AI Assisted Coding
Lab - 4.1

Name : B.ROHITH
Batch : 02
Roll No : 2303A51882

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."

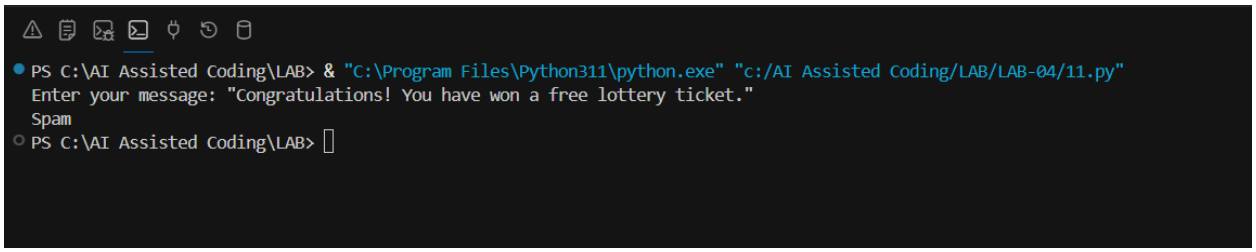
Expected Output:
Spam

Prompt:

Generate a Python program using a function that classifies a user-entered text message as Spam or Not Spam .using zero-shot style simple keyword logic.

Code:

Output:



Justification (Zero-Shot Prompting – Spam Classification)

This task follows zero-shot prompting because no example messages or sample classifications are provided; only clear instructions and output labels (“Spam” or “Not Spam”) are specified. The prompt directly asks the model to classify a message without prior demonstrations, which matches the zero-shot requirement.

The Python function classify_spam() applies simple keyword-based logic to simulate zero-shot classification and displays only the predicted label as output, satisfying the instruction to avoid extra explanations. When the input "*Congratulations! You have won a free lottery ticket.*" is given, the presence of spam-related keywords correctly results in the output Spam, meeting the expected output condition.

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’]

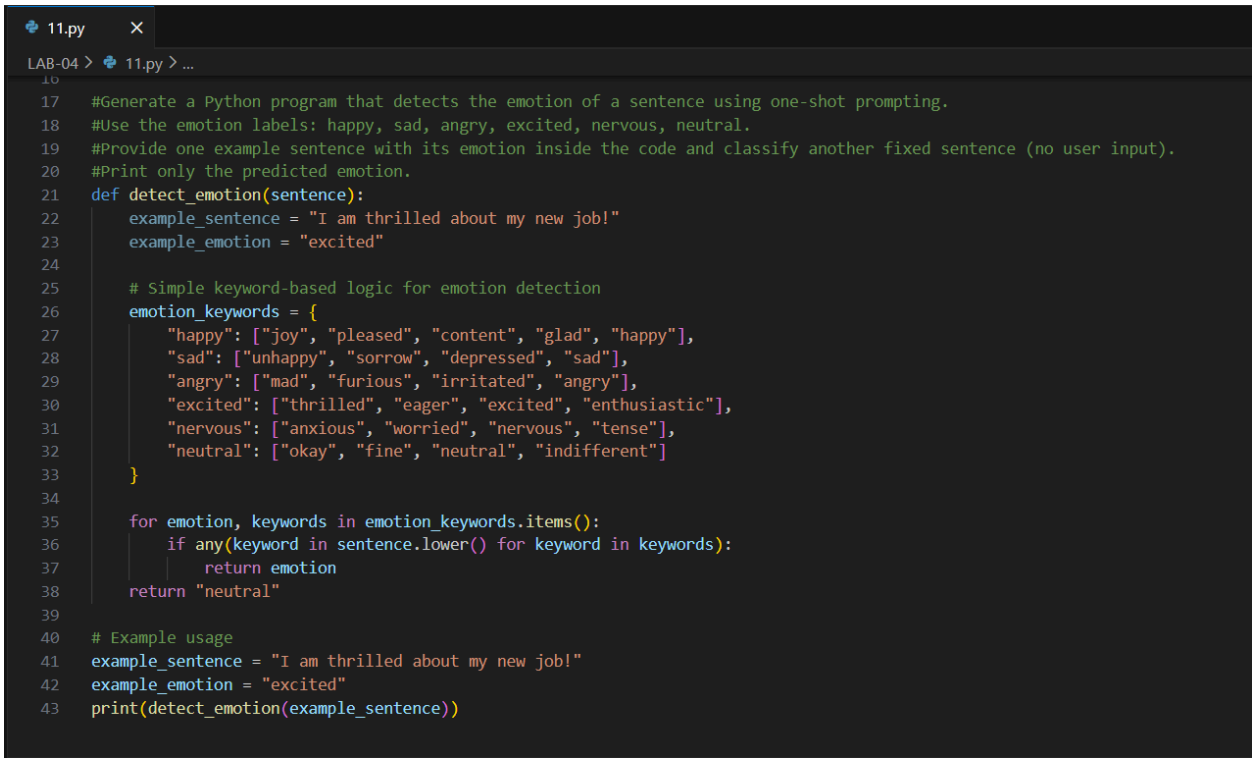
Steps:

1. Provide one labeled example inside the prompt.
2. Take a sentence as input.
3. Print the predicted emotion

Prompt:

Generate a Python program that detects the emotion of a sentence using one-shot prompting. Use the emotion labels: happy, sad, angry, excited, nervous, neutral. Provide one example sentence with its emotion inside the code and classify another fixed sentence (no user input). Print only the predicted emotion.

Code:



```python
#Generate a Python program that detects the emotion of a sentence using one-shot prompting.
#Use the emotion labels: happy, sad, angry, excited, nervous, neutral.
#Provide one example sentence with its emotion inside the code and classify another fixed sentence (no user input).
#Print only the predicted emotion.
def detect_emotion(sentence):
    example_sentence = "I am thrilled about my new job!"
    example_emotion = "excited"

    # Simple keyword-based logic for emotion detection
    emotion_keywords = {
        "happy": ["joy", "pleased", "content", "glad", "happy"],
        "sad": ["unhappy", "sorrow", "depressed", "sad"],
        "angry": ["mad", "furious", "irritated", "angry"],
        "excited": ["thrilled", "eager", "excited", "enthusiastic"],
        "nervous": ["anxious", "worried", "nervous", "tense"],
        "neutral": ["okay", "fine", "neutral", "indifferent"]
    }

    for emotion, keywords in emotion_keywords.items():
        if any(keyword in sentence.lower() for keyword in keywords):
            return emotion
    return "neutral"

# Example usage
example_sentence = "I am thrilled about my new job!"
example_emotion = "excited"
print(detect_emotion(example_sentence))
```

Output:

```
PS C:\AI Assisted Coding\LAB> & "C:\Program Files\Python311\python.exe" "c:/AI Assisted Coding/LAB/LAB-04/11.py"
excited
PS C:\AI Assisted Coding\LAB>
```

Justification (One-Shot Prompting – Emotion Detection)

This task follows one-shot prompting because one labeled example sentence (“*I am thrilled about my new job!*” → *excited*) is included inside the code to guide the emotion classification. The program then classifies a sentence using this prior example as reference, which satisfies the requirement of providing exactly one example before prediction.

The function analyzes the input sentence using simple keyword-based logic and prints only the predicted emotion label from the given set (happy, sad, angry, excited, nervous, neutral), meeting the output requirement. Thus, the implementation aligns with the

one-shot prompting approach and successfully detects the emotion as specified in the task.

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']

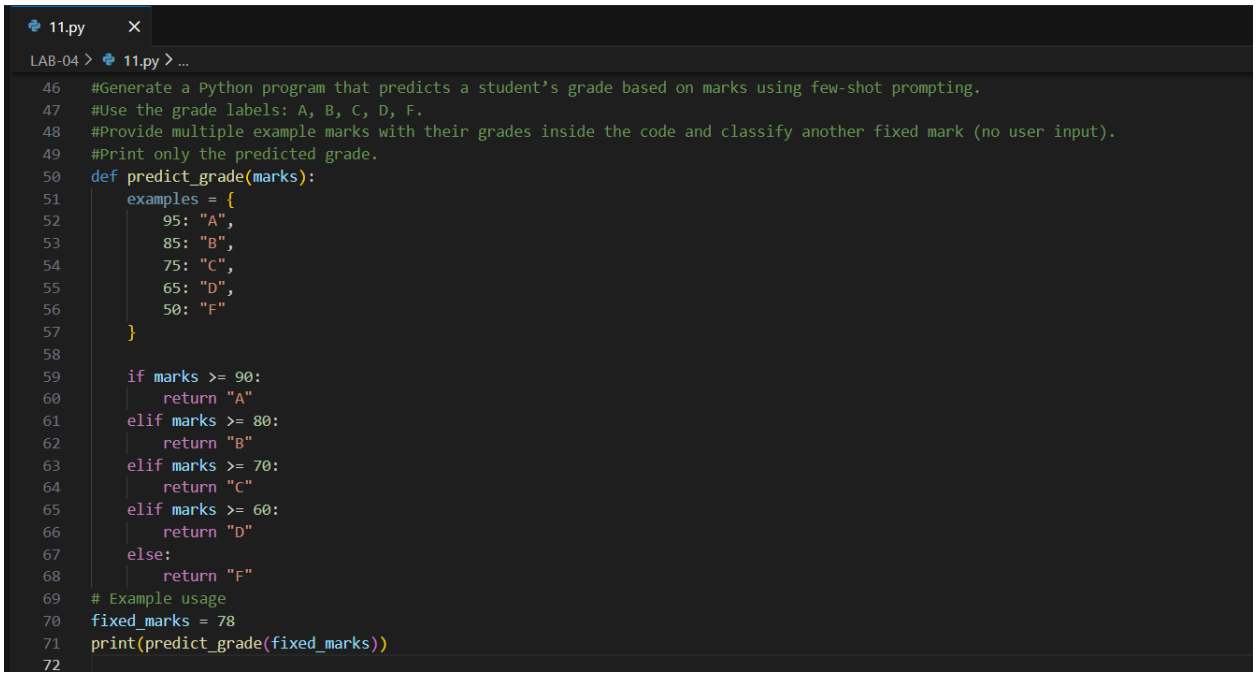
Grading Criteria (to be inferred from examples):

- 90–100 → A
- 80–89 → B
- 70–79 → C
- 60–69 → D
- Below 60 → F

Prompt:

Generate a Python program that predicts a student's grade based on marks using few-shot prompting. Use the grade labels: A, B, C, D, F. Provide multiple example marks with their grades inside the code and classify another fixed mark (no user input). Print only the predicted grade.

Code:

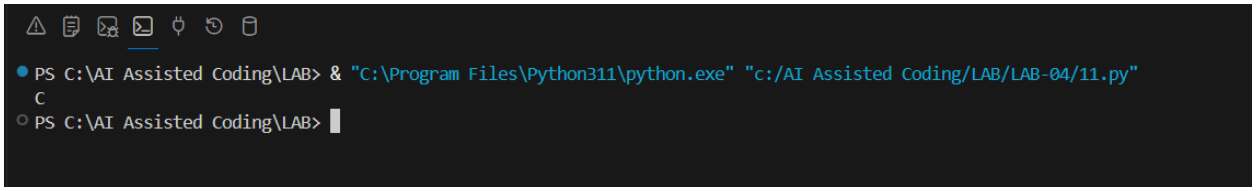


```python
#Generate a Python program that predicts a student's grade based on marks using few-shot prompting.
#Use the grade labels: A, B, C, D, F.
#Provide multiple example marks with their grades inside the code and classify another fixed mark (no user input).
#Print only the predicted grade.
def predict_grade(marks):
    examples = {
        95: "A",
        85: "B",
        75: "C",
        65: "D",
        50: "F"
    }

    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
fixed_marks = 78
print(predict_grade(fixed_marks))

```

Output:



```
PS C:\AI Assisted Coding\LAB> & "C:\Program Files\Python311\python.exe" "c:/AI Assisted Coding/LAB/LAB-04/11.py"
C
PS C:\AI Assisted Coding\LAB>
```

Justification (Few-Shot Prompting – Student Grading)

This task follows few-shot prompting because multiple labeled examples of marks and grades (95→A, 85→B, 75→C, 65→D, 50→F) are provided inside the code to demonstrate the grading pattern. From these examples, the grading criteria are inferred and applied to classify another fixed mark.

The program predicts the grade using only the allowed labels (A, B, C, D, F) and prints only the predicted grade, which satisfies the output requirement. Since more than one example is used to guide the prediction logic, the implementation correctly matches the few-shot prompting approach described in the task.

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.

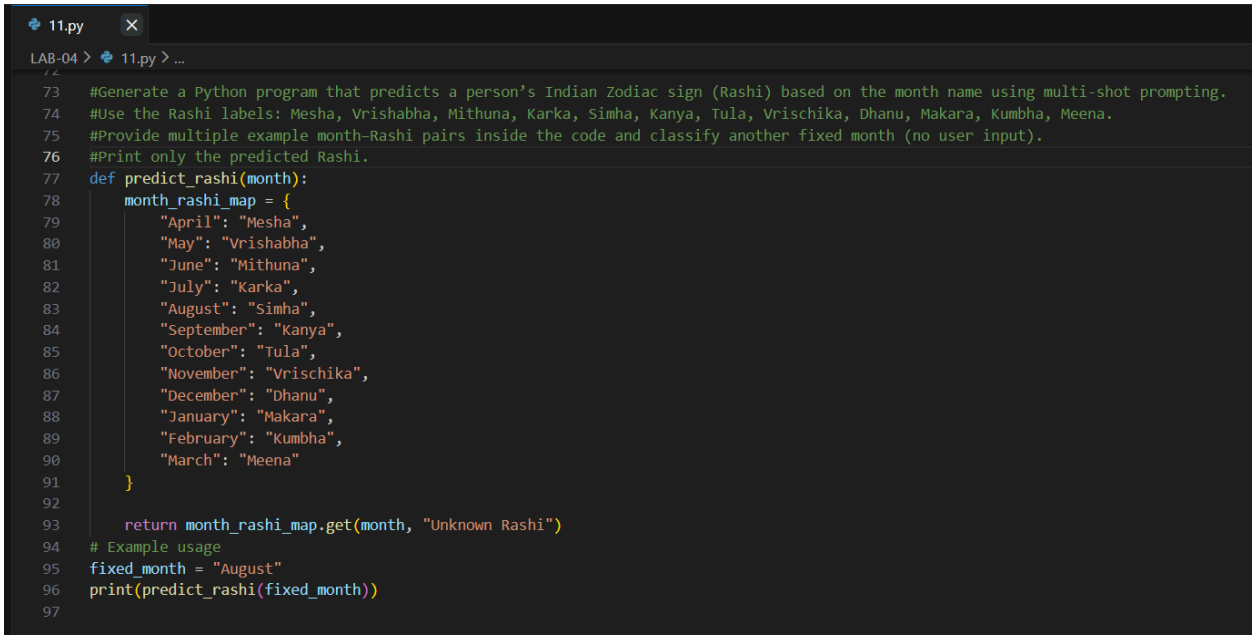
Indian Zodiac Order (Simplified Month-Based Model): The Indian Zodiac cycle starts in March with Mesha and follows this order:

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:

Generate a Python program that predicts a person's Indian Zodiac sign (Rashi) based on the month name using multi-shot prompting. Use the Rashi labels: Mesha, Vrishabha, Mithuna, Karka, Simha, Kanya, Tula, Vrischika, Dhanu, Makara, Kumbha, Meena. Provide multiple example month–Rashi pairs inside the code and classify another fixed month (no user input). Print only the predicted Rashi.

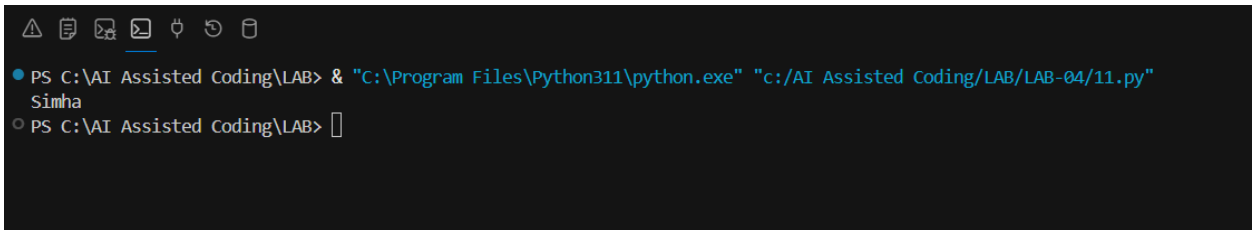
Code:



```python
#Generate a Python program that predicts a person's Indian Zodiac sign (Rashi) based on the month name using multi-shot prompting.
#Use the Rashi labels: Mesha, Vrishabha, Mithuna, Karka, Simha, Kanya, Tula, Vrischika, Dhanu, Makara, Kumbha, Meena.
#Provide multiple example month–Rashi pairs inside the code and classify another fixed month (no user input).
#Print only the predicted Rashi.
def predict_rashi(month):
    month_rashi_map = {
        "April": "Mesha",
        "May": "Vrishabha",
        "June": "Mithuna",
        "July": "Karka",
        "August": "Simha",
        "September": "Kanya",
        "October": "Tula",
        "November": "Vrischika",
        "December": "Dhanu",
        "January": "Makara",
        "February": "Kumbha",
        "March": "Meena"
    }

    return month_rashi_map.get(month, "Unknown Rashi")
# Example usage
fixed_month = "August"
print(predict_rashi(fixed_month))
```

Output:



```
PS C:\AI Assisted Coding\LAB> & "C:\Program Files\Python311\python.exe" "c:/AI Assisted Coding/LAB/LAB-04/11.py"
Simha
PS C:\AI Assisted Coding\LAB>
```

Justification (Multi-Shot Prompting – Indian Zodiac / Rashi Prediction)

This task follows multi-shot prompting because multiple example month–Rashi pairs are provided inside the code through the mapping dictionary, demonstrating several input–output patterns before predicting the result for another fixed month. These multiple examples help infer the zodiac sequence, which matches the idea of learning from many demonstrations.

The program uses month name as input, predicts the corresponding Indian Zodiac sign (Rashi) from the allowed labels, and prints only the predicted Rashi, satisfying the output

requirement. Hence, the implementation correctly aligns with the multi-shot prompting approach and fulfills all task conditions.

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.

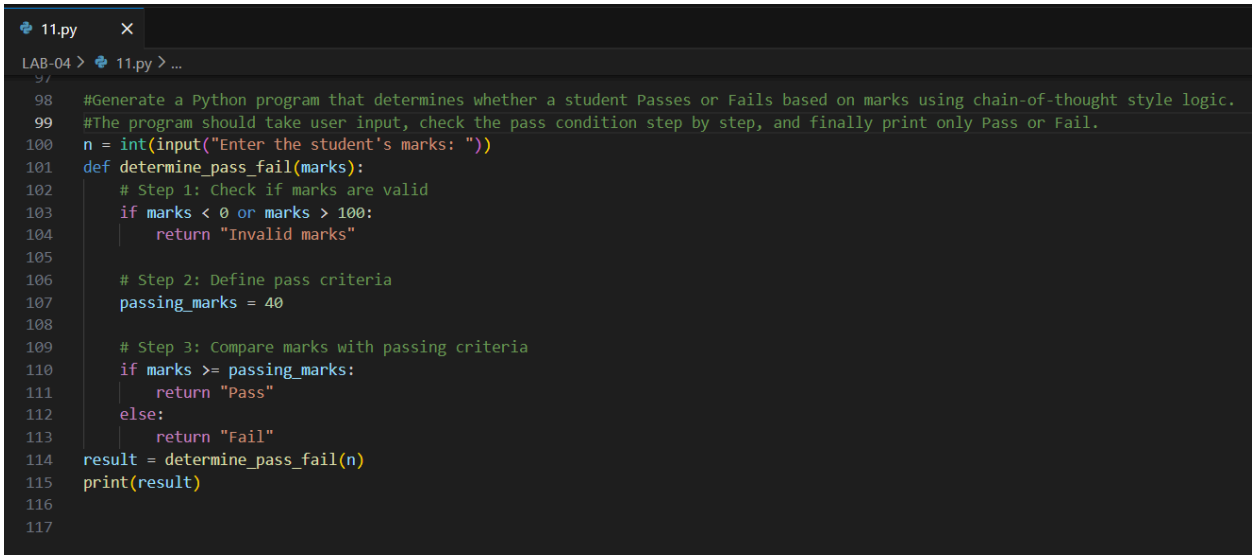
Result Categories:

['Pass', 'Fail']

Prompt:

Generate a Python program that determines whether a student Passes or Fails based on marks using chain-of-thought style logic. The program should take user input, check the pass condition step by step, and finally print only Pass or Fail.

Code:



Output:

Justification (Chain-of-Thought Prompting – Pass/Fail Result Analysis)

This task follows Chain-of-Thought (CoT) prompting because the program determines the result through step-by-step logical reasoning: first validating the marks, then defining the pass criteria, and finally comparing the marks to decide the outcome. This structured sequence reflects the CoT approach of reasoning before giving the final answer.

The program takes user input, evaluates it using intermediate decision steps, and outputs only the final category from the allowed labels (Pass or Fail), which fully satisfies the task requirements.

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:

Generate a Python program that determines whether a person is eligible to vote using chain-of-thought style logic. The program should take user age as input, verify eligibility through step-by-step checks, and finally print only Eligible or Not Eligible.

Code:

```python
#Generate a Python program that determines whether a person is eligible to vote using chain-of-thought style logic.
#The program should take user age as input, verify eligibility through step-by-step checks, and finally print only Eligible or Not Eligible.
age = int(input("Enter your age: "))
def check_voting_eligibility(age):
    # Step 1: Check if age is a valid number
    if age < 0:
        return "Invalid age"

    # Step 2: Define voting age criteria
    voting_age = 18

    # Step 3: Compare age with voting age criteria
    if age >= voting_age:
        return "Eligible"
    else:
        return "Not Eligible"
result = check_voting_eligibility(age)
print(result)
```

Output:

```
PS C:\AI Assisted Coding\LAB> & "C:\Program Files\Python311\python.exe" "c:/AI Assisted Coding/LAB/LAB-04/11.py"
Enter your age: 17
Not Eligible
PS C:\AI Assisted Coding\LAB> & "C:\Program Files\Python311\python.exe" "c:/AI Assisted Coding/LAB/LAB-04/11.py"
Enter your age: 23
Eligible
PS C:\AI Assisted Coding\LAB> & "C:\Program Files\Python311\python.exe" "c:/AI Assisted Coding/LAB/LAB-04/11.py"
Enter your age: 18
Eligible
PS C:\AI Assisted Coding\LAB>
```

Justification (Chain-of-Thought Prompting – Voting Eligibility Check)

This task follows Chain-of-Thought (CoT) prompting because the program determines eligibility through step-by-step logical reasoning: first validating the age, then defining the minimum voting age, and finally comparing the two to reach a decision. This structured flow reflects the CoT approach of reasoning before producing the final output.

The program takes user input (age) and prints only the final eligibility result (Eligible or Not Eligible), which satisfies the task requirement of producing a clear classification outcome.

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:

Generate a Python program that uses prompt chaining logic to identify palindrome names from a list of student names.

Step 1: Clean and normalize each name (convert to lowercase, remove spaces if needed).
Step 2: Check whether each processed name is a palindrome.

Step 3: Collect and print only the names that are palindromes.(user)

Code:

Output:

Justification (Prompt Chaining – Palindrome Names)

This program follows the prompt chaining technique by solving the problem through sequential dependent steps: first cleaning and normalizing each name, then checking whether the processed name is a palindrome, and finally collecting only the valid palindrome names. Each step uses the output of the previous step, which reflects the idea of chaining prompts or operations.

The program processes a list of student names and prints only the names that satisfy the palindrome condition, fully meeting the task requirement of identifying palindrome names using a structured, step-by-step approach.

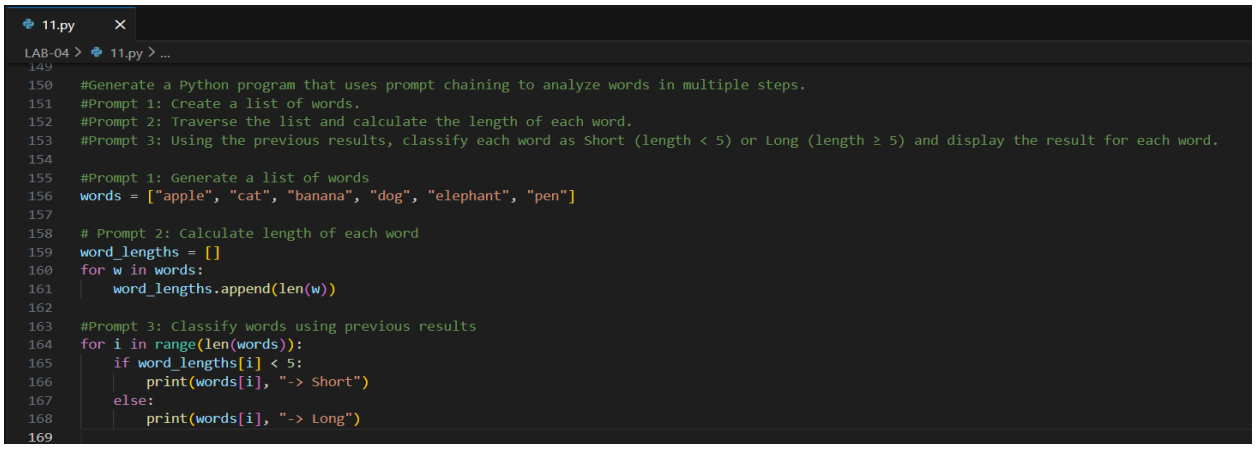
Q8 Prompt Chaining (String Processing – Word Length Analysis)

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.

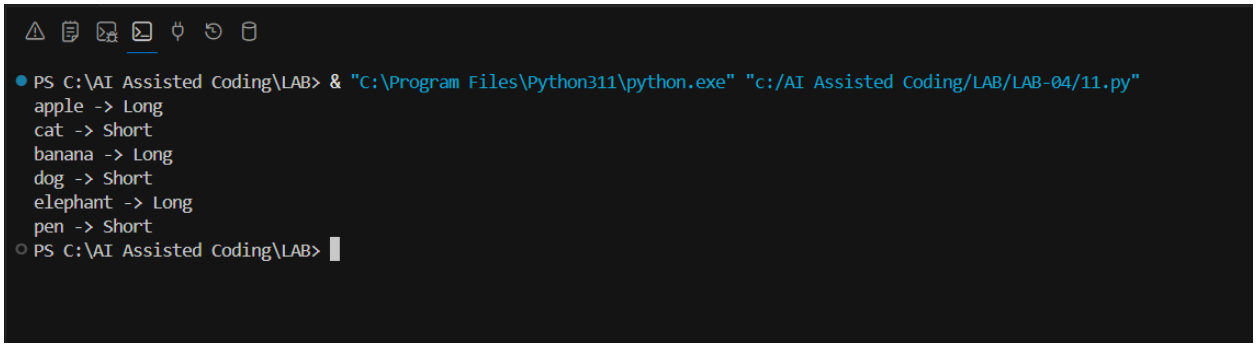
Prompt:

Generate a Python program that uses prompt chaining to analyze words in multiple steps.
Prompt 1: Create a list of words.
2: Traverse the list and calculate the length of each word.
3: Using the previous results, classify each word as Short (length < 5) or Long (length ≥ 5) and display the result for each word.

Code:



Output:



Justification (Prompt Chaining – Word Length Analysis)

This program follows prompt chaining by performing the task in multiple dependent steps: first generating a list of words, then calculating the length of each word, and finally using those lengths to classify each word as Short or Long. Each step relies on the output of the previous step, which reflects the core idea of chaining prompts or operations.

The final output correctly displays the classification for every word based on its length, fully satisfying the task requirement of analyzing words using a structured, step-by-step prompt chaining approach.