Name: Shryitha Suddala

Hall ticket:2303A51637

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:

```
# 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:

```
PS C:\Users\shryi\Downloads\AI> & C:/Users/shryi/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/shryi/Downloads/AI/task 4.2.py"
Response: Prime
PS C:\Users\shryi\Downloads\AI>
```

Significance:

This task follows zero-shot prompting because only an instruction was given without any examples. The AI relied on its prior knowledge to generate the prime-checking logic. The function correctly handles numbers less than 2 and checks divisibility up to the square root of the number. This shows that zero-shot prompting is effective for basic and well-defined problems. It demonstrates the AI's ability to apply standard logic without 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:

```python
# 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:

```
PS C:\Users\shryi\Downloads\AI> & C:/Users/shryi/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/shryi/Downloads/AI/task 4.2.py"
Sum of elements: 10
```

Significance:

One example input and output were provided to guide the AI. This helped clearly define the expected task and reduced ambiguity. The generated function correctly calculates the sum of all elements in a list. One-shot prompting improved accuracy and ensured the output matched the given pattern. It shows how a single example can guide correct code generation.

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:

```python
# 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:

```
PS C:\Users\shryi\Downloads\AI> & C:/Users/shryi/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/shryi/Downloads/AI/task 4.2.py"
Extracted digits from 'a1b2c3d4': 1234
Extracted digits from 'abc':
Extracted digits from '12345': 12345
```

Significance:

Multiple examples were given, including mixed characters, only letters, and only digits. These examples helped the AI understand both the main requirement and edge cases. The generated function accurately extracts digits from any alphanumeric string. Few-shot prompting improved clarity, correctness, and robustness. This demonstrates how examples strengthen pattern recognition.

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:

```
PS C:\Users\shryi\Downloads\AI> & C:/Users/shryi/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/shryi/Downloads/AI/task 4.2.py"
Number of vowels in 'Hello World': 3
PS C:\Users\shryi\Downloads\AI>
```

Significance:

Prompt:

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

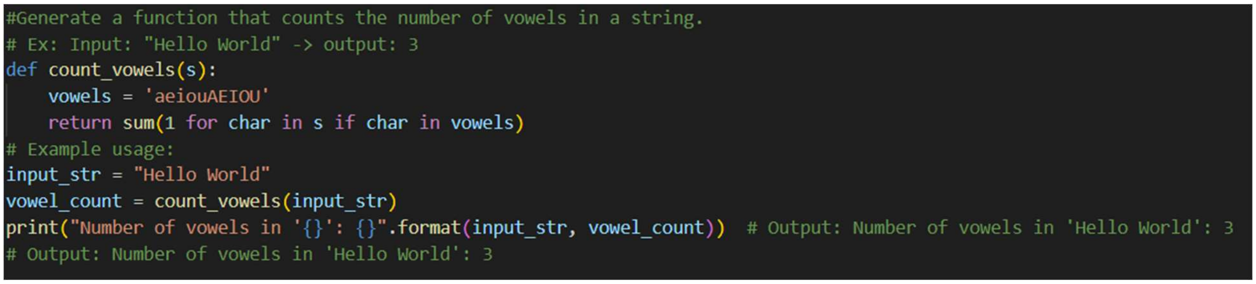
Ex: Input: "Hello World" -> output: 3

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

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

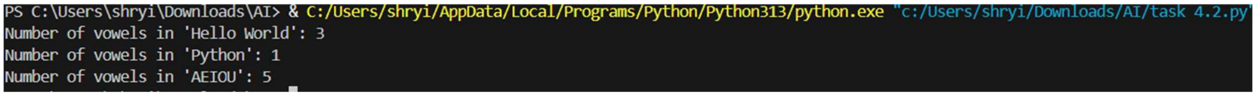
Code:

```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
```



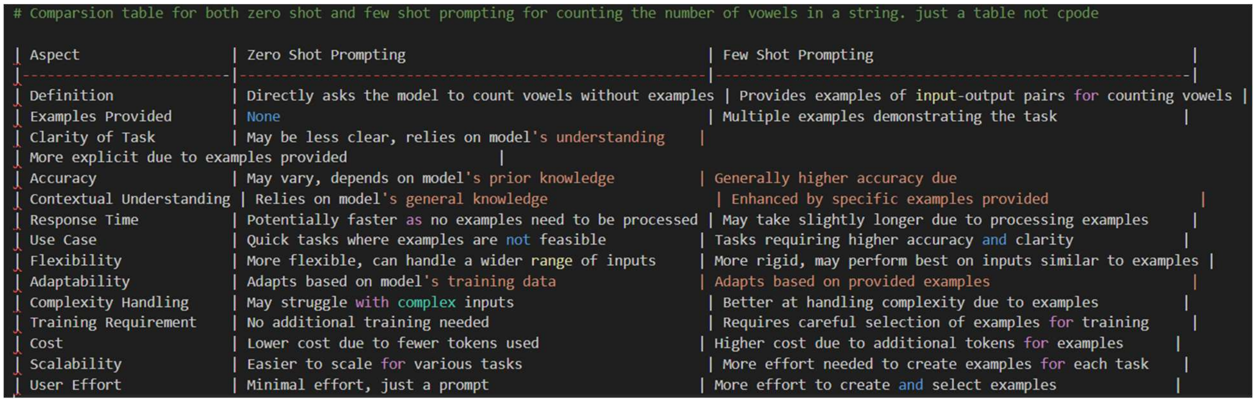
```python
#Generate a function that counts the number of vowels in a string.
# Ex: Input: "Hello World" -> output: 3
def count_vowels(s):
    vowels = 'aeiouAEIOU'
    return sum(1 for char in s if char in vowels)
# Example usage:
input_str = "Hello World"
vowel_count = count_vowels(input_str)
print("Number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



```
PS C:\Users\shryi\Downloads\AI> & C:/Users/shryi/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/shryi/Downloads/AI/task 4.2.py"
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                       |
| Complexity Handling    | May struggle with complex inputs                        | Better at handling complexity due to examples          |
| Training Requirement   | No additional training needed                           | Requires careful selection of examples for training     |
| Cost                   | Lower cost due to fewer tokens used                    | Higher cost due to additional tokens for examples      |
| Scalability            | Easier to scale for various tasks                       | More effort needed to create examples for each task    |
| User Effort            | Minimal effort, just a prompt                          | More effort to create and select examples              |
```

Significance:

In zero-shot prompting, the AI depends only on general knowledge of vowels. In few-shot prompting, examples clearly show how to count vowels in different inputs. The comparison shows that few-shot prompting provides better clarity and consistency. Examples reduce misunderstandings and improve output accuracy. This highlights the importance of examples in guiding AI behavior.

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:

```
PS C:\Users\shryi\Downloads\AI> & C:/Users/shryi/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/shryi/Downloads/AI/task 4.2.py"
Minimum of 10, 5, and 8 is: 5
Minimum of -1, -5, and -3 is: -5
Minimum of 0, 0, and 1 is: 0
PS C:\Users\shryi\Downloads\AI>
```

Significance:

Three examples were provided covering positive numbers, negative numbers, and equal values. These helped the AI learn the comparison pattern. The generated function correctly determines the minimum without using min(). Few-shot prompting ensured proper conditional logic and handling of different cases. This confirms that examples improve decision-based code generation.