

Lab Assignment 4.1

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:

- Construct a prompt without any examples.
- Clearly specify the output labels.
- Display only the predicted label.

Input:

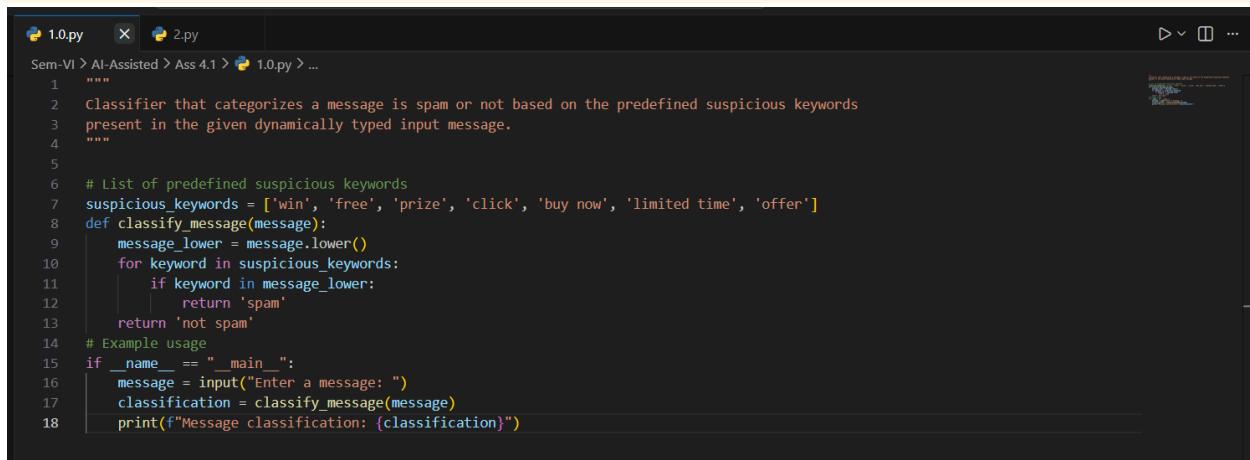
"Congratulations! You have won a free lottery ticket."

Expected Output:

Spam

Prompt Instructed:

Classifier that categorizes a message **is** spam **or not** based on the predefined suspicious keywords present **in** the given dynamically typed input message.



The screenshot shows a code editor window with a dark theme. The file is named '1.0.py'. The code defines a function 'classify_message' that checks if a given message contains any of a list of suspicious keywords ('win', 'free', 'prize', 'click', 'buy now', 'limited time', 'offer'). If any keyword is found, it returns 'spam'; otherwise, it returns 'not spam'. An example usage is shown at the bottom where the script is run from the command line.

```
1  """
2  Classifier that categorizes a message is spam or not based on the predefined suspicious keywords present in the given dynamically typed input message.
3  """
4
5
6  # List of predefined suspicious keywords
7  suspicious_keywords = ['win', 'free', 'prize', 'click', 'buy now', 'limited time', 'offer']
8  def classify_message(message):
9      message_lower = message.lower()
10     for keyword in suspicious_keywords:
11         if keyword in message_lower:
12             return 'spam'
13     return 'not spam'
14
15 # Example usage
16 if __name__ == "__main__":
17     message = input("Enter a message: ")
18     classification = classify_message(message)
19     print(f"Message classification: {classification}")
```

Output:

Enter a message: You can win a car.

Message classification: spam

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 Instructed:

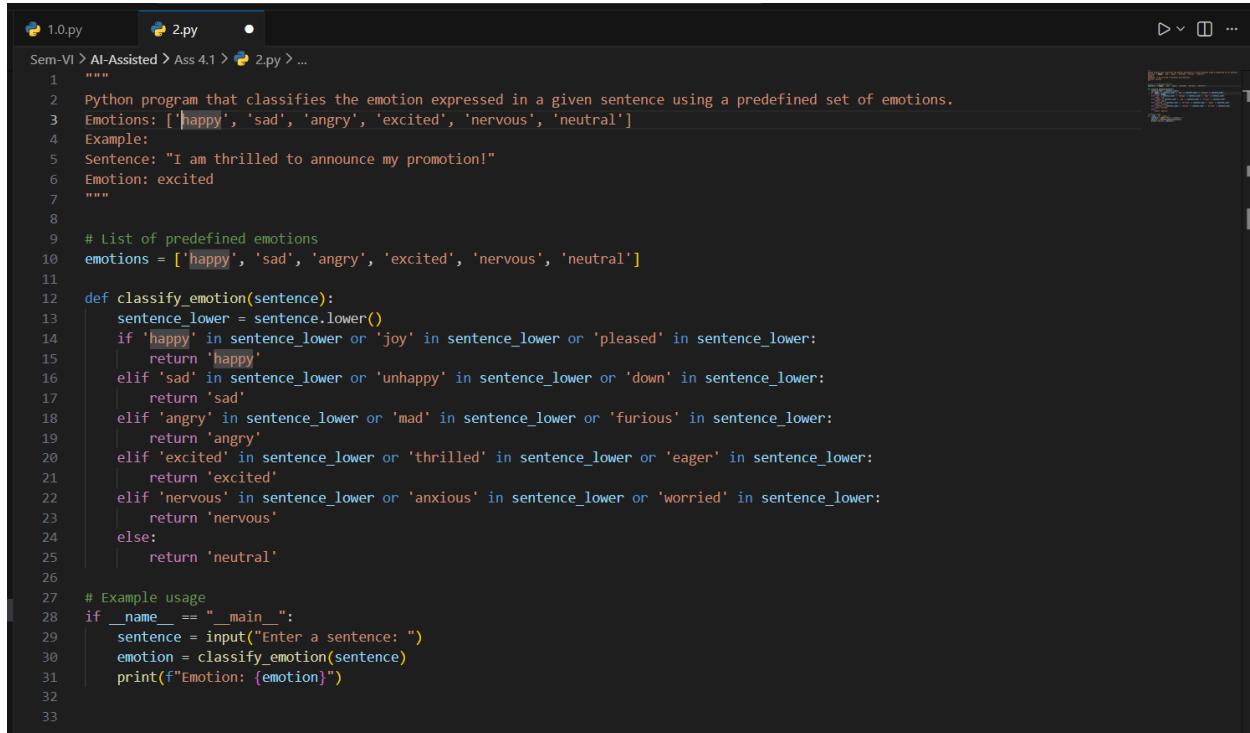
Python program that classifies the emotion expressed **in** a given sentence using a predefined set of emotions.

Emotions: ['happy', 'sad', 'angry', 'excited', 'nervous', 'neutral']

Example:

Sentence: "I am thrilled to announce my promotion!"

Emotion: excited



```

1.0.py 2.py
Sem-VI > AI-Assisted > Ass 4.1 > 2.py > ...
1 """
2 Python program that classifies the emotion expressed in a given sentence using a predefined set of emotions.
3 Emotions: ['happy', 'sad', 'angry', 'excited', 'nervous', 'neutral']
4 Example:
5 Sentence: "I am thrilled to announce my promotion!"
6 Emotion: excited
7 """
8
9 # List of predefined emotions
10 emotions = ['happy', 'sad', 'angry', 'excited', 'nervous', 'neutral']
11
12 def classify_emotion(sentence):
13     sentence_lower = sentence.lower()
14     if 'happy' in sentence_lower or 'joy' in sentence_lower or 'pleased' in sentence_lower:
15         return 'happy'
16     elif 'sad' in sentence_lower or 'unhappy' in sentence_lower or 'down' in sentence_lower:
17         return 'sad'
18     elif 'angry' in sentence_lower or 'mad' in sentence_lower or 'furious' in sentence_lower:
19         return 'angry'
20     elif 'excited' in sentence_lower or 'thrilled' in sentence_lower or 'eager' in sentence_lower:
21         return 'excited'
22     elif 'nervous' in sentence_lower or 'anxious' in sentence_lower or 'worried' in sentence_lower:
23         return 'nervous'
24     else:
25         return 'neutral'
26
27 # Example usage
28 if __name__ == "__main__":
29     sentence = input("Enter a sentence: ")
30     emotion = classify_emotion(sentence)
31     print(f"Emotion: {emotion}")
32
33

```

Input Sentence and Output:

Enter a sentence: I am mad at your work

Emotion: angry

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 Instructed:

Student grade calculator based on marks obtained

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

Example:

Input:

`marks = 85`

Output:

Grade: B

Input:

`marks = 72`

Output:

Grade: C

Input:

`marks = 58`

Output:

Grade: F

```

1.0.py 3.py 2.py
Sem-VI > AI-Assisted > Ass 4.1 > 3.py > ...
1 """
2 Student grade calculator based on marks obtained
3 Grades:
4 ['A', 'B', 'C', 'D', 'F']
5 Grading Criteria (to be inferred from examples):
6 90-100 → A
7 80-89 → B
8 70-79 → C
9 60-69 → D
10 Below 60 → F
11 Example:
12 Input:
13 marks = 85
14 Output:
15 Grade: B
16 Input:
17 marks = 72
18 Output:
19 Grade: C
20 Input:
21 marks = 58
22 Output:
23 Grade: F
24 """
25
26 def calculate_grade(marks):
27     if 90 <= marks <= 100:
28         return 'A'
29     elif 80 <= marks < 90:
30         return 'B'
31     elif 70 <= marks < 80:
32         return 'C'
33     elif 60 <= marks < 70:
34         return 'D'
35     else:
36         return 'F'
37 # Example usage
38 if __name__ == "__main__":
39     marks = int(input("Enter marks obtained (0-100): "))
40     grade = calculate_grade(marks)
41     print(f"Grade: {grade}")

```

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:

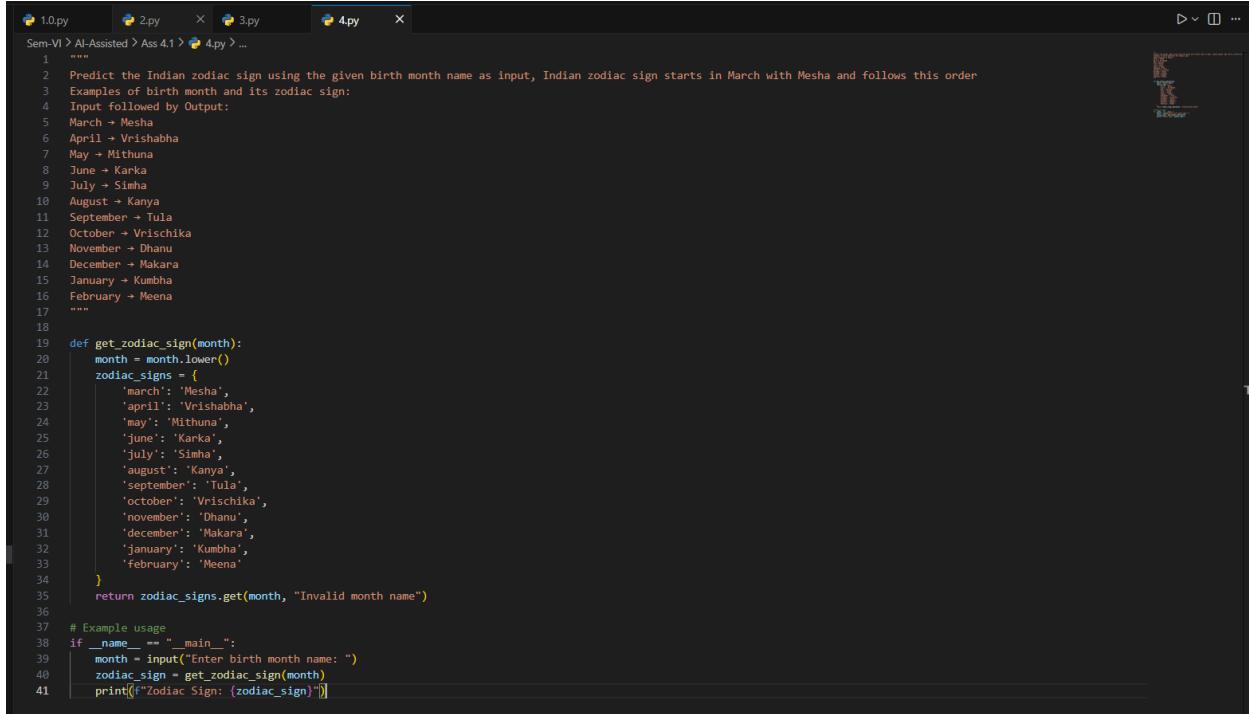
March → Mesha
April → Vrishabha
May → Mithuna
June → Karka
July → Simha
August → Kanya
September → Tula
October → Vrischika
November → Dhanu
December → Makara
January → Kumbha
February → Meena

Predict the Indian zodiac sign using the given birth month name as input,
Indian zodiac sign starts in March with Mesha and follows this order

Examples of birth month and its zodiac sign:

Input followed by Output:

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



```

1 """
2 Predict the Indian zodiac sign using the given birth month name as input, Indian zodiac sign starts in March with Mesha and follows this order
3 Examples of birth month and its zodiac sign:
4 Input followed by Output:
5 March → Mesha
6 April → Vrishabha
7 May → Mithuna
8 June → Karka
9 July → Simha
10 August → Kanya
11 September → Tula
12 October → Vrischika
13 November → Dhanu
14 December → Makara
15 January → Kumbha
16 February → Meena
17 """
18
19 def get_zodiac_sign(month):
20     month = month.lower()
21     zodiac_signs = {
22         'march': 'Mesha',
23         'april': 'Vrishabha',
24         'may': 'Mithuna',
25         'june': 'Karka',
26         'july': 'Simha',
27         'august': 'Kanya',
28         'september': 'Tula',
29         'october': 'Vrischika',
30         'november': 'Dhanu',
31         'december': 'Makara',
32         'january': 'Kumbha',
33         'february': 'Meena'
34     }
35     return zodiac_signs.get(month, "Invalid month name")
36
37 # Example usage
38 if __name__ == "__main__":
39     month = input("Enter birth month name: ")
40     zodiac_sign = get_zodiac_sign(month)
41     print(f"Zodiac Sign: {zodiac_sign}")

```

Q5. Result Analysis Based on Marks

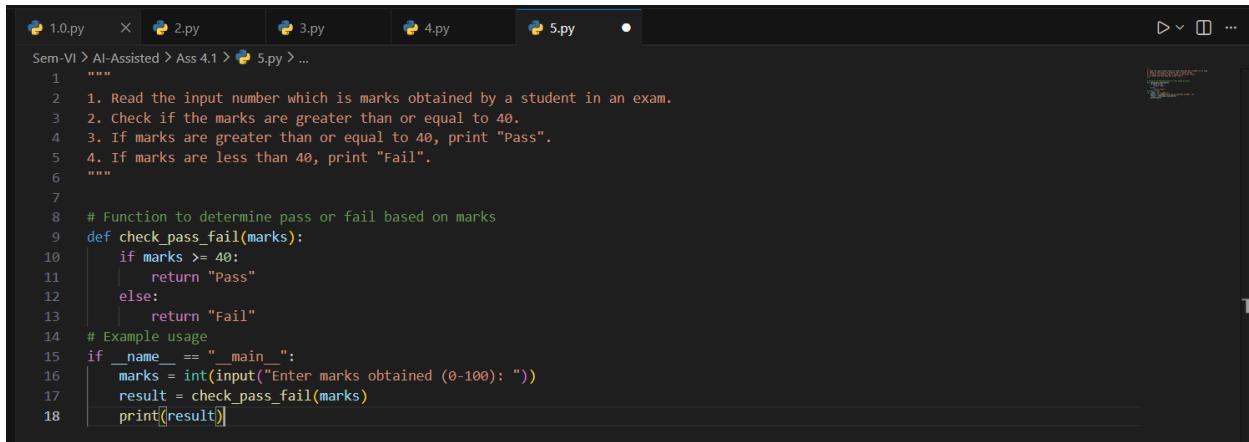
Task:

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 Instructed:



```

1 """
2 1. Read the input number which is marks obtained by a student in an exam.
3 2. Check if the marks are greater than or equal to 40.
4 3. If marks are greater than or equal to 40, print "Pass".
5 4. If marks are less than 40, print "Fail".
6 """
7
8 # Function to determine pass or fail based on marks:
9 def check_pass_fail(marks):
10     if marks >= 40:
11         return "Pass"
12     else:
13         return "Fail"
14 # Example usage
15 if __name__ == "__main__":
16     marks = int(input("Enter marks obtained (0-100): "))
17     result = check_pass_fail(marks)
18     print(result)

```

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 Instructed:

1. Read the input number which is age of a person and store it in a variable.
2. Check if the age is greater than or equal to 18.
3. If age is greater than or equal to 18, print "Eligible to vote".
4. If age is less than 18, print "Not eligible to vote".

The screenshot shows a terminal window with several tabs at the top labeled 1.0.py, 2.py, 3.py, 4.py, 5.py, 6.py, and The current tab is 6.py. The code in the terminal is as follows:

```
1.0.py  x  2.py  x  3.py  x  4.py  x  5.py  x  6.py  x  ...

Sem-VI > AI-Assisted > Ass 4.1 > 6.py > ...
1 """
2     1. Read the input number which is age of a person and store it in a variable.
3     2. Check if the age is greater than or equal to 18.
4     3. If age is greater than or equal to 18, print "Eligible to vote".
5     4. If age is less than 18, print "Not eligible to vote".
6 """
7
8 # Function to determine voting eligibility based on age
9 def check_voting_eligibility(age):
10     if age >= 18:
11         return "Eligible to vote"
12     else:
13         return "Not eligible to vote"
14
15 # Example usage
16 if __name__ == "__main__":
17     age = int(input("Enter age: "))
18     result = check_voting_eligibility(age)
19     print(result)
```

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 Instructed:

Generate a list of names of students and store it in a variable.

Traverse the list while reversing each name from the list and store it in a list in another variable.

If the original name and reversed name are the same making sure they are not case sensitive, print "Palindrome" along with the name.

```

1  """
2  1. Generate a list of names of students and store it in a variable.
3  2. Traverse the list while reversing each name from the list and store it in another variable.
4  3. If the original name and reversed name are the same making sure they are not case sensitive,
5  | print "Palindrome" along with the name.
6 """
7
8 # Function to check for palindromic names
9 def find_palindromic_names(names):
10    palindromic_names = []
11    for name in names:
12        reversed_name = name[::-1]
13        if name.lower() == reversed_name.lower():
14            palindromic_names.append((name, "Palindrome"))
15    return palindromic_names
16 # Example usage
17 if __name__ == "__main__":
18     student_names = ["Anna", "Bob", "Cathy", "David", "Eve", "Hannah"]
19     palindromes = find_palindromic_names(student_names)
20     for name, status in palindromes:
21         print(f"{name}: {status}")

```

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:

1. Generate a list of words **and** store it **in** a variable.
2. Traverse **and** calculate the length of each word **in** the list **and** store it **in** another list.
3. Now traverse the list of lengths and **print** if the word **is** short (length < 5), **or** long (length >= 5).

```

1  """
2  1. Generate a list of words and store it in a variable.
3  2. Traverse and calculate the length of each word in the list and store it in another list.
4  3. Now traverse the list of lengths and print if the word is short (length < 5), or long (length >= 5).
5 """
6
7 # Function to classify words based on their lengths
8 def classify_words_by_length(words):
9    lengths = [len(word) for word in words]
10   classifications = []
11   for length in lengths:
12       if length < 5:
13           classifications.append("short")
14       else:
15           classifications.append("long")
16   return classifications
17 # Example usage
18 if __name__ == "__main__":
19     word_list = ["apple", "bat", "car", "dolphin", "egg", "fish", "grape"]
20     classifications = classify_words_by_length(word_list)
21     for word, classification in zip(word_list, classifications):
22         print(f"{word}: {classification}")
23

```