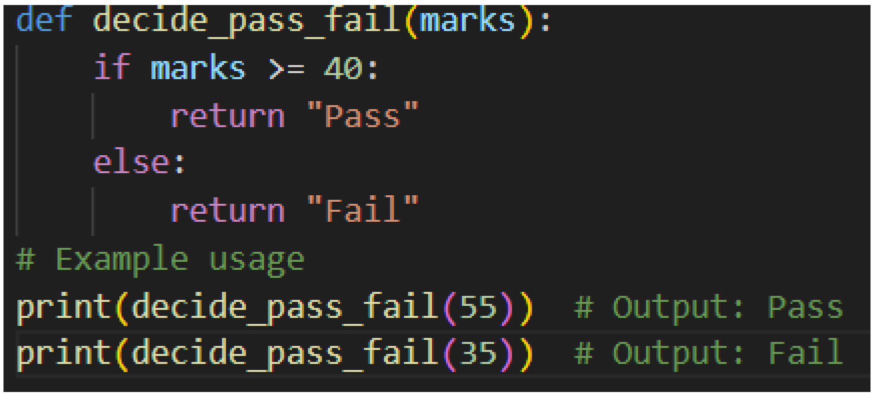
['Pass', 'Fail']

Prompt:



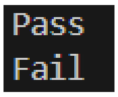
```
"""Think step by step and write a Python program to decide whether a student
has passed or failed based on marks.
Assume passing marks are 40 or above.
Print only the final result: Pass or Fail."""
```

Code:



```python
def decide_pass_fail(marks):
    if marks >= 40:
        return "Pass"
    else:
        return "Fail"
# Example usage
print(decide_pass_fail(55))  # Output: Pass
print(decide_pass_fail(35))  # Output: Fail
```

Output:



```
Pass
Fail
```

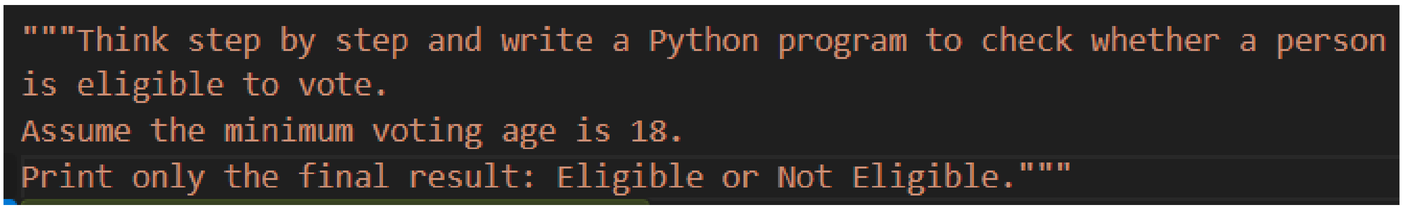
Justification:

Using chain-of-thought prompting helped the AI decide the condition before generating the final result. The code checks the pass condition and produces the correct output. Compared to simple prompting, this approach improved clarity in decision making.

Q6 Voting Eligibility Check (Chain-of-Thought Prompting)

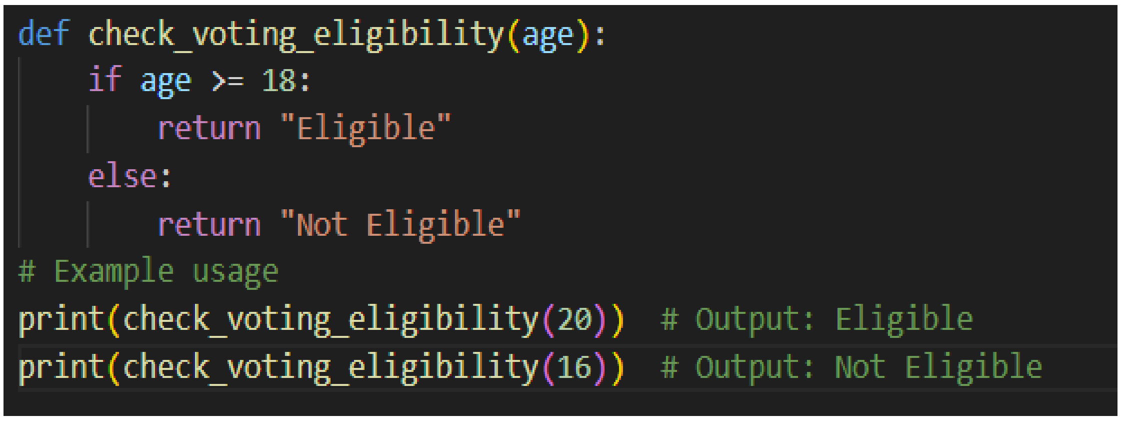
Task: Write a Python program that determines whether a person is eligible to vote using Chain-of-Thought (CoT) prompting.

Prompt:



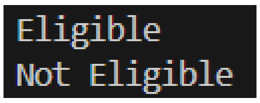
```
"""Think step by step and write a Python program to check whether a person
is eligible to vote.
Assume the minimum voting age is 18.
Print only the final result: Eligible or Not Eligible."""
```

Code:



```python
def check_voting_eligibility(age):
    if age >= 18:
        return "Eligible"
    else:
        return "Not Eligible"
# Example usage
print(check_voting_eligibility(20))  # Output: Eligible
print(check_voting_eligibility(16))  # Output: Not Eligible
```

Output:



```
Eligible
Not Eligible
```

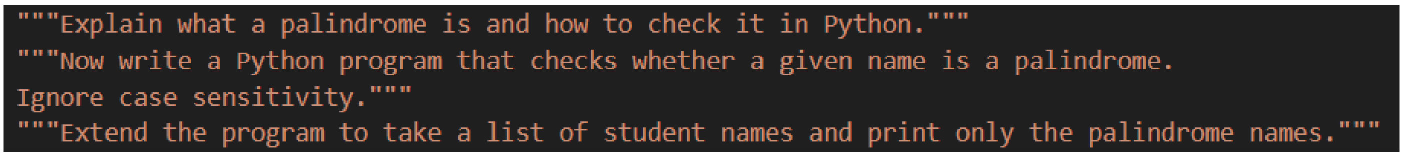
Justification:

Chain-of-thought prompting helped the AI clearly reason about the age condition before deciding eligibility. The generated code checks the minimum age logically and produces accurate results. The output is simple and easy to understand.

Q7 Prompt Chaining (String Processing – Palindrome Names)

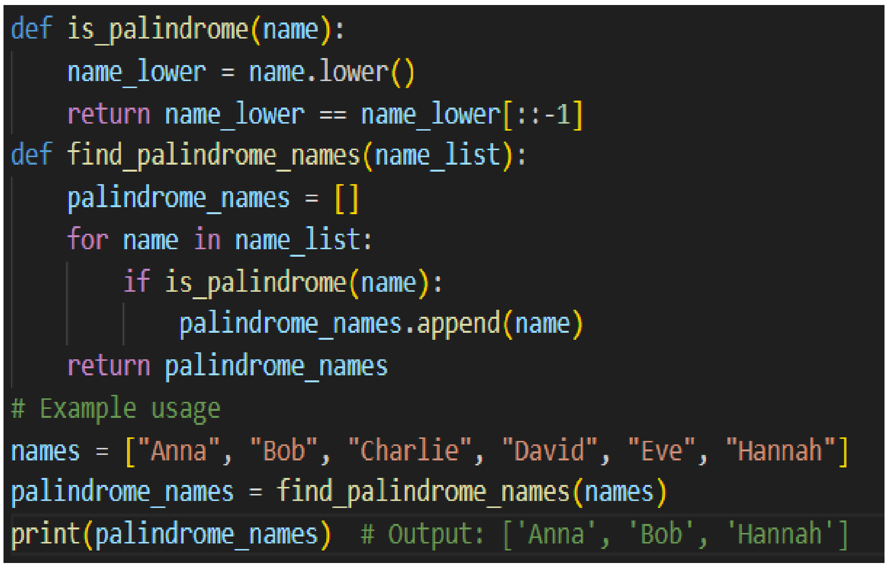
Task: Write a Python program that uses the prompt chaining technique to identify palindrome names from a list of student names.

Prompt:



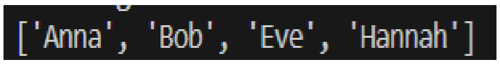
```
"""Explain what a palindrome is and how to check it in Python."""
"""Now write a Python program that checks whether a given name is a palindrome.
Ignore case sensitivity."""
"""Extend the program to take a list of student names and print only the palindrome names."""
```

Code:



```python
def is_palindrome(name):
    name_lower = name.lower()
    return name_lower == name_lower[::-1]
def find_palindrome_names(name_list):
    palindrome_names = []
    for name in name_list:
        if is_palindrome(name):
            palindrome_names.append(name)
    return palindrome_names
# Example usage
names = ["Anna", "Bob", "Charlie", "David", "Eve", "Hannah"]
palindrome_names = find_palindrome_names(names)
print(palindrome_names)  # Output: ['Anna', 'Bob', 'Hannah']
```

Output:



```
['Anna', 'Bob', 'Eve', 'Hannah']
```

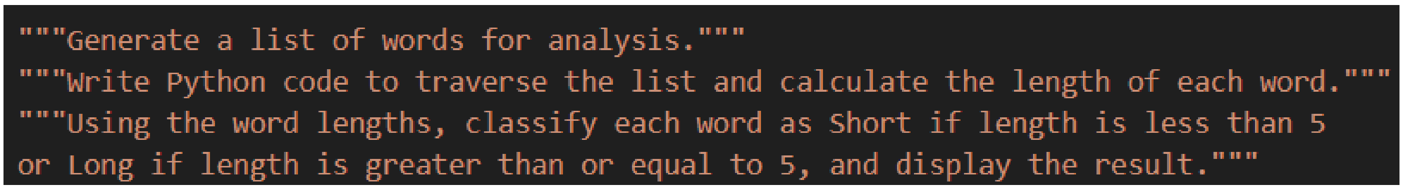
Justification:

Prompt chaining helped break the problem into smaller steps, making the logic easier to understand. The AI built the solution step by step. This improved accuracy and reduced confusion compared to a single prompt.

Q8 Prompt Chaining (String Processing – Word Length Analysis)

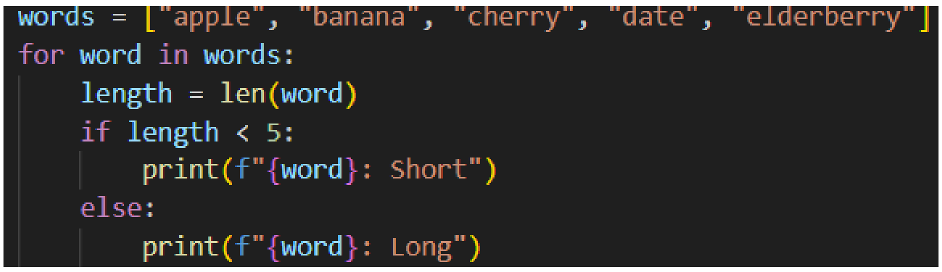
Task: Write a Python program that uses prompt chaining to analyze a list of words. In the first prompt, generate a list of words. In the second prompt, traverse the list and calculate the length of each word. In the third prompt, use the output of the previous step to determine whether each word is Short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

Prompt:



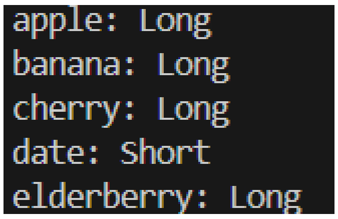
```
"""Generate a list of words for analysis."""
"""Write Python code to traverse the list and calculate the length of each word."""
"""Using the word lengths, classify each word as Short if length is less than 5
or Long if length is greater than or equal to 5, and display the result."""
```

Code:



```python
words = ["apple", "banana", "cherry", "date", "elderberry"]
for word in words:
    length = len(word)
    if length < 5:
        print(f"{word}: Short")
    else:
        print(f"{word}: Long")
```

Output:



```
apple: Long
banana: Long
cherry: Long
date: Short
elderberry: Long
```

Justification:

Prompt chaining helped divide the task into clear steps, making the logic easier to build. The AI first generated the data, then processed it, and finally classified the results. The final output clearly shows word lengths and their classification.