AI ASSISTED CODING

LAB ASSIGNMENT – 4.1

Name: A.Deepthi

Ht.No: 2303A52390

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.

Lab Outcomes (LOs):

After completing this lab, students will be able to:

- Use zero-shot prompting to instruct AI with minimal context.
- Use one-shot prompting with a single example to guide AI code generation.
- Apply few-shot prompting using multiple examples to improve AI responses.

- Compare AI outputs across the three prompting strategies.

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."

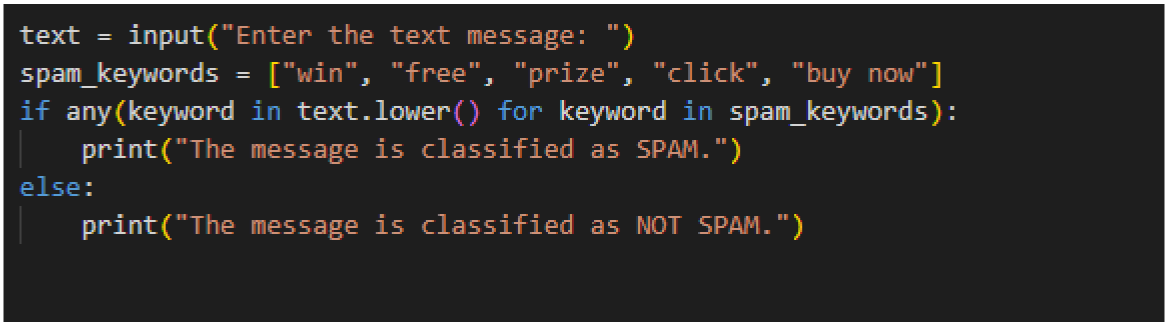
Expected Output:

Spam

Prompt:

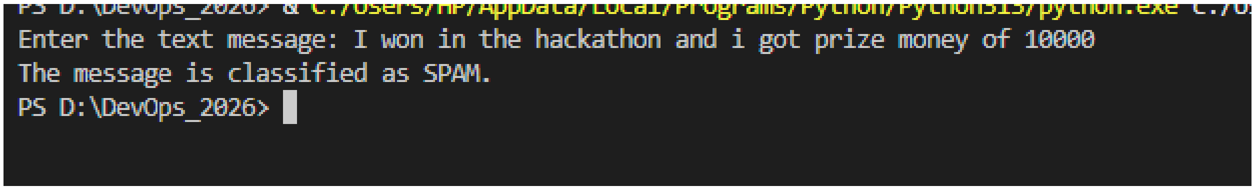
Classify the given text as spam or not spam take the text from user input

Code:



```python
text = input("Enter the text message: ")
spam_keywords = ["win", "free", "prize", "click", "buy now"]
if any(keyword in text.lower() for keyword in spam_keywords):
    print("The message is classified as SPAM.")
else:
    print("The message is classified as NOT SPAM.")
```

Output:



```
PS D:\DevOps_2026> & C:/Users/HP/AppData/Local/Programs/Python/Python313/python.exe c:/
Enter the text message: I won in the hackathon and i got prize money of 10000
The message is classified as SPAM.
PS D:\DevOps_2026> 
```

Explanation:

This program checks if any of the predefined spam keywords are present in the input text, it classifies the text as spam text if not the text is classified as not 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.

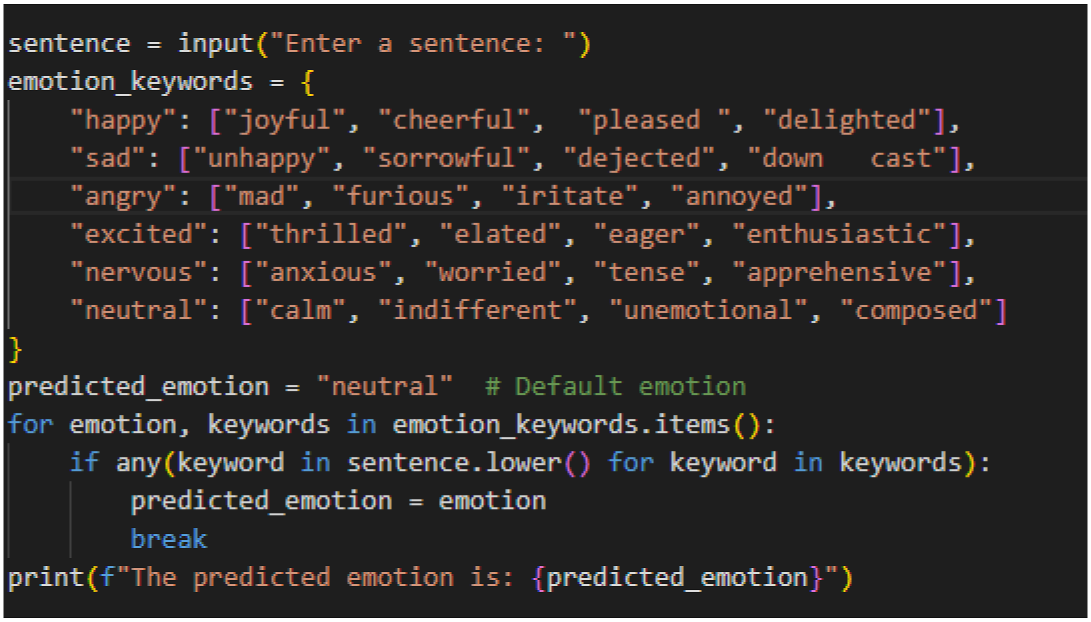
3. Print the predicted emotion

Prompt:

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.

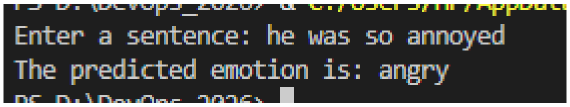
Take Input From User.

Input:



```python
sentence = input("Enter a sentence: ")
emotion_keywords = {
    "happy": ["joyful", "cheerful",  "pleased ", "delighted"],
    "sad": ["unhappy", "sorrowful", "dejected", "down   cast"],
    "angry": ["mad", "furious", "iritate", "annoyed"],
    "excited": ["thrilled", "elated", "eager", "enthusiastic"],
    "nervous": ["anxious", "worried", "tense", "apprehensive"],
    "neutral": ["calm", "indifferent", "unemotional", "composed"]
}
predicted_emotion = "neutral"  # Default emotion
for emotion, keywords in emotion_keywords.items():
    if any(keyword in sentence.lower() for keyword in keywords):
        predicted_emotion = emotion
        break
print(f"The predicted emotion is: {predicted_emotion}")
```

Output:



```
Enter a sentence: he was so annoyed
The predicted emotion is: angry
```

Explanation:

The program predicts the emotion of the sentence, which analyzes a given sentence to identify the underlying emotion. It first converts the input sentence to lowercase, ensuring consistency during keyword matching. The function then searches for specific keywords linked to emotions such as *happy, sad, angry, excited,* and *nervous*. If none of these keywords are detected, the function defaults to returning "neutral".

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):

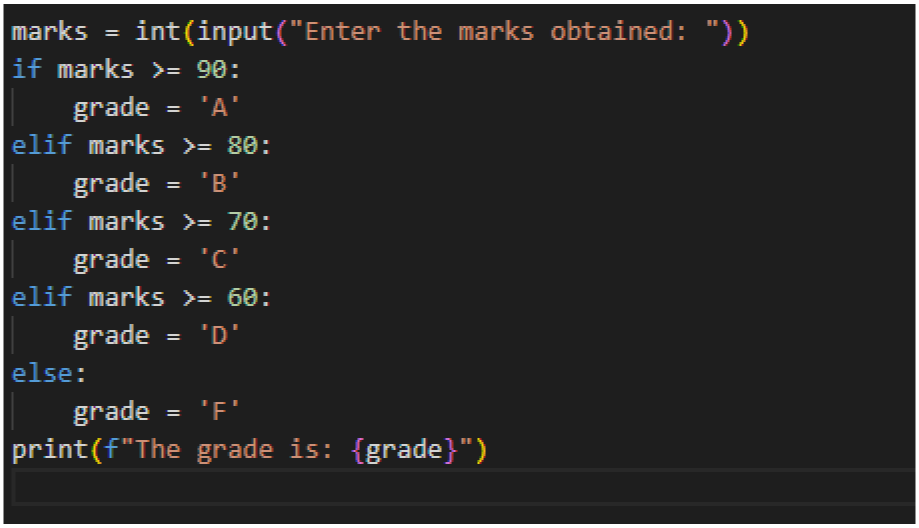
- 90–100 → A
- 80–89 → B
- 70–79 → C
- 60–69 → D
- Below 60 → F

Prompt:

Predict the student grade based on marks.
Possible grades are: A, B, C, D, F.
Examples:
Marks: 95
Grade: A

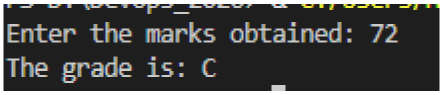
Marks: 82
Grade: B
Marks: 74
Grade: C
Marks: 61
Grade: D
Marks: 45
Grade: F
Print only the grade.
Take input from user

Code:



```
marks = int(input("Enter the marks obtained: "))
if marks >= 90:
    grade = 'A'
elif marks >= 80:
    grade = 'B'
elif marks >= 70:
    grade = 'C'
elif marks >= 60:
    grade = 'D'
else:
    grade = 'F'
print(f"The grade is: {grade}")
```

Output:



```
Enter the marks obtained: 72
The grade is: C
```

Explanation:

This program takes an integer input marks and returns a corresponding grade based on predefined thresholds.

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:

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.

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

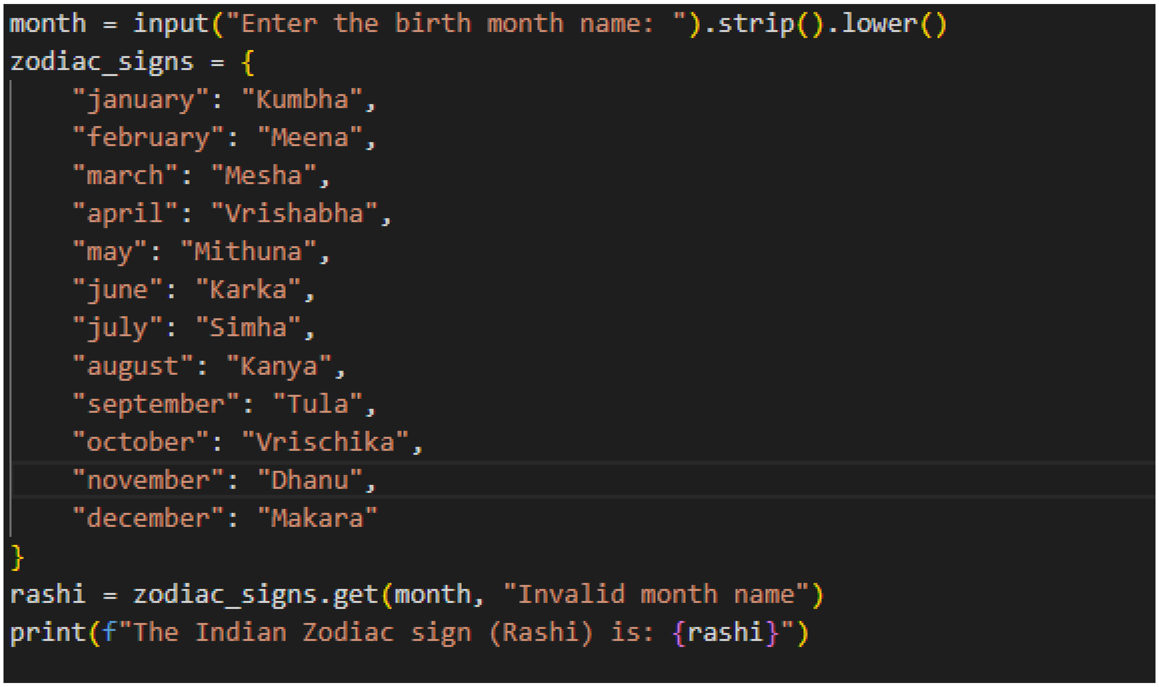
December → Makara

January → Kumbha

February → Meena

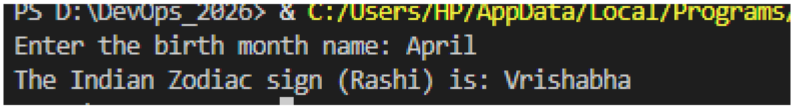
Print only the Rashi name.
Take Input From User.

Code:



```
month = input("Enter the birth month name: ").strip().lower()
zodiac_signs = {
    "january": "Kumbha",
    "february": "Meena",
    "march": "Mesha",
    "april": "Vrishabha",
    "may": "Mithuna",
    "june": "Karka",
    "july": "Simha",
    "august": "Kanya",
    "september": "Tula",
    "october": "Vrischika",
    "november": "Dhanu",
    "december": "Makara"
}
rashi = zodiac_signs.get(month, "Invalid month name")
print(f"The Indian Zodiac sign (Rashi) is: {rashi}")
```

Output:



```
PS D:\DevOps_2026> & C:/Users/HP/AppData/Local/Programs
Enter the birth month name: April
The Indian Zodiac sign (Rashi) is: Vrishabha
```

Explanation:

This program 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".

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.

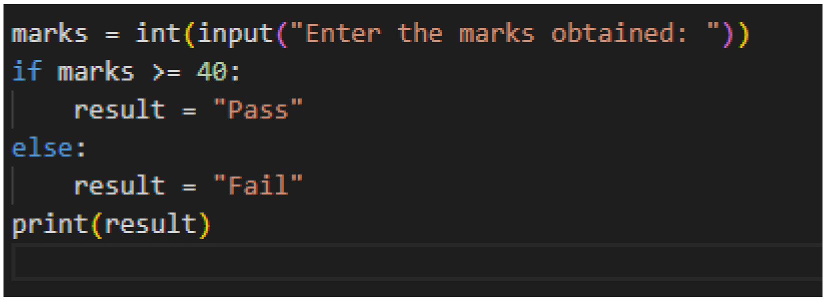
Result Categories:

['Pass', 'Fail']

Prompt:

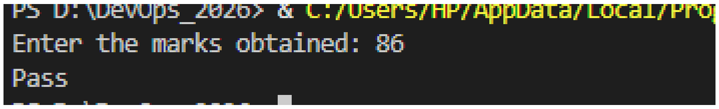
Determine whether the student Passes or Fails based on marks.
Think step by step before giving the final answer.
At the end, print only Pass or Fail.
Take input from user

Code:



```python
marks = int(input("Enter the marks obtained: "))
if marks >= 40:
    result = "Pass"
else:
    result = "Fail"
print(result)
```

Output:



```
PS D:\DevOps_2026> & C:/Users/HP/AppData/Local/Pro
Enter the marks obtained: 86
Pass
```

Explanation:

This program takes the marks obtained by a student as input and checks if the student has passed or failed in a subject 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".

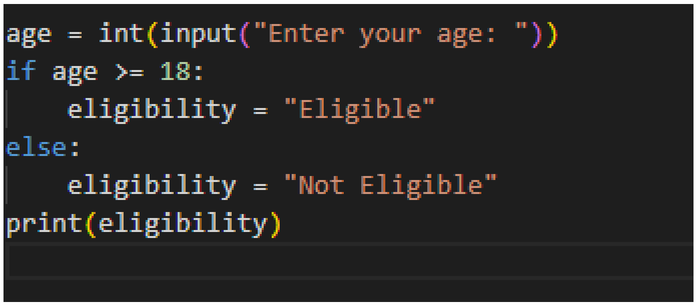
Q6. Voting Eligibility Check (Chain-of-Thought Prompting)

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

Prompt:

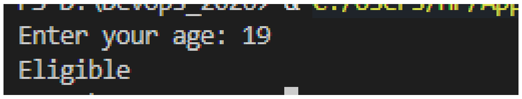
Determine whether a person is eligible to vote.
The minimum voting age is 18.
Reason step by step and then give the final answer.
Print only Eligible or Not Eligible. Take input from user

Code:



```python
age = int(input("Enter your age: "))
if age >= 18:
    eligibility = "Eligible"
else:
    eligibility = "Not Eligible"
print(eligibility)
```

Output:



```
Enter your age: 19
Eligible
```

Explanation:

This program 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".

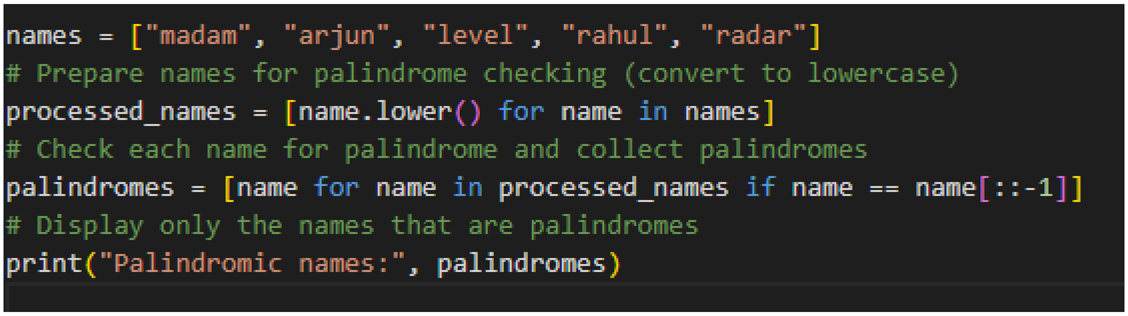
Q7. Prompt Chaining (String Processing – Palindrome Names)

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

Prompt:

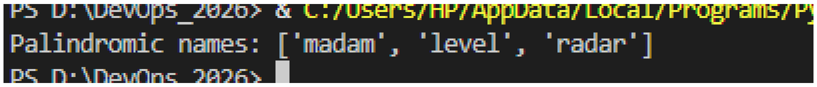
1. Given the following list of student names, prepare them for palindrome checking:
   Names: ["madam", "arjun", "level", "rahul", "radar"]
2. Using the processed list from the previous step, check each name and identify whether it is a palindrome.
3. From the previous results, display only the names that are palindromes.

Code:



```python
names = ["madam", "arjun", "level", "rahul", "radar"]
# Prepare names for palindrome checking (convert to lowercase)
processed_names = [name.lower() for name in names]
# Check each name for palindrome and collect palindromes
palindromes = [name for name in processed_names if name == name[::-1]]
# Display only the names that are palindromes
print("Palindromic names:", palindromes)
```

Output:



```
PS D:\DevOps_2026> & C:/Users/HP/AppData/Local/Programs/P
Palindromic names: ['madam', 'level', 'radar']
PS D:\DevOps_2026>
```

Explanation:

This program defines a list of student names and includes a function to check if a name is a palindrome (a word that reads the same backward as forward). It then filters the list to find all names that are palindromes and prints them. The use of list comprehension makes the filtering process concise and efficient.

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

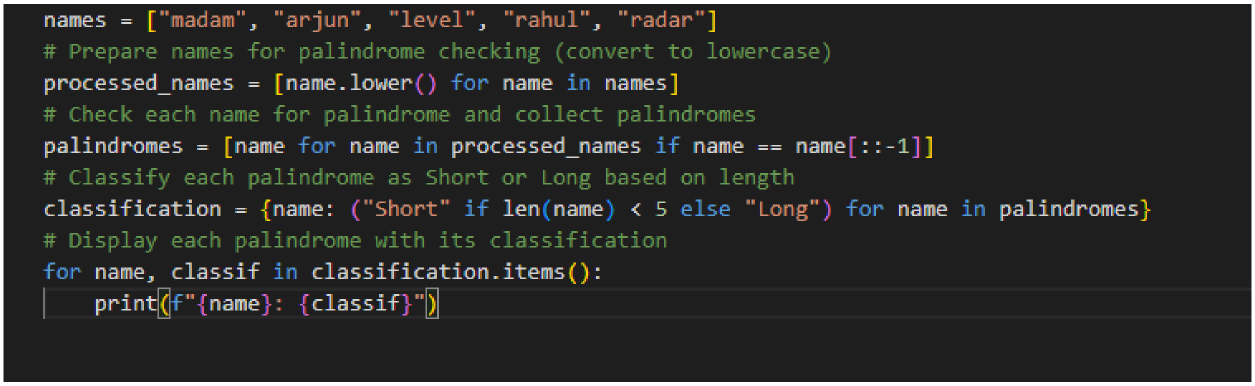
Task: Write a Python program that uses prompt chaining to analyze a list of words. In the first prompt, generate a list of words.

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:

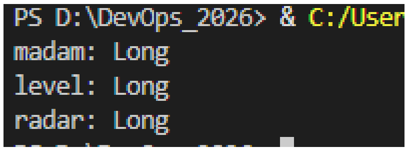
1. From the previous results, display only the names that are palindromes.
2. Using the generated list of words, calculate the length of each word.
3. Using the word lengths from the previous step, classify each word as:
   Short if length is less than 5
   Long if length is greater than or equal to 5
   Display each word with its classification.

Code:



```python
names = ["madam", "arjun", "level", "rahul", "radar"]
# Prepare names for palindrome checking (convert to lowercase)
processed_names = [name.lower() for name in names]
# Check each name for palindrome and collect palindromes
palindromes = [name for name in processed_names if name == name[::-1]]
# Classify each palindrome as Short or Long based on length
classification = {name: ("Short" if len(name) < 5 else "Long") for name in palindromes}
# Display each palindrome with its classification
for name, classif in classification.items():
    print(f"{name}: {classif}")
```

Output:



```
PS D:\DevOps_2026> & C:/User
madam: Long
level: Long
radar: Long
```

Explanation:

This code defines a list of student names and checks which of those names are palindromes (words that read the same forwards and backwards). The palindrome names are then filtered into a new list called palindromes.