Name: Achi Vaishnavi

Hall ticket:2303A51554

Batch:29

Assessment: 4.2

Task 1: Zero-shot: Prompt AI with only the instruction.

Prompt:

Generate a zero shot prompt to determine whether the given number is prime or not.

Code:

```python
# Generate a zero shot prompt to determine wheather the given number is prime or not.
def is_prime(n):
    if n < 2:
        return False
    for i in range(2, int(n**0.5) + 1):
        if n % i == 0:
            return False
    return True
number = 29
response = "Prime" if is_prime(number) else "Not Prime"
print("Response:", response)
# Output:
# Response: Prime
```

Output:

```
Response: Prime
```

Significance:

This task uses zero-shot prompting since no examples were given, only a direct instruction. The AI independently applied the concept of prime numbers and generated a logical solution. The function correctly eliminates numbers below 2 and checks for factors efficiently. This proves that zero-shot prompting is suitable for simple, standard programming problems. It reflects the AI's ability to reason without external guidance.

Task 2: One-shot: Provide one example:

Prompt:

Generate a function that calculates the sum of elements in a list.

EX: Input: [1 2 3 4] -> output:10

Code:

```
# Generate a function that calculates the sum of elements in a list.
# EX: Input: [1 2 3 4] -> output:10
def sum_of_elements(lst):
    return sum(lst)
# Example usage:
numbers = [1, 2, 3, 4]
result = sum_of_elements(numbers)
print("Sum of elements:", result)  # Output: Sum of elements: 10
```

Output:

Significance:

A single input-output example was provided to guide the AI's understanding. This helped the AI clearly identify that all elements in the list must be added. The produced function gives the correct total for the example input. One-shot prompting reduced confusion and improved correctness. It demonstrates how one example can effectively shape AI output.

Task 3: Few-shot: Give 2–3 examples

Prompt:

Generate function that extracts digits from an alphanumeric string

Ex:1 Input: "a1b2c3d4" -> output: "1234"

Ex:2 Input: "abc" -> output: ""

Ex:3 Input: "12345" -> output: "12345"

Code:

```
# Generate function that extracts digits from an alphanumeric string
# Ex:1 Input: "a1b2c3d4" -> output: "1234"
# Ex:2 Input: "abc" -> output: ""
# Ex:3 Input: "12345" -> output: "12345"
def extract_digits(s):
    return ''.join(filter(str.isdigit, s))
# Example usage:
input_str1 = "a1b2c3d4"
output1 = extract_digits(input_str1)
print("Extracted digits from '{}': {}".format(input_str1, output1))  #

input_str2 = "abc"
output2 = extract_digits(input_str2)
print("Extracted digits from '{}': {}".format(input_str2, output2))  #
input_str3 = "12345"
output3 = extract_digits(input_str3)
print("Extracted digits from '{}': {}".format(input_str3, output3))  #
# Output: Extracted digits from 'a1b2c3d4': 1234
# Output: Extracted digits from 'abc':
# Output: Extracted digits from '12345': 12345
```

Output:

```
Extracted digits from 'a1b2c3d4': 1234
Extracted digits from 'abc':
Extracted digits from '12345': 12345
```

Significance:

Several examples were used to show how digits should be separated from letters. These samples highlighted different scenarios such as mixed input, no digits, and only digits. The AI successfully generalized from these patterns and created an accurate function. Few-shot prompting improved precision and edge-case handling. This shows the benefit of providing multiple demonstrations.

Task 4: zero-shot vs few-shot prompting

Prompt:

Generate a function that counts the number of vowels in a string.

Code:

```python
#generate a function that counts the number of vowels in a string.
def count_vowels(s):
    vowels = 'aeiouAEIOU'
    return sum(1 for char in s if char in vowels)
# Example usage:
input_str = "Hello World"
vowel_count = count_vowels(input_str)
print("Number of vowels in '{}': {}".format(input_str, vowel_count))  # Output
```

Output:

```
Number of vowels in 'Hello World': 3
```

Prompt:

Generate a function that counts the number of vowels in a string.

Ex: Input: "Hello World" -> output: 3

Ex: Input: "Python" -> output: 1

Ex: Input: "AEIOU" -> output: 5

Code:

```python
#Generate a function that counts the number of vowels in a string.
# Ex: Input: "Hello World" -> output: 3
# Ex: Input: "Python" -> output: 1
# Ex: Input: "AEIOU" -> output: 5
def count_vowels(s):
    vowels = 'aeiouAEIOU'
    return sum(1 for char in s if char in vowels)
# Example usage:
input_str = "Hello World"
vowel_count = count_vowels(input_str)
print("Number of vowels in '{}': {}".format(input_str, vowel_count))  # Output: Number of vowels in 'Hello World': 3
# Output: Number of vowels in 'Hello World': 3

#Exanple usage:
input_str2 = "Python"
vowel_count2 = count_vowels(input_str2)
print("Number of vowels in '{}': {}".format(input_str2, vowel_count2))  # Output: Number of vowels in 'Python': 1

# Output: Number of vowels in 'Python': 1
input_str3 = "AEIOU"
vowel_count3 = count_vowels(input_str3)
print("Number of vowels in '{}': {}".format(input_str3, vowel_count3))
# Output: Number of vowels in 'AEIOU': 5
```

```python
#Generate a function that counts the number of vowels in a string.
# Ex: Input: "Hello World" -> output: 3
def count_vowels(s):
    vowels = 'aeiouAEIOU'
    return sum(1 for char in s if char in vowels)
# Example usage:
input_str = "Hello World"
vowel_count = count_vowels(input_str)
print("Number of vowels in '{}': {}".format(input_str, vowel_count))  # Output: Number of vowels in 'Hello World': 3
# Output: Number of vowels in 'Hello World': 3
```

Output:

```
Number of vowels in 'Hello World': 3
Number of vowels in 'Python': 1
Number of vowels in 'AEIOU': 5
```

Comparison Table:

```
# Comparsion table for both zero shot and few shot prompting for counting the number of vowels in a string. just a table not cpode

| Aspect                | Zero Shot Prompting                                    | Few Shot Prompting                                          |
|-----------------------|--------------------------------------------------------|-------------------------------------------------------------|
| Definition            | Directly asks the model to count vowels without examples | Provides examples of input-output pairs for counting vowels |
| Examples Provided     | None                                                   | Multiple examples demonstrating the task                    |
| Clarity of Task       | May be less clear, relies on model's understanding     |
| More explicit due to examples provided                 |
| Accuracy              | May vary, depends on model's prior knowledge           | Generally higher accuracy due
| Contextual Understanding | Relies on model's general knowledge                  | Enhanced by specific examples provided                      |
| Response Time         | Potentially faster as no examples need to be processed | May take slightly longer due to processing examples         |
| Use Case              | Quick tasks where examples are not feasible            | Tasks requiring higher accuracy and clarity                 |
| Flexibility           | More flexible, can handle a wider range of inputs      | More rigid, may perform best on inputs similar to examples  |
| Adaptability          | Adapts based on model's training data                  | Adapts based on provided examples                           |
```

Significance:

Zero-shot prompting depends on the AI's general language knowledge, while few-shot prompting clearly defines expectations through examples. The comparison indicates that examples make the task interpretation more precise. Few-shot prompting leads to more stable and predictable results. This task highlights how examples strengthen AI understanding. It proves that structured input improves output quality.

Task 5: Use few-shot prompting with 3 sample inputs

Prompt:

Generate a function that determines the minimum of three numbers without using the built-in min () function. with 3 or more examples.

Ex 1: Input: (10, 5, 8) -> output: 5

Ex 2: Input: (-1, -5, -3) -> output: -5

Ex 3: Input: (0, 0, 1) -> output: 0

Code:

```python
#Generate a function that determines the minimum of three numbers without using the built-in min() function. with 3 or more examples.
#Ex 1: Input: (10, 5, 8) -> output: 5
#Ex 2: Input: (-1, -5, -3) -> output: -5
#Ex 3: Input: (0, 0, 1) -> output: 0
def minimum_of_three(a, b, c):
    if a <= b and a <= c:
        return a
    elif b <= a and b <= c:
        return b
    else:
        return c
# Example usage:
num1 = 10
num2 = 5
num3 = 8
min_value = minimum_of_three(num1, num2, num3)
print("Minimum of {}, {}, and {} is: {}".format(num1, num2, num3, min_value))
# Output: Minimum of 10, 5, and 8 is: 5
# Example 2
num4 = -1
num5 = -5
num6 = -3
min_value2 = minimum_of_three(num4, num5, num6)
print("Minimum of {}, {}, and {} is: {}".format(num4, num5, num6, min_value2))  #
# Output: Minimum of -1, -5, and -3 is: -5
# Example 3
num7 = 0
num8 = 0
num9 = 1
min_value3 = minimum_of_three(num7, num8, num9)
print("Minimum of {}, {}, and {} is: {}".format(num7, num8, num9, min_value3))  #
# Output: Minimum of 0, 0, and 1 is: 0
```

Output:

```
Minimum of 10, 5, and 8 is: 5
Minimum of -1, -5, and -3 is: -5
Minimum of 0, 0, and 1 is: 0
```

Significance:

Three different sample cases were supplied to guide the AI. These included positive values, negative values, and repeated numbers, which clarified all major conditions. The generated logic correctly identifies the smallest number without using built-in functions. Few-shot prompting helped the AI follow a consistent decision-making pattern. This confirms the effectiveness of example-driven prompting.