# Sample Coding Questions

## String Occurence

### 1. Counting Occurrences of a Specific Character

Problem Statement: Write Python code to count the occurrences of a specific character in a given string.

```python
input_str = "hello"

char = "l"

count = input_str.count(char)

print(count)  # Output: 2
```

### 2. Finding the Most Common Character

Problem Statement: Write Python code to find the most common character in a string.

```python
input_str = "hello"


# Initialize variables to keep track of the most common character and its count

most_common_char = ""

max_count = 0


for char in input_str:

    count = input_str.count(char)

    if count > max_count:

        max_count = count

        most_common_char = char
```

```
print(f"The most common character is: {most_common_char}")
```

**3. Problem Statement: Character Occurrence Count**

You are given a word consisting of uppercase and lowercase letters. Your task is to count the occurrences of each character in the word.

Write a Python program that takes a word as input and prints the count of each character in the following format:

Character – Count

Assume that the word consists only of alphanumeric characters.

```
input_word = "Hello"

# Initialize a dictionary to store character counts
char_count = {}

# Iterate through each character in the input word
for char in input_word:
    if char in char_count:
        char_count[char] += 1
    else:
        char_count[char] = 1

# Print the character counts
for char, count in char_count.items():
    print(f"{char} - {count}")
```

**4. Counting Alphabetic Characters**

Problem Statement: Write Python code to count the number of alphabetic characters in a string.

```
input_str = "hello123"

count = 0


for char in input_str:

    if char.isalpha():

        count += 1


print(count)  # Output: 5
```

**5. Counting Digits, Letters, and Special Characters**

Problem Statement: Write Python code to count the number of digits, letters, and special characters in a string.

```
input_str = "hello123!@"

num_digits = 0

num_letters = 0


for char in input_str:

    if char.isdigit():

        num_digits += 1

    elif char.isalpha():

        num_letters += 1
```

```
num_special = len(input_str) - (num_digits + num_letters)

print(f"Digits: {num_digits}, Letters: {num_letters}, Special Characters:
{num_special}")
```

**6. Finding Unique Characters**

Problem Statement: Write Python code to find all unique characters in a string.

```
input_str = "hello"
unique_chars = []      #list approach

for char in input_str:
    if char not in unique_chars:
        unique_chars.append(char)

print(unique_chars)  # Output: ['h', 'e', 'l', 'o']
```

**7. Counting Vowels and Consonants**

Problem Statement: Write Python code to count the number of vowels and consonants in a string.

```
input_str = "hello"
vowels = "aeiouAEIOU"

num_vowels = 0
for char in input_str:
    if char in vowels:
        num_vowels += 1

num_consonants = len(input_str) - num_vowels

print(f"Vowels: {num_vowels}, Consonants: {num_consonants}")
```

**8. Problem Statement: Basic String Compression**

You are given a string containing only uppercase and lowercase letters. Your task is to perform basic string compression using the counts of repeated characters.

Write a Python program that compresses the input string according to the following rules:

For a sequence of the same character, replace it with the character followed by the count of its occurrences.

If the compressed string is not smaller than the original string, return the original string.

Implement the compression logic without using functions.

Sample Input - "aabcccccaaa"

Sample Output - "a2b1c5a3"

```
input_str = "aabcccccaaa"

compressed_str = ""
count = 1

for i in range(len(input_str)-1):
    if input_str[i] == input_str[i+1]:
        count += 1
    else:
        compressed_str += input_str[i] + str(count)
        count = 1

compressed_str += input_str[-1] + str(count)

print(compressed_str)
```

**9. Problem Statement – String Rotation**

**Given two strings, check if one is a rotation of the other. For example, "waterbottle" is a rotation of "erbottlewat."**

```
# Get input from the user
str1 = input("Enter the first string: ")
str2 = input("Enter the second string: ")

if len(str1) != len(str2):
    is_rotation = False
else:
    concatenated = str1 + str1
    if str2 in concatenated:
        is_rotation = True
    else:
        is_rotation = False

# if the lengths of str1 and str2 are not equal. If they are not equal,
```

# it means that one cannot be a rotation of the other, so is_rotation is set to False.

# If the lengths are equal, it proceeds with the rotation check:

# concatenated = str1 + str1 concatenates str1 with itself to create a

# new string called concatenated. This is done because a rotation of str1 will always be found

# within str1 concatenated with itself.

# if str2 in concatenated: checks if str2 is a substring of concatenated.

# If it is, it means that str2 is a rotation of str1, so is_rotation is set to True.

# therwise, it's set to False.

print(f"The second string is a rotation of the first string: {is_rotation}")

**10. Problem Statement - Longest Substring Without Repeating Characters**

**Find the length of the longest substring without repeating characters in a given string.**

# Given string

input_str = "abcabcbb"

# Initialize variables

max_length = 0

start = 0

end = 0

```
# Iterate through the characters of the string
for i in range(len(input_str)):
    for j in range(start, end):
        if input_str[i] == input_str[j]:
            start = j + 1
            break
    end += 1
    max_length = max(max_length, end - start)

# Print the result
print(max_length)  # Output: 3
```