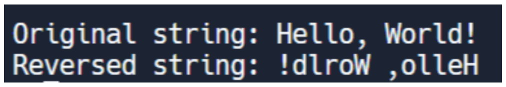
Reverse a String:

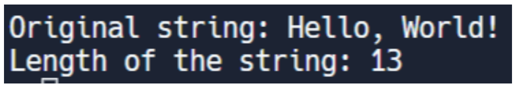
```python
def reverse_string(string):
    reversed_string = string[::-1]
    return reversed_string

input_string = "Hello, World!"
reversed_string = reverse_string(input_string)
print("Original string:", input_string)
print("Reversed string:", reversed_string)
```



```
Original string: Hello, World!
Reversed string: !dlroW ,olleH
```

Length without len():

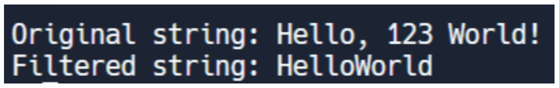
```python
def calculate_length(string):
    length = 0
    for _ in string:
        length += 1
    return length

input_string = "Hello, World!"
string_length = calculate_length(input_string)
print("Original string:", input_string)
print("Length of the string:", string_length)
```



```
Original string: Hello, World!
Length of the string: 13
```

Only Alphabets:

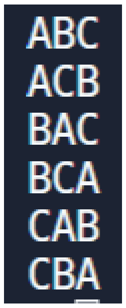
```python
def remove_non_alphabetic(string):
    result = ""
    for char in string:
        if char.isalpha():
            result += char
    return result

input_string = "Hello, 123 World!"
filtered_string = remove_non_alphabetic(input_string)
print("Original string:", input_string)
print("Filtered string:", filtered_string)
```



```
Original string: Hello, 123 World!
Filtered string: HelloWorld
```

Lexicographical Permutations:

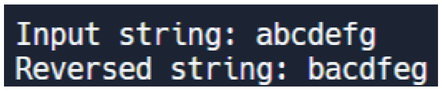
```python
ans = []
def permute(a, l, r):
    if l == r:
        ans.append(''.join(a))
    else:
        for i in range(l, r):
            a[l], a[i] = a[i], a[l]
            permute(a, l + 1, r)
            a[l], a[i] = a[i], a[l]

string = "ABC"
n = len(string)
a = list(string)
permute(a, 0, n)
for i in sorted(ans):
    print(i)
```



```
ABC
ACB
BAC
BCA
CAB
CBA
```

K-based Reversal:

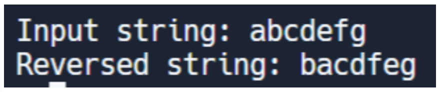
```python
def reverse_string(s, k):
    result = ""
    i = 0
    while i < len(s):
        if i % (2 * k) == 0:
            result += s[i:i+k][::-1]
        else:
            result += s[i:i+k]
        i += k

    return result

s = "abcdefg"
k = 2
reversed_string = reverse_string(s, k)
print("Input string:", s)
print("Reversed string:", reversed_string)
```



```
Input string: abcdefg
Reversed string: bacdfeg
```

Alternate Merging:

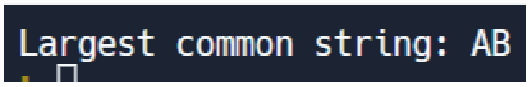
```python
def merge_strings(word1, word2):
    merged = ""
    i = 0
    while i < len(word1) or i < len(word2):
        if i < len(word1):
            merged += word1[i]
        if i < len(word2):
            merged += word2[i]
        i += 1
    return merged

word1 = "abc"
word2 = "pqr"
merged_string = merge_strings(word1, word2)
print("Merged string:", merged_string)
```



```
Input string: abcdefg
Reversed string: bacdfeg
```

GCD of String:

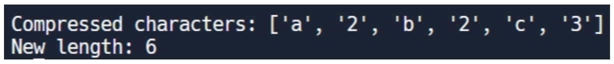
```python
def gcd(a, b):
    while b != 0:
        a, b = b, a % b
    return a

def largest_common_divisor(str1, str2):
    len1 = len(str1)
    len2 = len(str2)
    gcd_len = gcd(len1, len2)
    substring = str1[:gcd_len]

    if substring * (len1 // gcd_len) == str1 and
substring * (len2 // gcd_len) == str2:
        return substring
    else:
        return ""

str1 = "ABABAB"
str2 = "ABAB"
largest_string = largest_common_divisor(str1, str2)
print("Largest common string:", largest_string)
```



```
Largest common string: AB
```

String Compression:

```python
def compress(chars):
    current_char = chars[0]
    compressed_pos = 0
    count = 1

    for i in range(1, len(chars)):
        if chars[i] == current_char:
            count += 1
        else:
            chars[compressed_pos] = current_char
            compressed_pos += 1

            if count > 1:
                count_str = str(count)

chars[compressed_pos:compressed_pos+len(count_str)] =
count_str
                compressed_pos += len(count_str)
                count = 1

            current_char = chars[i]
    chars[compressed_pos] = current_char
    compressed_pos += 1

    if count > 1:
        count_str = str(count)

chars[compressed_pos:compressed_pos+len(count_str)] =
count_str
        compressed_pos += len(count_str)

    return compressed_pos

chars1 = ["a", "a", "b", "b", "c", "c", "c"]
result1 = compress(chars1)
print("Compressed characters:", chars1[:result1])
print("New length:", result1)
```



```
Compressed characters: ['a', '2', 'b', '2', 'c', '3']
New length: 6
```

Other String Programs to practice:

Reverse a String

Check whether a String is Palindrome or not

Find Duplicate characters in a string

Write a Code to check whether one string is a rotation of another

Write a Program to check whether a string is a valid shuffle of two strings or not

Count and Say problem

Write a program to find the longest Palindrome in a string.[Longest palindromic Substring]

Find Longest Recurring Subsequence in String

Print all Subsequences of a string.

Print all the permutations of the given string

Split the Binary string into two substring with equal 0's and 1's

Word Wrap Problem [VERY IMP].

EDIT Distance [Very Imp]

Find next greater number with same set of digits. [Very Very IMP]

Balanced Parenthesis problem.[Imp]

Word break Problem[Very Imp]

Rabin Karp Algo

KMP Algo

Convert a Sentence into its equivalent mobile numeric keypad sequence.

Minimum number of bracket reversals needed to make an expression balanced.

Count All Palindromic Subsequence in a given String.

Count of number of given string in 2D character array

Search a Word in a 2D Grid of characters.

Boyer Moore Algorithm for Pattern Searching.

Converting Roman Numerals to Decimal

Longest Common Prefix

Number of flips to make binary string alternate

Find the first repeated word in string.

Minimum number of swaps for bracket balancing.

Find the longest common subsequence between two strings.

Program to generate all possible valid IP addresses from given string.

Write a program tofind the smallest window that contains all characters of string itself.

Rearrange characters in a string such that no two adjacent are same

Minimum characters to be added at front to make string palindrome

Given a sequence of words, print all anagrams together

Find the smallest window in a string containing all characters of another string

Recursively remove all adjacent duplicates

String matching where one string contains wildcard characters

Function to find Number of customers who could not get a computer

Transform One String to Another using Minimum Number of Given Operation

Check if two given strings are isomorphic to each other

Recursively print all sentences that can be formed from list of word lists