String Data Structure

[**Complete Guide on Strings with Interview Questions**](https://www.geeksforgeeks.org/complete-guide-on-strings-with-interview-questions/)

[**Practice Problems on String**](https://practice.geeksforgeeks.org/explore/?category%5B%5D=Strings&page=1&category%5B%5D=Strings)

**What is String?**

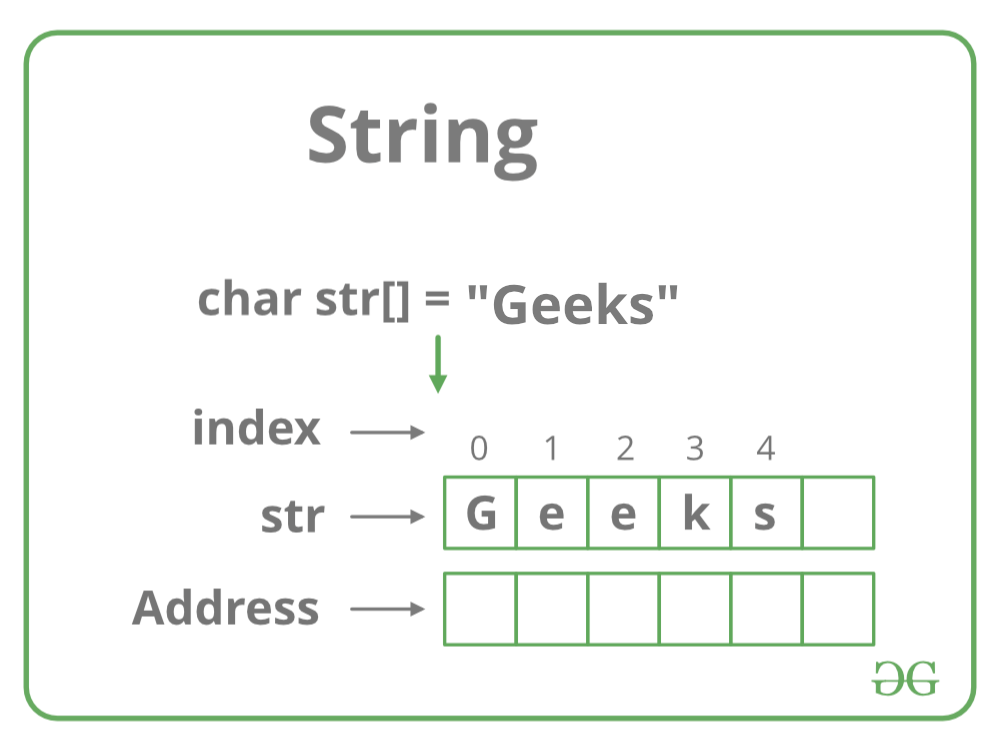
Strings are defined as an array of characters. The difference between a character array and a string is the string is terminated with a special character ‘\0’.

Below are some examples of strings:

*“geeks”, “for”, “geeks”, “GeeksforGeeks”, “Geeks for Geeks”, “123Geeks”, “@123 Geeks”*

**How String is represented in Memory?**

In C, a string can be referred to either using a character pointer or as a character array. When strings are declared as character arrays, they are stored like other types of arrays in C. For example, if str[] is an auto variable then the string is stored in the stack segment, if it’s a global or static variable then stored in the data segment, etc.

[](https://www.geeksforgeeks.org/complete-guide-on-strings-with-interview-questions/)

*What is String & How String is represented in Memory*

[**‘Recent Articles’ on Strings**](https://www.geeksforgeeks.org/category/data-structures/c-strings/)

**Topics :**

* [Basics](https://www.geeksforgeeks.org/string-data-structure/?ref=ghm#basics)
* [String in C & C++](https://www.geeksforgeeks.org/string-data-structure/?ref=ghm#C%20AND%20C++)
* [Strings in Java](https://www.geeksforgeeks.org/string-data-structure/?ref=ghm#Java)
* [String in Python](https://www.geeksforgeeks.org/string-data-structure/?ref=ghm#Python)
* [Arthimetic Operation in String](https://www.geeksforgeeks.org/string-data-structure/?ref=ghm#arthimetic)
* [Character Counting Based Problems](https://www.geeksforgeeks.org/string-data-structure/?ref=ghm#counting)
* [Subsequence & Substring](https://www.geeksforgeeks.org/string-data-structure/?ref=ghm#subsequence)
* [Reverse & Rotation](https://www.geeksforgeeks.org/string-data-structure/?ref=ghm#reverse%20&%20rotation)
* [Sorting & Searching](https://www.geeksforgeeks.org/string-data-structure/?ref=ghm#sorting%20&%20searching)
* [Case Sensitive String](https://www.geeksforgeeks.org/string-data-structure/?ref=ghm#case%20sensitive%20string)
* [Occurrence Based String](https://www.geeksforgeeks.org/string-data-structure/?ref=ghm#occurrence%20based%20string)
* [Spacing](https://www.geeksforgeeks.org/string-data-structure/?ref=ghm#spacing)
* [Anagram](https://www.geeksforgeeks.org/string-data-structure/?ref=ghm#anagram)
* [Palindrome](https://www.geeksforgeeks.org/string-data-structure/?ref=ghm#palindrome)
* [Binary String](https://www.geeksforgeeks.org/string-data-structure/?ref=ghm#binary)
* [Lexicographic pattern](https://www.geeksforgeeks.org/string-data-structure/?ref=ghm#lexicographic%20pattern)
* [Pattern Searching](https://www.geeksforgeeks.org/string-data-structure/?ref=ghm#Pattern)
* [Split String](https://www.geeksforgeeks.org/string-data-structure/?ref=ghm#split%20string)
* [Balance Parentheses & Bracket Evaluation](https://www.geeksforgeeks.org/string-data-structure/?ref=ghm#balence%20parentheses%20&%20bracket%20evaluation)
* [Conversion](https://www.geeksforgeeks.org/string-data-structure/?ref=ghm#conversion)
* [Misc](https://www.geeksforgeeks.org/string-data-structure/?ref=ghm#misc)
* [Quick Links](https://www.geeksforgeeks.org/string-data-structure/?ref=ghm#quick)

**Basics :**

1. [Function to copy string](https://www.geeksforgeeks.org/function-copy-string-iterative-recursive/)
2. [Pangram Checking](https://www.geeksforgeeks.org/pangram-checking/)
3. [Missing characters to make a string Pangram](https://www.geeksforgeeks.org/missing-characters-make-string-pangram/)
4. [Check if a string is Pangrammatic Lipogram](https://www.geeksforgeeks.org/check-string-pangrammatic-lipogram/)
5. [Removing punctuations from a given string](https://www.geeksforgeeks.org/removing-punctuations-given-string/)
6. [Rearrange characters in a string such that no two adjacent are same](https://www.geeksforgeeks.org/rearrange-characters-string-no-two-adjacent/)
7. [Program to check if input is an integer or a string](https://www.geeksforgeeks.org/program-check-input-integer-string/)
8. [Quick way to check if all the characters of a string are same](https://www.geeksforgeeks.org/quick-way-check-characters-string/)
9. [Program to find the initials of a name](https://www.geeksforgeeks.org/program-find-initials-name/)
10. [Check Whether a number is Duck Number or not](https://www.geeksforgeeks.org/check-whether-number-duck-number-not/)
11. [Round the given number to nearest multiple of 10](https://www.geeksforgeeks.org/round-the-given-number-to-nearest-multiple-of-10/)
12. [Change string to a new character set](https://www.geeksforgeeks.org/change-string-to-a-new-character-set/)
13. [Find one extra character in a string](https://www.geeksforgeeks.org/find-one-extra-character-string/)

**Strings in Python :**

1. String Methods in Python : [Set 1](https://www.geeksforgeeks.org/string-methods-python-set-1/), [Set 2](https://www.geeksforgeeks.org/python-string-methods-set-2-len-count-center-ljust-rjust-isalpha-isalnum-isspace-join/) , [Set 3](https://www.geeksforgeeks.org/python-string-methods-set-3-strip-lstrip-rstrip-min-max-maketrans-translate-relplace/)
2. [Dictionary and counter in Python to find winner of election](https://www.geeksforgeeks.org/dictionary-counter-python-find-winner-election/)
3. [Maximum length of consecutive 1’s in a binary string in Python using Map function](https://www.geeksforgeeks.org/maximum-length-consecutive-1s-binary-string-python-using-map-function/)
4. [Python code to print common characters of two Strings in alphabetical order](https://www.geeksforgeeks.org/python-code-print-common-characters-two-strings-alphabetical-order/)
5. [Using Counter() in Python to find minimum character removal to make two strings anagram](https://www.geeksforgeeks.org/using-counter-python-find-minimum-character-removal-make-two-strings-anagram/)
6. [Reverse string in Python](https://www.geeksforgeeks.org/reverse-string-python-5-different-ways/)
7. [Python groupby method to remove all consecutive duplicates](https://www.geeksforgeeks.org/python-groupby-method-remove-consecutive-duplicates/)
8. [Generate two output strings depending upon occurrence of character in input string in Python](https://www.geeksforgeeks.org/generate-two-output-strings-depending-upon-occurrence-character-input-string-python/)
9. [Python Dictionary to find mirror characters in a string](https://www.geeksforgeeks.org/python-dictionary-find-mirror-characters-string/)
10. [Python | Convert a list of characters into a string](https://www.geeksforgeeks.org/python-convert-list-characters-string/)
11. [Map function and Lambda expression in Python to replace characters](https://www.geeksforgeeks.org/map-function-lambda-expression-python-replace-characters/)
12. [Zip function in Python to change to a new character set](https://www.geeksforgeeks.org/zip-function-python-change-new-character-set/)
13. [SequenceMatcher in Python for Longest Common Substring](https://www.geeksforgeeks.org/sequencematcher-in-python-for-longest-common-substring/)
14. [Python | Print the initials of a name with last name in full](https://www.geeksforgeeks.org/python-print-initials-name-last-name-full/)
15. [Python counter and dictionary intersection example (Make a string using deletion and rearrangement)](https://www.geeksforgeeks.org/python-counter-dictionary-intersection-example-make-string-using-deletion-rearrangement/)
16. [Python program to count number of vowels using sets in given string](https://www.geeksforgeeks.org/python-program-count-number-vowels-using-sets-given-string/)
17. [Python set to check if string is panagram](https://www.geeksforgeeks.org/python-set-check-string-panagram/)
18. [Python | Check if a Substring is Present in a Given String](https://www.geeksforgeeks.org/python-check-substring-present-given-string/)
19. [Python sorted() to check if two strings are anagram or not](https://www.geeksforgeeks.org/python-sorted-check-two-strings-anagram-not/)
20. [Python | Remove leading zeros from an IP address](https://www.geeksforgeeks.org/python-remove-leading-zeros-ip-address/)
21. [Python | Count all prefixes in given string with greatest frequency](https://www.geeksforgeeks.org/python-count-prefixes-given-string-greatest-frequency/)
22. [Check if both halves of the string have same set of characters in Python](https://www.geeksforgeeks.org/check-halves-string-set-characters-python/)
23. [Concatenated string with uncommon characters in Python](https://www.geeksforgeeks.org/concatenated-string-uncommon-characters-python/)
24. [Second most repeated word in a sequence in Python](https://www.geeksforgeeks.org/second-repeated-word-sequence-python/)
25. [Regex in Python to put spaces between words starting with capital letters](https://www.geeksforgeeks.org/regex-in-python-to-put-spaces-between-words-starting-with-capital-letters/)
26. [Python code to move spaces to front of string in single traversal](https://www.geeksforgeeks.org/python-code-move-spaces-front-string-single-traversal/)
27. [String slicing in Python to rotate a string](https://www.geeksforgeeks.org/string-slicing-python-rotate-string/)
28. [String slicing in Python to check if a string can become empty by recursive deletion](https://www.geeksforgeeks.org/string-slicing-python-check-string-can-become-empty-recursive-deletion/)
29. [Reverse words in a given String in Python](https://www.geeksforgeeks.org/reverse-words-given-string-python/)
30. [Run Length Encoding in Python](https://www.geeksforgeeks.org/run-length-encoding-python/)
31. [Anagram checking in Python using collections.Counter()](https://www.geeksforgeeks.org/anagram-checking-python-collections-counter/)
32. [Remove all duplicates from a given string in Python](https://www.geeksforgeeks.org/remove-duplicates-given-string-python/)
33. [Remove all consecutive duplicates from the string](https://www.geeksforgeeks.org/remove-consecutive-duplicates-string/)
34. [Python program to check if a string is palindrome or not](https://www.geeksforgeeks.org/python-program-check-string-palindrome-not/)

**Python String Methods | Set 1 (find, rfind, startwith, endwith, islower, isupper, lower, upper, swapcase & title)**

Some of the string basics have been covered in the below articles

[Strings Part-1](https://www.geeksforgeeks.org/interesting-facts-about-strings-in-python-set-1/)

[Strings Part-2](https://www.geeksforgeeks.org/interesting-facts-about-strings-in-python-set-2/)

The important string methods will be discussed in this article

**1. find(“string”, beg, end)**:- This function is used to find the position of the substring within a string.It takes 3 arguments, **substring , starting index( by default 0) and ending index( by default string length)**.

* It returns “-1 ” if string is not found in given range.
* It returns first occurrence of string if found.

**2. rfind(“string”, beg, end)**:- This function has the similar working as find(), but it returns the position of the **last occurrence** of string.

* Python

# Python code to demonstrate working of

# find() and rfind()

str **=** "geeksforgeeks is for geeks"

str2 **=** "geeks"

# using find() to find first occurrence of str2

# returns 8

print ("The first occurrence of str2 is at : ", end**=**"")

print (str.find( str2, 4) )

# using rfind() to find last occurrence of str2

# returns 21

print ("The last occurrence of str2 is at : ", end**=**"")

**print** ( str.rfind( str2, 4) )

Output:

The first occurrence of str2 is at : 8  
The last occurrence of str2 is at : 21

**Time complexity**: O(n)

**Auxiliary Space** : O(1)

**3. startswith(“string”, beg, end)** :- The purpose of this function is to return true if the function **begins with mentioned string(prefix)** else return false.

**4. endswith(“string”, beg, end)** :- The purpose of this function is to return true if the **function ends with mentioned string(suffix)** else return false.

# Python code to demonstrate working of

# startswith() and endswith()

str **=** "geeks"

str1 **=** "geeksforgeeksportal"

# using startswith() to find if str

# starts with str1

**if** str1.startswith(str):

        print ("str1 begins with : " **+** str)

**else** : **print** ("str1 does not begin with : "**+** str)

# using endswith() to find

# if str ends with str1

**if** str1.endswith(str):

**print** ("str1 ends with : " **+** str)

**else** :

**print** ("str1 does not end with : " **+** str)

Output:

str1 begins with : geeks  
str1 does not end with : geeks

**Time complexity**: O(n)

**Auxiliary Space** : O(1)

**5. islower(“string”)** :- This function returns true if all the letters in the string are **lower cased,** otherwise false.

**6. isupper(“string”)** :- This function returns true if all the letters in the string are **upper cased**, otherwise false.

# Python code to demonstrate working of

# isupper() and islower()

str **=** "GeeksforGeeks"

str1 **=** "geeks"

# checking if all characters in str are upper cased

**if** str.isupper() :

    print ("All characters in str are upper cased")

**else** :

**print** ("All characters in str are not upper cased")

# checking if all characters in str1 are lower cased

**if** str1.islower() :

**print** ("All characters in str1 are lower cased")

**else** :

    print ("All characters in str1 are not lower cased")

Output:

All characters in str are not upper cased  
All characters in str1 are lower cased

**Time complexity**: O(n)

**Auxiliary Space** : O(1)

**7. lower()** :- This function returns the new string with all the letters **converted into its lower case**.

**8. upper()** :- This function returns the new string with all the letters**converted into its upper case**.

**9. swapcase()** :- This function is used to swap the cases of string i.e upper case is converted to lower case and vice versa.

**10. title()** :- This function converts the string to its **title case** i.e the first letter of every word of string is upper cased and else all are lower cased.

* Python3

# Python code to demonstrate working of

# upper(), lower(), swapcase() and title()

str **=** "GeeksForGeeks is fOr GeeKs"

# Converting string into its lower case

str1 **=** str.lower();

**print** (" The lower case converted string is : " **+** str1)

# Converting string into its upper case

str2 **=** str.upper();

print (" The upper case converted string is : " **+** str2)

# Converting string into its swapped case

str3 **=** str.swapcase();

print (" The swap case converted string is : " **+** str3)

# Converting string into its title case

str4 **=** str.title();

print (" The title case converted string is : " **+** str4)

Output:

The lower case converted string is : geeksforgeeks is for geeks  
 The upper case converted string is : GEEKSFORGEEKS IS FOR GEEKS  
 The swap case converted string is : gEEKSfORgEEKS IS FoR gEEkS  
 The title case converted string is : Geeksforgeeks Is For Geeks

**Time complexity**: O(n)

**Auxiliary Space** : O(1)

**Python String Methods | Set 2 (len, count, center, ljust, rjust, isalpha, isalnum, isspace & join)**

Some of the string methods are covered in the set 3 below

[String Methods Part- 1](https://www.geeksforgeeks.org/python-string-methods-set-1-find-rfind-startwith-endwith-islower-isupper-lower-upper-swapcase-title/)

More methods are discussed in this article

**1. len()** :- This function returns the **length** of the string.

**2. count(“string”, beg, end)** :- This function **counts**the occurrence of mentioned **substring** in whole string. This function takes 3 arguments, s**ubstring, beginning position( by default 0) and end position(by default string length).**

# Python code to demonstrate working of

# len() and count()

str **=** "geeksforgeeks is for geeks"

# Printing length of string using len()

**print** (" The length of string is : ", len(str));

# Printing occurrence of "geeks" in string

# Prints 2 as it only checks till 15th element

print (" Number of appearance of ""geeks"" is : ",end**=**"")

print (str.count("geeks",0,15))

Output:

The length of string is : 26  
 Number of appearance of geeks is : 2

**3. center()** :- This function is used to **surround the string with a character** repeated both sides of string multiple times. By default the character is a space. Takes 2 arguments,**length of string and the character.**

**4. ljust()** :- This function returns the **original string shifted to left** that has a **character at its right**. It left adjusts the string. By default the character is space. It also takes two arguments,**length of string and the character.**

**5. rjust()** :- This function returns the **original string shifted to right** that has a **character at its left**. It right adjusts the string. By default the character is space. It also takes two arguments, **length of string and the character.**

# Python code to demonstrate working of

# center(), ljust() and rjust()

str **=** "geeksforgeeks"

# Printing the string after centering with '-'

**print** ("The string after centering with '-' is : ",end**=**"")

**print** ( str.center(20,'-'))

# Printing the string after ljust()

**print** ("The string after ljust is : ",end**=**"")

print ( str.ljust(20,'-'))

# Printing the string after rjust()

print ("The string after rjust is : ",end**=**"")

print ( str.rjust(20,'-'))

Output:

The string after centering with '-' is : ---geeksforgeeks----  
The string after ljust is : geeksforgeeks-------  
The string after rjust is : -------geeksforgeeks

**6. isalpha()** :- This function returns true when all the characters in the string are **alphabets**else returns false.

**7. isalnum()** :- This function returns true when all the characters in the string are **combination of numbers and/or alphabets** else returns false.

**8. isspace()** :- This function returns true when all the characters in the string are **spaces** else returns false.

# Python code to demonstrate working of

# isalpha(), isalnum(), isspace()

str **=** "geeksforgeeks"

str1 **=** "123"

# Checking if str has all alphabets

**if** (str.isalpha()):

       print ("All characters are alphabets in str")

**else** : print ("All characters are not alphabets in str")

# Checking if str1 has all numbers

**if** (str1.isalnum()):

       print ("All characters are numbers in str1")

**else** : **print** ("All characters are not numbers in str1")

# Checking if str1 has all spaces

**if** (str1.isspace()):

       print ("All characters are spaces in str1")

**else** : **print** ("All characters are not spaces in str1")

Output:

All characters are alphabets in str  
All characters are numbers in str1  
All characters are not spaces in str1

**9. join()** :- This function is used to **join a sequence** of strings mentioned in its arguments with the string.

# Python code to demonstrate working of

# join()

str **=** "\_"

str1 **=** ( "geeks", "for", "geeks" )

# using join() to join sequence str1 with str

print ("The string after joining is : ", end**=**"")

print ( str.join(str1))

Output:

The string after joining is : geeks\_for\_geeks

**Python String Methods | Set 3 (strip, lstrip, rstrip, min, max, maketrans, translate, replace & expandtabs())**

Some of the string methods are covered in the below sets.

[String Methods Part- 1](https://www.geeksforgeeks.org/python-string-methods-set-1-find-rfind-startwith-endwith-islower-isupper-lower-upper-swapcase-title/)

[String Methods Part- 2](https://www.geeksforgeeks.org/python-string-methods-set-2-len-count-center-ljust-rjust-isalpha-isalnum-isspace-join/)

More methods are discussed in this article

**1. strip()**:- This method is used to**delete all the leading and trailing**characters mentioned in its argument.

**2. lstrip()**:- This method is used to **delete all the leading** characters mentioned in its argument.

**3. rstrip()**:- This method is used to **delete all the trailing** characters mentioned in its argument.

**Python**

# Python code to demonstrate working of

# strip(), lstrip() and rstrip()

str **=** "---geeksforgeeks---"

# using strip() to delete all '-'

print ( " String after stripping all '-' is : ", end**=**"")

**print** ( str.strip('-') )

# using lstrip() to delete all trailing '-'

**print** ( " String after stripping all leading '-' is : ", end**=**"")

**print** ( str.lstrip('-') )

# using rstrip() to delete all leading '-'

**print** ( " String after stripping all trailing '-' is : ", end**=**"")

**print** ( str.rstrip('-') )

Output:

String after stripping all '-' is : geeksforgeeks  
 String after stripping all leading '-' is : geeksforgeeks---  
 String after stripping all trailing '-' is : ---geeksforgeeks

**4. min(“string”)**:- This function returns the **minimum value alphabet** from the string.

**5. max(“string”)** :- This function returns the **maximum value alphabet** from string.

**Python**

# Python code to demonstrate working of

# min() and max()

str **=** "geeksforgeeks"

# using min() to print the smallest character

# prints 'e'

print ("The minimum value character is : " **+** min(str))

# using max() to print the largest character

# prints 's'

print ("The maximum value character is : " **+** max(str))

Output:

The minimum value character is : e  
The maximum value character is : s

**6. maketrans()**:- It is used to **map the contents of string 1 with string 2** with respective indices to be translated later using translate().

**7. translate()**:- This is used to **swap the string elements mapped** with the help of maketrans().

**Python**

# Python code to demonstrate working of

# maketrans() and translate()

**from** string **import** maketrans # for maketrans()

str **=** "geeksforgeeks"

str1 **=** "gfo"

str2 **=** "abc"

# using maketrans() to map elements of str2 with str1

mapped **=** maketrans( str1, str2 )

# using translate() to translate using the mapping

print "The string after translation using mapped elements is : "

print  str.translate(mapped)

Output:

The string after translation using mapped elements is :   
aeeksbcraeeks

In the above code, ‘g’ is replaced by a, ‘f’ is replaced by b, and ‘o’ is replaced by ‘c’ in the string using the translate function.

**8.replace()**:- This function is used to **replace the substring with a new substring** in the string. This function has 3 arguments. **The string to replace, new string which would replace and max value denoting the limit to replace action ( by default unlimited ).**

**Python**

# Python code to demonstrate working of

# replace()

str **=** "nerdsfornerds is for nerds"

str1 **=** "nerds"

str2 **=** "geeks"

# using replace() to replace str2 with str1 in str

# only changes 2 occurrences

print ("The string after replacing strings is : ", end**=**"")

print (str.replace( str1, str2, 2))

Output:

The string after replacing strings is : geeksforgeeks is for nerds

This method is contributed by [Chinmoy Lenka](https://auth.geeksforgeeks.org/user/Chinmoy%20Lenka)

**9. expandtabs()**:- It is used to **replace all tab characters(“\t”) with whitespace** or simply spaces using the given tab size, which is optional to supply.

Syntax: string.tabsize(tabsize)

Parameters: Specifying the number of characters to be replaced for one tab character. By default, the function takes tab size as 8.

Return Value: A string in which all the tab characters are replaced with spaces.

**Python3**

# Python code to illustrate expandtabs()

string **=** 'GEEKS\tFOR\tGEEKS'

# No parameters, by default size is 8

print (string.expandtabs())

# tab size taken as 2

**print**(string.expandtabs(2))

# tab size taken as 5

print(string.expandtabs(5))

Output:

GEEKS FOR GEEKS  
GEEKS FOR GEEKS  
GEEKS FOR GEEKS

**Dictionary and counter in Python to find winner of election**

Given an array of names of candidates in an election. A candidate name in the array represents a vote cast to the candidate. Print the name of candidates received Max vote. If there is tie, print a lexicographically smaller name.

**Examples:**

Input : votes[] = {"john", "johnny", "jackie",   
 "johnny", "john", "jackie",   
 "jamie", "jamie", "john",  
 "johnny", "jamie", "johnny",   
 "john"};  
Output : John  
We have four Candidates with name as 'John',   
'Johnny', 'jamie', 'jackie'. The candidates  
John and Johny get maximum votes. Since John  
is alphabetically smaller, we print it.

[Recommended: Please try your approach on ***{IDE}*** first, before moving on to the solution.](https://ide.geeksforgeeks.org/)

We have existing solution for this problem please refer [Find winner of an election where votes are represented as candidate names](https://www.geeksforgeeks.org/find-winner-election-votes-represented-candidate-names/) link. We can solve this problem quickly in python using [Dictionary data structure](https://www.youtube.com/watch?v=z7z_e5-l2yE&t=31s).

**Method 1:**

*Approach is very simple,*

1. *Convert given list of votes into dictionary using Counter(iterator) method. We will have a dictionary having candidate names as****Key****and their frequency ( counts ) as****Value****.*
2. *Since more than 1 candidate may get same number of votes and in this situation we need to print lexicographically smaller name, so now we will create another dictionary by traversing previously created dictionary, counts of votes will be****Key****and candidate names will be****Value****.*
3. *Now find value of maximum vote casted for a candidate and get list of candidates mapped on that count value.*
4. *Sort list of candidates having same number of maximum votes and print first element of sorted list in order to print lexicographically smaller name.*

**Implementation:**

* Python3

# Function to find winner of an election where votes

# are represented as candidate names

**from** collections **import** Counter

**def** winner(input):

    # convert list of candidates into dictionary

    # output will be likes candidates = {'A':2, 'B':4}

    votes **=** Counter(input)

    # create another dictionary and it's key will

    # be count of votes values will be name of

    # candidates

    dict **=** {}

**for** value **in** votes.values():

        # initialize empty list to each key to

        # insert candidate names having same

        # number of votes

        dict[value] **=** []

**for** (key,value) **in** votes.items():

        dict[value].append(key)

    # sort keys in descending order to get maximum

    # value of votes

    maxVote **=** sorted(dict.keys(),reverse**=**True)[0]

    # check if more than 1 candidates have same

    # number of votes. If yes, then sort the list

    # first and print first element

**if** len(dict[maxVote])>1:

**print** (sorted(dict[maxVote])[0])

**else**:

**print** (dict[maxVote][0])

# Driver program

**if** \_\_name\_\_ **==** "\_\_main\_\_":

    input **=**['john','johnny','jackie','johnny',

            'john','jackie','jamie','jamie',

            'john','johnny','jamie','johnny',

            'john']

    winner(input)

**Output**

john

**Time complexity : O(nlogn)**

**Auxiliary Space : O(n)**

**Maximum length of consecutive 1’s in a binary string in Python using Map function**

We are given a binary string containing 1’s and 0’s. Find the maximum length of consecutive 1’s in it.

**Examples:**

Input : str = '11000**1111**01010111'  
Output : 4

[Recommended: Please try your approach on ***{IDE}*** first, before moving on to the solution.](https://ide.geeksforgeeks.org/)

We have an existing solution for this problem please refer to [Maximum consecutive one’s (or zeros) in a binary array](https://www.geeksforgeeks.org/maximum-consecutive-ones-or-zeros-in-a-binary-array/) link. We can solve this problem within single line of code in Python. The approach is very simple,

1. Separate all sub-strings of consecutive 1’s separated by zeros using [split()](https://www.geeksforgeeks.org/how-to-split-a-string-in-cc-python-and-java/) method of string.
2. Print maximum length of split sub-strings of 1’s.

**Implementation:**

**Python**

# Function to find Maximum length of consecutive 1's in a binary string

**def** maxConsecutive1(input):

     # input.split('0') --> splits all sub-strings of consecutive 1's

     # separated by 0's, output will be like ['11','1111','1','1','111']

     # map(len,input.split('0'))  --> map function maps len function on each

     # sub-string of consecutive 1's

     # max() returns maximum element from a list

**print** max(map(len,input.split('0')))

# Driver program

**if** \_\_name\_\_ **==** "\_\_main\_\_":

    input **=** '11000111101010111'

    maxConsecutive1(input)

**Output**

4

**Time complexity : O(n)**

**Auxiliary Space : O(1)**

**Python code to print common characters of two Strings in alphabetical order**

* Difficulty Level : [Easy](https://www.geeksforgeeks.org/easy/)

Given two strings, print all the common characters in lexicographical order. If there are no common letters, print -1. All letters are lower case.

**Examples:**

Input :   
string1 : geeks  
string2 : forgeeks  
Output : eegks  
Explanation: The letters that are common between   
the two strings are e(2 times), g(1 time), k(1 time) and   
s(1 time).  
Hence the lexicographical output is "eegks"

Input :   
string1 : hhhhhello  
string2 : gfghhmh  
Output : hhh

[Recommended: Please try your approach on ***{IDE}*** first, before moving on to the solution.](https://ide.geeksforgeeks.org/)

This problem has existing solution please refer [Print common characters of two Strings in alphabetical order](https://www.geeksforgeeks.org/print-common-characters-two-strings-alphabetical-order-2/) link. We will solve this problem in python using [intersection](https://www.geeksforgeeks.org/sets-in-python/) property and [collections.Counter()](https://www.geeksforgeeks.org/counters-in-python-set-1/) module. Approach is simple,

1. *Convert both strings into dictionary data type using****Counter(str)****method, which contains characters of string as key and their frequencies as value.*
2. *Now find common elements between two strings using****intersection ( a&b )****property.*
3. *Resultant will also be an counter dictionary having common elements as keys and their common frequencies as value.*
4. *Use****elements()****method of counter dictionary to expand list of keys by their frequency number of times.*
5. *Sort the list and concatenate each character of output list without space to print resultant string.*

**Implementation:**

**Python3**

# Function to print common characters of two Strings

# in alphabetical order

**from** collections **import** Counter

**def** common(str1,str2):

    # convert both strings into counter dictionary

    dict1 **=** Counter(str1)

    dict2 **=** Counter(str2)

    # take intersection of these dictionaries

    commonDict **=** dict1 & dict2

**if** len(commonDict) **==** 0:

        print (**-**1)

**return**

    # get a list of common elements

    commonChars **=** list(commonDict.elements())

    # sort list in ascending order to print resultant

    # string on alphabetical order

    commonChars **=** sorted(commonChars)

    # join characters without space to produce

    # resultant string

    print (''.join(commonChars))

# Driver program

**if** \_\_name\_\_ **==** "\_\_main\_\_":

    str1 **=** 'geeks'

    str2 **=** 'forgeeks'

    common(str1, str2)

**Output:**

eegks

**Time complexity : O(n)**

**Auxiliary Space : O(n)**

**Using Counter() in Python to find minimum character removal to make two strings anagram**

* Difficulty Level : [Easy](https://www.geeksforgeeks.org/easy/)

Given two strings in lowercase, the task is to make them [Anagram](https://www.geeksforgeeks.org/check-whether-two-strings-are-anagram-of-each-other/). The only allowed operation is to remove a character from any string. Find minimum number of characters to be deleted to make both the strings anagram? If two strings contains same data set in any order then strings are called **Anagrams**.

**Examples**:

Input : str1 = "bcadeh" str2 = "hea"  
Output: 3  
We need to remove b, c and d from str1.

Input : str1 = "cddgk" str2 = "gcd"  
Output: 2

Input : str1 = "bca" str2 = "acb"  
Output: 0

[Recommended: Please try your approach on ***{IDE}*** first, before moving on to the solution.](https://ide.geeksforgeeks.org/)

This problem has existing solution please refer [Remove minimum number of characters so that two strings become anagram](https://www.geeksforgeeks.org/remove-minimum-number-characters-two-strings-become-anagram/) link. We will solve this problem in python quickly using [Counter()](https://www.geeksforgeeks.org/counters-in-python-set-1/) and [Dictionary Data Structure](https://www.youtube.com/watch?v=z7z_e5-l2yE&t=28s) and **intersection** property of [Set](https://www.geeksforgeeks.org/using-counter-python-find-minimum-character-removal-make-two-strings-anagram/) data structure. Approach is very simple,

1. *Convert each string into a dictionary data structure using****Counter(iterable)****method.*
2. *Count number of keys in both dictionaries ( count1, count2) and count number of keys common in both dictionaries.*
3. *If no common keys found that means we need to remove****count1 + count2****characters from both the strings.*
4. *Else****(max(count1, count2) – countCommon)****will be the number of characters to be removed*

**Implementation:**

**Python3**

# Function remove minimum number of characters so that

# two strings become anagram

**from** collections **import** Counter

**def** removeChars(str1, str2):

    # make dictionaries from both strings

    dict1 **=** Counter(str1)

    dict2 **=** Counter(str2)

    # extract keys from dict1 and dict2

    keys1 **=** dict1.keys()

    keys2 **=** dict2.keys()

    # count number of keys in both lists of keys

    count1 **=** len(keys1)

    count2 **=** len(keys2)

    # convert list of keys in set to find common keys

    set1 **=** set(keys1)

    commonKeys **=** len(set1.intersection(keys2))

**if** (commonKeys **==** 0):

**return** count1 **+** count2

**else**:

**return** (max(count1, count2)**-**commonKeys)

# Driver program

**if** \_\_name\_\_ **==** "\_\_main\_\_":

    str1 **=**'bcadeh'

    str2 **=**'hea'

    print (removeChars(str1, str2))

**Output**

3

***Alternate Solution :***

1. *Convert each string into a dictionary data structure using****Counter(iterable)****method.*
2. *Find the common elements from both dictionary*
3. *Add up the values from common dictionary in order to get the total number of common elements.*

**Implementation:**

**Python3**

# Function remove minimum number of characters so that

# two strings become anagram

**from** collections **import** Counter

**def** removeChars(a, b):

    # make dictionaries from both strings

    c1 **=** Counter(a)

    c2 **=** Counter(b)

    # finding the common elements from both dictionary

    common **=** c1&c2

    value **=** 0

    # adding up the key from common dictionary in order

    # to get the total number of common elements

**for** key **in** common:

        value **=** value **+** common[key]

    # returning the number of elements to be

    # removed to form an anagram

**return** (len(a)**-**2**\***value**+** len(b))

# Driver program

**if** \_\_name\_\_ **==** "\_\_main\_\_":

    str1 **=**'bcadeh'

    str2 **=**'hea'

    print (removeChars(str1, str2))

**Output**

3

**Time Complexity:**O(n) where n is length of string

**Reverse string in Python (6 different ways)**

Python string library doesn’t support the in-built “[**reverse()**](https://www.geeksforgeeks.org/python-list-reverse/)” as done by other python containers like list, hence knowing other methods to reverse string can prove to be useful. This article discusses several ways to achieve it in [Python](https://www.geeksforgeeks.org/python-programming-language/).

**Example:**

**Input:**  Geeksforgeeks  
**Output:** skeegrofskeeG

**Reverse a string in Python using a loop**

In this example, we call a function to reverse a string, which iterates to every element and intelligently **joins each character in the beginning** so as to obtain the reversed string.

**Time complexity: O(n)**

**Auxiliary Space: O(1)**

**Implementation:**

1. Python3

**def** reverse(s):

    str **=** ""

**for** i **in** s:

        str **=** i **+** str

**return** str

s **=** "Geeksforgeeks"

print("The original string is : ", end**=**"")

**print**(s)

print("The reversed string(using loops) is : ", end**=**"")

print(reverse(s))

**Output**

The original string is : Geeksforgeeks  
The reversed string(using loops) is : skeegrofskeeG

**Reverse a string in Python using recursion**

The string is passed as an argument to a [recursive function](https://www.geeksforgeeks.org/recursive-functions/) to reverse the string. In the function, the base condition is that if the length of the string is equal to 0, the string is returned. If not equal to 0, the reverse function is recursively called to slice the part of the string except the first character and concatenate the first character to the end of the sliced string. ‘

**Time complexity**: O(n)

**Auxiliary Space**: O(n)

**Implementation:**

1. Python3

**def** reverse(s):

**if** len(s) **==** 0:

**return** s

**else**:

**return** reverse(s[1:]) **+** s[0]

s **=** "Geeksforgeeks"

**print**("The original string is : ", end**=**"")

print(s)

print("The reversed string(using recursion) is : ", end**=**"")

print(reverse(s))

**Output**

The original string is : Geeksforgeeks  
The reversed string(using recursion) is : skeegrofskeeG

**Reverse string in Python using stack**

An empty [stack](https://www.geeksforgeeks.org/stack-data-structure/) is created. One by one character of the string is pushed to the stack. One by one all characters from the stack are popped and put back to a string.

**Time complexity: O(n)**

**Auxiliary Space: O(n)**

**Implementation:**

1. Python3

# Function to create an empty stack. It

# initializes size of stack as 0

**def** createStack():

    stack **=** []

**return** stack

# Function to determine the size of the stack

**def** size(stack):

**return** len(stack)

# Stack is empty if the size is 0

**def** isEmpty(stack):

**if** size(stack) **==** 0:

**return** true

# Function to add an item to stack . It

# increases size by 1

**def** push(stack, item):

    stack.append(item)

# Function to remove an item from stack.

# It decreases size by 1

**def** pop(stack):

**if** isEmpty(stack):

**return**

**return** stack.pop()

# A stack based function to reverse a string

**def** reverse(string):

    n **=** len(string)

    # Create a empty stack

    stack **=** createStack()

    # Push all characters of string to stack

**for** i **in** range(0, n, 1):

        push(stack, string[i])

    # Making the string empty since all

    # characters are saved in stack

    string **=** ""

    # Pop all characters of string and put

    # them back to string

**for** i **in** range(0, n, 1):

        string **+=** pop(stack)

**return** string

# Driver code

s **=** "Geeksforgeeks"

**print**("The original string is : ", end**=**"")

print(s)

**print**("The reversed string(using stack) is : ", end**=**"")

print(reverse(s))

**Output**

The original string is : Geeksforgeeks  
The reversed string(using stack) is : skeegrofskeeG

**Reverse string in Python using an extended slice**

Extended slice offers to put a “step” field as[**[start, stop, step]**](https://www.geeksforgeeks.org/python-range-function/), and giving no field as start and stop indicates default to 0 and string length respectively, and “**-1**” denotes starting from the end and stop at the start, hence reversing a string.

**Time complexity**: O(n)

**Auxiliary Space**: O(1)

**Implementation:**

1. Python3

# Function to reverse a string

**def** reverse(string):

    string **=** string[::**-**1]

**return** string

s **=** "Geeksforgeeks"

**print**("The original string is : ", end**=**"")

**print**(s)

**print**("The reversed string(using extended slice syntax) is : ", end**=**"")

print(reverse(s))

**Output**

The original string is : Geeksforgeeks  
The reversed string(using extended slice syntax) is : skeegrofskeeG

**Reverse string in Python using reversed() method**

The [reversed()](https://www.geeksforgeeks.org/python-reversed-function/) returns the reversed iterator of the given string and then its elements are joined empty string separated using join(). And reversed order string is formed.

**Time complexity**: O(n)

**Auxiliary Space**: O(1)

**Implementation:**

1. Python3

# Python code to reverse a string

# using reversed()

# Function to reverse a string

**def** reverse(string):

    string **=** "".join(reversed(string))

**return** string

s **=** "Geeksforgeeks"

print("The original string is : ", end**=**"")

**print**(s)

print("The reversed string(using reversed) is : ", end**=**"")

print(reverse(s))

**Output**

The original string is : Geeksforgeeks  
The reversed string(using reversed) is : skeegrofskeeG

**Reverse string in Python using list comprehension()**

List comprehension creates the list of elements of a string in reverse order and then its elements are joined using [join()](https://www.geeksforgeeks.org/join-function-python/). And reversed order string is formed.

**Time complexity: O(n)**

**Auxiliary Space: O(1)**

**Implementation:**

1. Python3

# Function to reverse a string

**def** reverse(string):

    string **=** [string[i] **for** i **in** range(len(string)**-**1, **-**1, **-**1)]

**return** "".join(string)

s **=** "Geeksforgeeks"

print("The original string  is : ", s)

print("The reversed string(using reversed) is : ", reverse(s))

**Output**

The original string is : Geeksforgeeks  
The reversed string(using reversed) is : skeegrofskeeG

**Reverse string in Python using the function call**

Function to reverse a string by converting string to list then reversed it and again convert it to string.

**Time complexity: O(n)**

**Auxiliary Space: O(1)**

**Implementation:**

1. Python3

# Function to reverse a string

# by converting string to list

# then reversed it and again convert it to string

**def** reverse(string):

    string **=** list(string)

    string.reverse()

**return** "".join(string)

s **=** "Geeksforgeeks"

print("The original string  is : ", s)

print("The reversed string(using reversed) is : ", reverse(s))

# This code is contributed by Susobhan AKhuli

**Output**

The original string is : Geeksforgeeks  
The reversed string(using reversed) is : skeegrofskeeG

*From <*[*https://www.geeksforgeeks.org/reverse-string-python-5-different-ways/*](https://www.geeksforgeeks.org/reverse-string-python-5-different-ways/)*>*

**Python groupby method to remove all consecutive duplicates**

Given a string S, remove all the consecutive duplicates. Examples:

Input : aaaaabbbbbb  
Output : ab

Input : geeksforgeeks  
Output : geksforgeks

Input : aabccba  
Output : abcba

[Recommended: Please try your approach on ***{IDE}*** first, before moving on to the solution.](https://ide.geeksforgeeks.org/)

We have existing solution for this problem please refer [Remove all consecutive duplicates from the string](https://www.geeksforgeeks.org/remove-consecutive-duplicates-string/) link. We can solve this problem in python quickly using [itertools.groupby()](https://docs.python.org/3/library/itertools.html" \l "itertools.groupby) method.

**How itertools.groupby(iterable,key[optional]) works in Python?**

**Group by** method takes two input one is **iterable (list,tuple,dictionary)** and second is key function which calculates keys for each element present in iterable. It returns key and iterable of grouped items. If key function not specified or is None, key defaults to an identity function and returns the element unchanged. For example,

**Python3**

numbers **=** [1, 1, 1, 3, 3, 2, 2, 2, 1, 1]

**import** itertools

**for** (key,group) **in** itertools.groupby(numbers):

    print (key,list(group))

**Output**

1 [1, 1, 1]  
3 [3, 3]  
2 [2, 2, 2]  
1 [1, 1]

**Python3**

# function to remove all consecutive duplicates

# from the string in Python

**from** itertools **import** groupby

**def** removeAllConsecutive(input):

    # group all consecutive characters based on their

    # order in string and we are only concerned

    # about first character of each consecutive substring

    # in given string, so key value will work for us

    # and we will join these keys without space to

    # generate resultant string

    result **=** []

**for** (key,group) **in** groupby(input):

        result.append(key)

    print (''.join(result))

# Driver program

**if** \_\_name\_\_ **==** "\_\_main\_\_":

    input **=** 'aaaaabbbbbb'

    removeAllConsecutive(input)

**Output**

ab

**Time complexity : O(n)**

**Auxiliary Space : O(n)**

**Generate two output strings depending upon occurrence of character in input string in Python**

Given an input string str[], generate two output strings. One of which consists of those character which occurs only once in input string and second which consists of multi-time occurring characters. Output strings must be sorted.

**Examples:**

Input : str = "geeksforgeeks"  
Output : String with characters occurring once:  
"for".  
String with characters occurring multiple times:  
"egks"

Input : str = "geekspractice"  
Output : String with characters occurring once:  
"agikprst"  
String with characters occurring multiple times:  
"ce"

[Recommended: Please try your approach on ***{IDE}*** first, before moving on to the solution.](https://ide.geeksforgeeks.org/)

We have existing solution for this problem please refer [Generate two output strings depending upon occurrence of character in input string](https://www.geeksforgeeks.org/generate-two-output-strings-depending-upon-occurrence-character-input-string/) link. We can solve this problem quickly in python using [Counter(iterable)](https://www.geeksforgeeks.org/counters-in-python-set-1/) method. Approach is simple,

1. Convert string into dictionary having characters as keys and their frequencies as value using **counter()** method.
2. Now separate out list of characters having frequency 1 and having frequency more than 1.
3. Sort characters in both lists to get output strings.

**Implementation:**

**Python3**

# Function Generate two output strings depending upon

# occurrence of character in input string

**from** collections **import** Counter

**def** generateStrings(input):

    # convert string into dictionary

    # having characters as keys and frequency as value

    freqDict **=** Counter(input)

    # separate out characters having frequency 1 and more than 1

    freq1 **=** [ key **for** (key,count) **in** freqDict.items() **if** count**==**1]

    freqMore1 **=** [ key **for** (key,count) **in** freqDict.items() **if** count>1]

    # sort lists and concatenate characters

    # with out space to print resultant strings

    freq1.sort()

    freqMore1.sort()

    # print output strings

    print ('String with characters occurring once:')

    print (''.join(freq1))

    print ('String with characters occurring multiple times:')

    print (''.join(freqMore1))

# Driver program

**if** \_\_name\_\_ **==** "\_\_main\_\_":

    input **=** "geeksforgeeks"

    generateStrings(input)

**Output**

String with characters occurring once:  
for  
String with characters occurring multiple times:  
egks

**Python Dictionary to find mirror characters in a string**

* Difficulty Level : [Easy](https://www.geeksforgeeks.org/easy/)

Given a string and a number N, we need to mirror the characters from the N-th position up to the length of the string in alphabetical order. In mirror operation, we change ‘a’ to ‘z’, ‘b’ to ‘y’, and so on.

**Examples:**

Input : N = 3  
 paradox  
Output : paizwlc  
We mirror characters from position 3 to end.

Input : N = 6  
 pneumonia  
Output : pneumlmrz

[Recommended: Please try your approach on ***{IDE}*** first, before moving on to the solution.](https://ide.geeksforgeeks.org/)

We have an existing solution for this problem please refer to [Mirror characters of a string](https://www.geeksforgeeks.org/mirror-characters-string/) link. We can solve this problem in Python using [Dictionary Data Structure](https://www.youtube.com/watch?v=z7z_e5-l2yE&t=82s). The mirror value of ‘a’ is ‘z’,’b’ is ‘y’, etc, so we create a dictionary data structure and one-to-one map reverse sequence of alphabets onto the original sequence of alphabets. Now traverse characters from length k in given string and change characters into their mirror value using a dictionary.

**Implementation:**

* Python3

# function to mirror characters of a string

**def** mirrorChars(input,k):

    # create dictionary

    original **=** 'abcdefghijklmnopqrstuvwxyz'

    reverse **=** 'zyxwvutsrqponmlkjihgfedcba'

    dictChars **=** dict(zip(original,reverse))

    # separate out string after length k to change

    # characters in mirror

    prefix **=** input[0:k**-**1]

    suffix **=** input[k**-**1:]

    mirror **=** ''

    # change into mirror

**for** i **in** range(0,len(suffix)):

         mirror **=** mirror **+** dictChars[suffix[i]]

    # concat prefix and mirrored part

**print** (prefix**+**mirror)

# Driver program

**if** \_\_name\_\_ **==** "\_\_main\_\_":

    input **=** 'paradox'

    k **=** 3

    mirrorChars(input,k)

**Output**

paizwlc

**Time complexity: O(n)**

**Auxiliary Space: O(n)**

**Python | Convert a list of characters into a string**

Given a list of characters, merge all of them into a string. Examples:

Input : ['g', 'e', 'e', 'k', 's', 'f', 'o',   
 'r', 'g', 'e', 'e', 'k', 's']  
Output : geeksforgeeks

Input : ['p', 'r', 'o', 'g', 'r', 'a', 'm',   
 'm', 'i', 'n', 'g']  
Output : programming

Recommended Practice

[Convert a list of characters into a String](https://practice.geeksforgeeks.org/problems/convert-a-list-of-characters-into-a-string5142/1/)

[Try It!](https://practice.geeksforgeeks.org/problems/convert-a-list-of-characters-into-a-string5142/1/)

Initialize an empty string at the beginning. Traverse in the list of characters, for every index add character to the initial string. After complete traversal, print the string which has been added with every character.

**Implementation:**

* Python

# Python program to convert a list

# of character

**def** convert(s):

    # initialization of string to ""

    new **=** ""

    # traverse in the string

**for** x **in** s:

        new **+=** x

    # return string

**return** new

# driver code

s **=** ['g', 'e', 'e', 'k', 's', 'f', 'o', 'r', 'g', 'e', 'e', 'k', 's']

print(convert(s))

**Output**

geeksforgeeks

**Time Complexity: O(n)**

**Auxiliary Space: O(n)**

**Method 2 : Using join() function**

By using [join()](https://www.geeksforgeeks.org/python-string-methods-set-2-len-count-center-ljust-rjust-isalpha-isalnum-isspace-join/) function in python, all characters in the list can be joined. The syntax is:

**str = ""**  
**str1 = ( "geeks", "for", "geeks" )**  
**str.join(str1)**

The list of characters can be joined easily by initializing str=”” so that there are no spaces in between.

**Implementation:**

* Python

# Python program to convert a list

# of character

**def** convert(s):

    # initialization of string to ""

    str1 **=** ""

    # using join function join the list s by

    # separating words by str1

**return**(str1.join(s))

# driver code

s **=** ['g', 'e', 'e', 'k', 's', 'f', 'o', 'r', 'g', 'e', 'e', 'k', 's']

print(convert(s))

**Output**

geeksforgeeks

**Time Complexity: O(n)**

**Auxiliary Space: O(n)**

**Map function and Lambda expression in Python to replace characters**

Given a string S, c1 and c2. Replace character c1 with c2 and c2 with c1. Examples:

Input : str = 'grrksfoegrrks'  
 c1 = e, c2 = r   
Output : geeksforgeeks

Input : str = 'ratul'  
 c1 = t, c2 = h   
Output : rahul

[Recommended: Please try your approach on ***{IDE}*** first, before moving on to the solution.](https://ide.geeksforgeeks.org/)

We have an existing solution for this problem in C++. Please refer to [Replace a character c1 with c2 and c2 with c1 in a string S](https://www.geeksforgeeks.org/replace-character-c1-c2-c2-c1-string-s/). We can solve this problem quickly in Python using a [Lambda expression](https://www.geeksforgeeks.org/python-lambda-anonymous-functions-filter-map-reduce/) and the [map()](https://www.geeksforgeeks.org/sum-2d-array-python-using-map-function/) function.

We will create a **lambda expression** where character c1 in string will be replaced by c2 and c2 will be replaced by c1. All other characters will remain the same. Then we will **map** this expression on each character of string and return an updated string.

**Implementation:**

* Python3

# Function to replace a character c1 with c2

# and c2 with c1 in a string S

**def** replaceChars(input,c1,c2):

     # create lambda to replace c1 with c2, c2

     # with c1 and other will remain same

     # expression will be like "lambda x:

     # x if (x!=c1 and x!=c2) else c1 if (x==c2) else c2"

     # and map it onto each character of string

     newChars **=** map(**lambda** x: x **if** (x!**=**c1 **and** x!**=**c2) **else** \

                c1 **if** (x**==**c2) **else** c2,input)

     # now join each character without space

     # to print resultant string

     print (''.join(newChars))

# Driver program

**if** \_\_name\_\_ **==** "\_\_main\_\_":

    input **=** 'grrksfoegrrks'

    c1 **=** 'e'

    c2 **=** 'r'

    replaceChars(input,c1,c2)

**Output**

geeksforgeeks

**Zip function in Python to change to a new character set**

* Difficulty Level : [Basic](https://www.geeksforgeeks.org/basic/)

Given a 26 letter character set, which is equivalent to character set of English alphabet i.e. (abcd….xyz) and act as a relation. We are also given several sentences and we have to translate them with the help of given new character set.

**Examples:**

New character set : qwertyuiopasdfghjklzxcvbnm  
Input : "utta"  
Output : geek

Input : "egrt"  
Output : code

[Recommended: Please try your approach on ***{IDE}*** first, before moving on to the solution.](https://ide.geeksforgeeks.org/)

We have existing solution for this problem please refer [Change string to a new character set](https://www.geeksforgeeks.org/change-string-to-a-new-character-set/) link. We will solve this problem in python using [Zip()](https://www.geeksforgeeks.org/using-iterations-in-python-effectively/) method and [Dictionary](https://www.youtube.com/watch?v=z7z_e5-l2yE&t=31s) data structures. Approach is simple,

1. Create a dictionary data structure where we will map original character set in english language with new given character set, **zip(newCharSet,origCharSet)** does it for us. It will map each character of original character set onto each given character of new character set sequentially and return list of tuples of pairs, now we convert it into dictionary using **dict()**.
2. Now iterate through original string and convert it into new string.

**Implementation:**

**Python3**

# Function to change string to a new character

**def** newString(charSet,input):

    # map original character set of english

    # onto new character set given

    origCharSet **=** 'abcdefghijklmnopqrstuvwxyz'

    mapChars **=** dict(zip(charSet,origCharSet))

    # iterate through original string and get

    #  characters of original character set

    changeChars **=** [mapChars[chr] **for** chr **in** input]

    # join characters without space to get new string

**print** (''.join(changeChars))

# Driver program

**if** \_\_name\_\_ **==** "\_\_main\_\_":

    charSet **=** 'qwertyuiopasdfghjklzxcvbnm'

    input **=** 'utta'

    newString(charSet,input)

**Output**

geek

**SequenceMatcher in Python for Longest Common Substring**

* Difficulty Level : [Medium](https://www.geeksforgeeks.org/medium/)

Given two strings ‘X’ and ‘Y’, print the longest common sub-string. Examples:

Input : X = "GeeksforGeeks",   
 Y = "GeeksQuiz"  
Output : Geeks

Input : X = "zxabcdezy",   
 Y = "yzabcdezx"  
Output : abcdez

[Recommended: Please try your approach on ***{IDE}*** first, before moving on to the solution.](https://ide.geeksforgeeks.org/)

We have existing solution for this problem please refer [Print the longest common substring](https://www.geeksforgeeks.org/print-longest-common-substring/) link. We will solve problem in python using SequenceMatcher.find\_longest\_match() method.

**How SequenceMatcher.find\_longest\_match(aLow,aHigh,bLow,bHigh) method works ?**

First we initialize **SequenceMatcher** object with two input string str1 and str2, **find\_longest\_match(aLow,aHigh,bLow,bHigh)** takes 4 parameters aLow, bLow are start index of first and second string respectively and aHigh, bHigh are length of first and second string respectively. find\_longest\_match() returns named tuple (i, j, k) such that a[i:i+k] is equal to b[j:j+k], if no blocks match, this returns (aLow, bLow, 0).

**Implementation:**

* Python3

# Function to find Longest Common Sub-string

**from** difflib **import** SequenceMatcher

**def** longestSubstring(str1,str2):

     # initialize SequenceMatcher object with

     # input string

     seqMatch **=** SequenceMatcher(None,str1,str2)

     # find match of longest sub-string

     # output will be like Match(a=0, b=0, size=5)

     match **=** seqMatch.find\_longest\_match(0, len(str1), 0, len(str2))

     # print longest substring

**if** (match.size!**=**0):

          print (str1[match.a: match.a **+** match.size])

**else**:

**print** ('No longest common sub-string found')

# Driver program

**if** \_\_name\_\_ **==** "\_\_main\_\_":

    str1 **=** 'GeeksforGeeks'

    str2 **=** 'GeeksQuiz'

    longestSubstring(str1,str2)

**Output**

Geeks

**Python | Print the initials of a name with last name in full**

Given a name, print the initials of a name(uppercase) with last name(with first alphabet in uppercase) written in full separated by dots.

**Examples:**

Input : geeks for geeks  
Output : G.F.Geeks

Input : mohandas karamchand gandhi  
Output : M.K.Gandhi

[Recommended: Please try your approach on ***{IDE}*** first, before moving on to the solution.](https://ide.geeksforgeeks.org/)

A **naive approach** of this will be to iterate for spaces and print the next letter after every space except the last space. At last space we have to take all the characters after the last space in a simple approach. Using**Python in inbuilt functions** we can [split](https://www.geeksforgeeks.org/how-to-split-a-string-in-cc-python-and-java/) the words into a list, then traverse till the second last word and print the first character in capitals using [upper()](https://www.geeksforgeeks.org/isupper-islower-lower-upper-python-applications/) function in python and then add the last word using [title()](https://www.geeksforgeeks.org/python-string-methods-set-1-find-rfind-startwith-endwith-islower-isupper-lower-upper-swapcase-title/) function in Python which automatically converts the first alphabet to capital.

**Implementation:**

**Python3**

# python program to print initials of a name

**def** name(s):

    # split the string into a list

    l **=** s.split()

    new **=** ""

    # traverse in the list

**for** i **in** range(len(l)**-**1):

        s **=** l[i]

        # adds the capital first character

        new **+=** (s[0].upper()**+**'.')

    # l[-1] gives last item of list l. We

    # use title to print first character in

    # capital.

    new **+=** l[**-**1].title()

**return** new

# Driver code

s **=**"mohandas karamchand gandhi"

print(name(s))

**Output**

M.K.Gandhi

*From <*[*https://www.geeksforgeeks.org/python-print-initials-name-last-name-full/*](https://www.geeksforgeeks.org/python-print-initials-name-last-name-full/)*>*

**Python counter and dictionary intersection example (Make a string using deletion and rearrangement)**

Given two strings, find if we can make first string from second by deleting some characters from second and rearranging remaining characters.

**Examples:**

Input : s1 = ABHISHEKsinGH  
 : s2 = gfhfBHkooIHnfndSHEKsiAnG  
Output : Possible

Input : s1 = Hello  
 : s2 = dnaKfhelddf  
Output : Not Possible

Input : s1 = GeeksforGeeks  
 : s2 = rteksfoGrdsskGeggehes  
Output : Possible

[Recommended: Please try your approach on ***{IDE}*** first, before moving on to the solution.](https://ide.geeksforgeeks.org/)

We have existing solution for this problem please refer [Make a string from another by deletion and rearrangement of characters](https://www.geeksforgeeks.org/make-string-another-deletion-rearrangement-characters/) link. We will this problem quickly in python. Approach is very simple,

1. Convert both string into dictionary using [Counter(iterable)](https://www.geeksforgeeks.org/counters-in-python-set-1/) method, each dictionary contains characters within string as Key and their frequencies as Value.
2. Now take [intersection](https://www.geeksforgeeks.org/sets-in-python/) of two dictionaries and compare resultant output with dictionary of first string, if both are equal that means it is **possible** to convert string otherwise not.

**Implementation:**

**Python3**

# Python code to find if we can make first string

# from second by deleting some characters from

# second and rearranging remaining characters.

**from** collections **import** Counter

**def** makeString(str1,str2):

    # convert both strings into dictionaries

    # output will be like str1="aabbcc",

    # dict1={'a':2,'b':2,'c':2}

    # str2 = 'abbbcc', dict2={'a':1,'b':3,'c':2}

    dict1 **=** Counter(str1)

    dict2 **=** Counter(str2)

    # take intersection of two dictionaries

    # output will be result = {'a':1,'b':2,'c':2}

    result **=** dict1 & dict2

    # compare resultant dictionary with first

    # dictionary comparison first compares keys

    # and then compares their corresponding values

**return** result **==** dict1

# Driver program

**if** \_\_name\_\_ **==** "\_\_main\_\_":

    str1 **=** 'ABHISHEKsinGH'

    str2 **=** 'gfhfBHkooIHnfndSHEKsiAnG'

**if** (makeString(str1,str2)**==**True):

**print**("Possible")

**else**:

        print("Not Possible")

**Output**

Possible

***Time Complexity:****O(n)*

***Auxiliary Space:****O(1)*

**Python program to count number of vowels using sets in given string**

* Difficulty Level : [Medium](https://www.geeksforgeeks.org/medium/)

Given a string, count the number of vowels present in given string using Sets.

**Prerequisite:**[Sets in Python](https://www.geeksforgeeks.org/sets-in-python/)

Examples:

Input : GeeksforGeeks  
Output : No. of vowels : 5

Input : Hello World  
Output : No. of vowels : 3

[Recommended: Please try your approach on ***{IDE}*** first, before moving on to the solution.](https://ide.geeksforgeeks.org/)

**Approach:**

1. Create a set of vowels using set() and initialize a count variable to 0.

2. Traverse through the alphabets in the string and check if the letter in the string is present in set vowel.

3. If it is present, the vowel count is incremented.

Below is the implementation of above approach:

# Python3 code to count vowel in

# a string using set

# Function to count vowel

**def** vowel\_count(str):

    # Initializing count variable to 0

    count **=** 0

    # Creating a set of vowels

    vowel **=** set("aeiouAEIOU")

    # Loop to traverse the alphabet

    # in the given string

**for** alphabet **in** str:

        # If alphabet is present

        # in set vowel

**if** alphabet **in** vowel:

            count **=** count **+** 1

    print("No. of vowels :", count)

# Driver code

str **=** "GeeksforGeeks"

# Function Call

vowel\_count(str)

Output:

No. of vowels : 5

**Python set to check if string is pangram**

Given a string, check if the given string is pangram or not.

**Examples:**

Input : The quick brown fox jumps over the lazy dog  
Output : The string is a pangram

Input : geeks for geeks  
Output : The string is not pangram

[Recommended: Please try your approach on ***{IDE}*** first, before moving on to the solution.](https://ide.geeksforgeeks.org/)

A normal way would have been to use frequency table and check if all elements were present or not. But using **import ascii\_lowercase as asc\_lower** we import all the lower characters in set and all characters of string in another set. In the function, two sets are formed- one for all lower case letters and one for the letters in the string. The two sets are subtracted and if it is an empty set, the string is a pangram.

Below is Python implementation of the above approach:

**Python**

# import from string all ascii\_lowercase and asc\_lower

**from** string **import** ascii\_lowercase as asc\_lower

# function to check if all elements are present or not

**def** check(s):

**return** set(asc\_lower) **-** set(s.lower()) **==** set([])

# driver code

string **=**"The quick brown fox jumps over the lazy dog"

**if**(check(string)**==** True):

**print**("The string is a pangram")

**else**:

    print("The string isn't a pangram")

**Output**

The string is a pangram

**Time Complexity: O(n)**

**Auxiliary Space: O(1)**

**Python | Check if a Substring is Present in a Given String**

In this article, we will cover how to check if a Python string contains another string or a substring in Python. Given two strings, check if a substring is there in the given string or not.

Example 1: Input : Substring = "geeks"   
 String="geeks for geeks"  
Output : yes  
Example 2: Input : Substring = "geek"  
 String="geeks for geeks"  
Output : yes

**Does Python have a string containing the substring method**

Yes, Checking a substring is one of the most used tasks in python. Python uses many methods to check a string containing a substring like, find(), index(), count(), etc. The most efficient and fast method is by using an “**in**” operator which is used as a comparison operator. Here we will cover different approaches like:

* Using the if… in
* Checking using the [split() method](https://www.geeksforgeeks.org/python-string-split/)
* Using [find() method](https://www.geeksforgeeks.org/python-string-find/#:~:text=Python%20String%20find()%20method%20returns%20the%20lowest%20index%20or,found%20then%20it%20returns%20%2D1.&text=Parameters%3A,searched%20in%20the%20given%20string.)
* Using “[count()” method](https://www.geeksforgeeks.org/python-string-count-method/)
* Using the [index() method](https://www.geeksforgeeks.org/python-list-index/#:~:text=index()%20is%20an%20inbuilt,index%20where%20the%20element%20appears.&text=Parameters%3A,lowest%20index%20will%20be%20returned.)
* Using  \_\_contains\_\_” magic class.
* Using [regular expressions](https://www.geeksforgeeks.org/python-regex/)

**Method 1: Check substring using the if… in.**

**Python3**

# Take input from users

MyString1 **=** "A geek in need is a geek indeed"

**if** "need" **in** MyString1:

**print**("Yes! it is present in the string")

**else**:

    print("No! it is not present")

**Output**

Yes! it is present in the string

**Method 2: Checking substring using the split() method**

Checking if a substring is present in the given string or not without using any inbuilt function. First split the given string into words and store them in a variable s then using the if condition, check if a substring is present in the given string or not.

**Python3**

# Python code

# To check if a substring is present in a given string or not

# input strings str1 and substr

string **=** "geeks for geeks"  # or string=input() -> taking input from the user

substring **=** "geeks"  # or substring=input()

# splitting words in a given string

s **=** string.split()

# checking condition

# if substring is present in the given string then it gives output as yes

**if** substring **in** s:

**print**("yes")

**else**:

    print("no")

**Output**

yes

**Method 3: Check substring using the find() method**

We can iteratively check for every word, but Python provides us an inbuilt function [find()](https://www.geeksforgeeks.org/python-string-methods-set-1-find-rfind-startwith-endwith-islower-isupper-lower-upper-swapcase-title/) which checks if a substring is present in the string, which is done in one line. find() function returns -1 if it is not found, else it returns the first occurrence, so using this function this problem can be solved.

**Python3**

# function to check if small string is

# there in big string

**def** check(string, sub\_str):

**if** (string.find(sub\_str) **== -**1):

        print("NO")

**else**:

        print("YES")

# driver code

string **=** "geeks for geeks"

sub\_str **=** "geek"

check(string, sub\_str)

**Output**

YES

**Method 4: Check substring using “count()” method**

You can also count the number of occurrences of a specific substring in a string, then you can use the Python count() method. If the substring is not found then “yes ” will print otherwise “no will be printed”.

**Python3**

**def** check(s2, s1):

**if** (s2.count(s1) > 0):

**print**("YES")

**else**:

        print("NO")

s2 **=** "A geek in need is a geek indeed"

s1 **=** "geeks"

check(s2, s1)

**Output**

NO

**Method 5: Check substring using the index() method**

The **.index() method**returns the starting index of the substring passed as a parameter. Here “**substring**” is present at index 16.

**Python3**

any\_string **=** "Geeks for Geeks substring "

start **=** 0

end **=** 1000

print(any\_string.index('substring', start, end))

**Output:**

16

**Method 6: Check substring using the “\_\_contains\_\_” magic class.**

Python String \_\_contains\_\_(). This method is used to check if the string is present in the other string or not.

**Python3**

a **=** ['Geeks-13', 'for-56', 'Geeks-78', 'xyz-46']

**for** i **in** a:

**if** i.\_\_contains\_\_("Geeks"):

        print(f"Yes! {i} is containing.")

**Output**

Yes! Geeks-13 is containing.  
Yes! Geeks-78 is containing.

**Method 7: Check substring using regular expressions**

RegEx can be used to check if a string contains the specified search pattern. Python has a built-in package called **re**, which can be used to work with Regular Expressions.

**Python3**

# When you have imported the re module,

# you can start using regular expressions.

**import** re

# Take input from users

MyString1 **=** "A geek in need is a geek indeed"

MyString2 **=** "geeks"

# re.search() returns a Match object

# if there is a match anywhere in the string

**if** re.search(MyString2, MyString1):

    print("YES,string '{0}' is present in string '{1}'" .format(

        MyString2, MyString1))

**else**:

    print("NO,string '{0}' is not present in string '{1}' " .format(

        MyString2, MyString1))

**Output**

NO,string 'geeks' is not present in string 'A geek in need is a geek indeed'

**Method: Using list comprehension**

**Python3**

s**=**"geeks for geeks"

s2**=**"geeks"

print(["yes" **if** s2 **in** s **else** "no"])

**Output**

['yes']

**Method: Using lambda function**

**Python3**

s**=**"geeks for geeks"

s2**=**"geeks"

x**=**list(filter(**lambda** x: (s2 **in** s),s.split()))

print(["yes" **if** x **else** "no"])

**Output**

['yes']

**Method:**Using **countof**function

**Python3**

**import** operator as op

s**=**"geeks for geeks"

s2**=**"geeks"

print(["yes" **if** op.countOf(s.split(),s2)>0 **else** "no"])

**Output**

['yes']

**Python Program to Check if Two Strings are Anagram**

**Question:**

Given two strings s1 and s2, check if both the strings are [anagrams](https://www.geeksforgeeks.org/check-whether-two-strings-are-anagram-of-each-other/) of each other.

**Examples:**

**Input :** s1 = "listen"  
 s2 = "silent"  
**Output :** The strings are anagrams.

**Input :** s1 = "dad"  
 s2 = "bad"  
**Output :** The strings aren't anagrams.

[Recommended: Please try your approach on ***{IDE}*** first, before moving on to the solution.](https://ide.geeksforgeeks.org/)

**Solution:**

**Method #1 : Using sorted() function**

Python provides a inbuilt function [sorted()](https://www.geeksforgeeks.org/sorted-function-python/) which does not modify the original string, but returns sorted string.

Below is the Python implementation of the above approach:

**Python**

# function to check if two strings are

# anagram or not

**def** check(s1, s2):

    # the sorted strings are checked

**if**(sorted(s1)**==** sorted(s2)):

**print**("The strings are anagrams.")

**else**:

**print**("The strings aren't anagrams.")

# driver code

s1 **=**"listen"

s2 **=**"silent"

check(s1, s2)

**Output**

The strings are anagrams.

**Time Complexity:**O(nlogn)

**Auxiliary Space:**O(1)

**Method #2 : Using Counter() function**

* Count all the frequencies of 1st string and 2 and using counter()
* If they are equal then print anagram

**Python3**

# Python3 program for the above approach

**from** collections **import** Counter

# function to check if two strings are

# anagram or not

**def** check(s1, s2):

    # implementing counter function

**if**(Counter(s1) **==** Counter(s2)):

**print**("The strings are anagrams.")

**else**:

        print("The strings aren't anagrams.")

# driver code

s1 **=** "listen"

s2 **=** "silent"

check(s1, s2)

**Output**

The strings are anagrams.

**Time Complexity: O(n)**

**Auxiliary Space: O(1)**

**Method #3: Using Inbuilt List and Sort() Methods**

Taking 2 User Inputs and appending to a list and then sorting the elements in a list and it checks If they are equal then print anagram else aren’t anagrams

**Python3**

## Example 1 for "The strings are anagrams."

#Declare Inputs

inp1 **=** "listen"

inp2 **=** "silent"

#Sort Elements

x **=** [inp1[i] **for** i **in** range(0,len(inp1))]

x.sort()

y **=** [inp2[i] **for** i **in** range(0,len(inp2))]

y.sort()

# the sorted strings are checked

**if** (x **==** y):**print**("The strings are anagrams.")

**else**: **print**("The strings aren't anagrams.")

## Example 2 for "The strings aren't anagrams."

#Declare Inputs

inp1 **=** "listen"

inp2 **=** "silenti"

#Sort Elements

x **=** [inp1[i] **for** i **in** range(0,len(inp1))]

x.sort()

y **=** [inp2[i] **for** i **in** range(0,len(inp2))]

y.sort()

# the sorted strings are checked

**if** (x **==** y):**print**("The strings are anagrams.")

**else**: print("The strings aren't anagrams.")

**Output**

The strings are anagrams.  
The strings aren't anagrams.

**Time Complexity: O(nlogn)**

**Auxiliary Space: O(n)**

**Method #4: Using a dictionary to achieve constant time complexity**

A dictionary in python provides constant time lookup. The algorithm is very simple and works like this:

1. Loop through the characters of the two strings simultaneously. Suppose c1 is the current character of the first string and c2 the current character of the second string.
2. For each of c1 and c2, say c, do the following:
3. If c is not present as a key in the dictionary, then create a dictionary entry with the key being c and the value being

* 1  if c is c1 (current character of the first string)
* -1 if c is c2 (current character of the second string)

1. If c is present as a key in the dictionary, then instead of creating a new key-value pair, just increase or decrease the value in a similar way as in the previous step.
2. After that, loop through the dictionary. If there is a key-value pair in the dictionary where it’s value is different than 0, the two strings and not anagram. Otherwise, they are.

**Python3**

**def** anagrams(inp1: str, inp2: str) **-**> bool:

  # if the length of the two strings is not the same, they are not anagrams.

**if** len(inp1) !**=** len(inp2):

**return** False

  # initialize the dictionary

  counts **=** {}

  # loop simultaneously through the characters of the two strings.

**for** c1, c2 **in** zip(inp1, inp2):

**if** c1 **in** counts.keys():

      counts[c1] **+=** 1

**else**:

      counts[c1] **=** 1

**if** c2 **in** counts.keys():

      counts[c2] **-=** 1

**else**:

      counts[c2] **= -**1

  # Loop through the dictionary values.

  # if the dictionary contains even one value which is

  # different than 0, the strings are not anagrams.

**for** count **in** counts.values():

**if** count !**=** 0:

**return** False

**return** True

# test the implementation

**def** main():

  inp1 **=** "listen"

  inp2 **=** "silent"

**if** anagrams(inp1, inp2):

**print**(f"{inp1} and {inp2} are anagrams")

**else**:

**print**(f"{inp1} and {inp2} are not anagrams")

**if** \_\_name\_\_ **==** "\_\_main\_\_":

  main()

**Output**

listen and silent are anagrams

**Time Complexity: O(n) since we have to loop through all characters for both of the strings.**

**Auxiliary Space: O(n) in the worst case where all letters are different for even one of the string inputs. If both string inputs have eg. 4 letters out of which only 2 are different (eg. “SaaS” and “SaSa”), the space needed would be O(n/2)**

**Python | C++ | Remove leading zeros from an IP address**

Given an IP address, remove leading zeros from the IP address.

**Examples:**

Input : 100.020.003.400   
Output : 100.20.3.400

Input :001.200.001.004   
Output : 1.200.1.4

Recommended Problem

Remove leading zeros from an IP address

The approach is to split the given string by “.” and then convert it to an integer which removes the leading zeros and then join back them to a string.To convert a string to an integer we can use int(s) and then convert it back to string by str(s) and then join them back by using join function.

**Implementation:**

1. C++
2. Python

# Python program to remove leading zeros

# an IP address and print the IP

# function to remove leading zeros

**def** removeZeros(ip):

    # splits the ip by "."

    # converts the words to integers to remove leading removeZeros

    # convert back the integer to string and join them back to a string

    new\_ip **=** ".".join([str(int(i)) **for** i **in** ip.split(".")])

**return** new\_ip ;

# driver code

# example1

ip **=**"100.020.003.400"

print(removeZeros(ip))

# example2

ip **=**"001.200.001.004"

print(removeZeros(ip))

**Output**

100.20.3.400  
1.200.1.4

**Method 2 :**[**Regex**](https://www.geeksforgeeks.org/regular-expression-python-examples-set-1/)

Using a capture group, match the last digit and copy it and prevents all the digits from being replaced.

[regex \d](https://www.geeksforgeeks.org/regular-expression-python-examples-set-1/) can be explained as:

1. **\d :** Matches any decimal digit

**\d Matches any decimal digit, this is equivalent**  
 **to the set class [0-9].**

1. \b allows you to perform a “whole words only” search using a regular expression in the form of \bword\b   
   **regex \b can be explained as:**

**\b allows you to perform a "whole words only" search u**  
**sing a regular expression in the form of \bword\b**

**Implementation:**

1. Python

# Python program to remove leading zeros

# an IP address and print the IP using regex

**import** re

# function to remove leading zeros

**def** removeZeros(ip):

    new\_ip **=** re.sub(r'\b0+(\d)', r'\1', ip)

    # splits the ip by "."

    # converts the words to integers to remove leading removeZeros

    # convert back the integer to string and join them back to a string

**return** new\_ip

# driver code

# example1

ip **=**"100.020.003.400"

print(removeZeros(ip))

# example2

ip **=**"001.200.001.004"

print(removeZeros(ip))

**Output**

100.20.3.400  
1.200.1.4

**Python | Count all prefixes in given string with greatest frequency**

Given a string, print and count all prefixes in which first alphabet has greater frequency than second alphabet.

Take two alphabets from the user and compare them. The prefixes in which the alphabet given first has greater frequency than the second alphabet, such prefixes are printed, else the result will be 0.

**Examples :**

**Input :** string1 = "geek",   
 alphabet1 = "e", alphabet2 = "k"  
**Output :**  
ge  
gee  
geek  
3

**Input :** string1 = "geek",  
 alphabet1 = "k", alphabet2 = "e"  
**Output :**  
0

[Recommended: Please try your approach on ***{IDE}*** first, before moving on to the solution.](https://ide.geeksforgeeks.org/)

**Approach:** Take an empty string to store the string values of all the prefixes formed. Then check for the alphabet with greater frequency than the second alphabet. If no such case is found then the result will be **0** prefixes.

**Implementation:**

**Python3**

# Python program to Count all

# prefixes in given string with

# greatest frequency

# Function to print the prefixes

**def** prefix(string1, alphabet1, alphabet2):

    count **=** 0

    non\_empty\_string **=** ""

    string2 **=** list(string1)

    # Loop for iterating the length of

    # the string and print the prefixes

    # and the count of query prefixes.

**for** i **in** range(0, len(string2)):

        non\_empty\_string **=** non\_empty\_string **+** (string2[i])

**if** (non\_empty\_string.count(alphabet1) >

            non\_empty\_string.count(alphabet2)):

            # prints all required prefixes

            print(non\_empty\_string)

            # increment count

            count **+=** 1

    # returns count of the

    # required prefixes

**return**(count)

# Driver Code

**print**(prefix("geeksforgeeks", "e", "g"))

**Output**

gee  
geek  
geeks  
geeksf  
geeksfo  
geeksfor  
geeksforge  
geeksforgee  
geeksforgeek  
geeksforgeeks  
10

**Complexity Analysis:**

1. **Time Complexity: O(N),**where N is the length of the string.
2. **Auxiliary Space: O(N)**

**Check if both halves of the string have same set of characters in Python**

Given a string of lowercase characters only, the task is to check if it is possible to split a string from middle which will gives two halves having the same characters and same frequency of each character. If the length of the given string is ODD then ignore the middle element and check for the rest. Examples:

Input : abbaab  
Output : NO  
The two halves contain the same characters  
but their frequencies do not match so they  
are NOT CORRECT

Input : abccab  
Output : YES

[Recommended: Please try your approach on ***{IDE}*** first, before moving on to the solution.](https://ide.geeksforgeeks.org/)

This problem has existing solution, please refer [Check if both halves of the string have same set of characters](https://www.geeksforgeeks.org/check-half-string-character-frequency-character/) link. We will solve this problem in Python quickly using [**Dictionary**](https://www.youtube.com/watch?v=z7z_e5-l2yE&t=29s) comparison. Approach is very simple :

1. Break string in two parts and convert both parts into dictionary using [Counter(iterator)](https://www.geeksforgeeks.org/counters-in-python-set-1/) method, each dictionary contains it’s character as key and frequency as value.
2. Now compare these two dictionaries. In python we can compare two using **==** operator, it first checks keys of both dictionaries are same or not, then it checks for values of each key. If everything is equal that means two dictionaries are identical.

**Implementation:**

**Python3**

# Function to Check if both halves of

# the string have same set of characters

**from** collections **import** Counter

**def** checkTwoHalves(input):

    length **=** len(input)

    # Break input string in two parts

**if** (length **%** 2 !**=** 0):

        first **=** input[0:length **//** 2]

        second **=** input[(length **//** 2) **+** 1:]

**else**:

        first **=** input[0:length **//** 2]

        second **=** input[length **//** 2:]

    # Convert both halves into dictionary and compare

**if** Counter(first) **==** Counter(second):

**print** ('YES')

**else**:

**print** ('NO')

# Driver program

**if** \_\_name\_\_ **==** "\_\_main\_\_":

    input **=** 'abbaab'

    checkTwoHalves(input)

**Output**

NO

**Time Complexity: O(n)**

**Auxiliary Space: O(n)**

**Concatenated string with uncommon characters in Python**

Two strings are given and you have to modify 1st string such that all the common characters of the 2nd string have to be removed and the uncommon characters of the 2nd string have to be concatenated with uncommon characters of the 1st string.

**Examples:**

Input : S1 = "aacdb"  
 S2 = "gafd"  
Output : "cbgf"

Input : S1 = "abcs";  
 S2 = "cxzca";  
Output : "bsxz"

[Recommended: Please try your approach on ***{IDE}*** first, before moving on to the solution.](https://ide.geeksforgeeks.org/)

This problem has existing solution please refer [Concatenated string with uncommon characters of two strings](https://www.geeksforgeeks.org/concatenated-string-uncommon-characters-two-strings/) link. We can solve this problem quickly in Python using [Set](https://www.geeksforgeeks.org/sets-in-python/) and [List Comprehension](https://www.geeksforgeeks.org/python-list-comprehension-and-slicing/). Approach is simple,

1. Convert both strings into set so that they could have only unique characters. Now take intersection of two sets to get common character both strings have.
2. Now separate out those characters in each string which are not common in both of them and concatenate the characters.

**Implementation:**

* Python3

# Function to concatenated string with uncommon

# characters of two strings

**def** uncommonConcat(str1, str2):

    # convert both strings into set

    set1 **=** set(str1)

    set2 **=** set(str2)

    # take intersection of two sets to get list of

    # common characters

    common **=** list(set1 & set2)

    # separate out characters in each string

    # which are not common in both strings

    result **=** [ch **for** ch **in** str1 **if** ch **not in** common] **+** [ch **for** ch **in** str2 **if** ch **not in** common]

    # join each character without space to get

    # final string

    print( ''.join(result) )

# Driver program

**if** \_\_name\_\_ **==** "\_\_main\_\_":

    str1 **=** 'aacdb'

    str2 **=** 'gafd'

    uncommonConcat(str1,str2)

**Output**

cbgf

**Approach#2: Using set symmetric difference**

We can use the symmetric difference operation of set to pull out all the uncommon character from both the string and make a string.

**Implementation:**

* Python3

# Function to concatenated string with uncommon

# characters of two strings

**def** uncommonConcat(str1, str2):

    # convert both strings into set

    set1 **=** set(str1)

    set2 **=** set(str2)

    # Performing symmetric difference operation of set

    # to pull out uncommon characters

    uncommon **=** list(set1 ^ set2)

    # join each character without space to get

    # final string

**print**( ''.join(uncommon) )

# Driver program

**if** \_\_name\_\_ **==** "\_\_main\_\_":

    str1 **=** 'aacdb'

    str2 **=** 'gafd'

    uncommonConcat(str1,str2)

**Output**

cbgf

**Second most repeated word in a sequence in Python**

Given a sequence of strings, the task is to find out the second most repeated (or frequent) string in the given sequence. (Considering no two words are the second most repeated, there will be always a single word).

**Examples:**

Input : {"aaa", "bbb", "ccc", "bbb",   
 "aaa", "aaa"}  
Output : bbb

Input : {"geeks", "for", "geeks", "for",   
 "geeks", "aaa"}  
Output : for

[Recommended: Please try your approach on ***{IDE}*** first, before moving on to the solution.](https://ide.geeksforgeeks.org/)

This problem has existing solution please refer [Second most repeated word in a sequence](https://www.geeksforgeeks.org/second-repeated-word-sequence/) link. We can solve this problem quickly in Python using [Counter(iterator)](https://www.geeksforgeeks.org/counters-in-python-set-1/) method.

Approach is very simple –

1. Create a dictionary using **Counter(iterator)** method which contains words as keys and it’s frequency as value.
2. Now get a list of all values in dictionary and sort it in descending order. Choose second element from the sorted list because it will be the second largest.
3. Now traverse dictionary again and print key whose value is equal to second largest element.

**Implementation**

**Python3**

# Python code to print Second most repeated

# word in a sequence in Python

**from** collections **import** Counter

**def** secondFrequent(input):

    # Convert given list into dictionary

    # it's output will be like {'ccc':1,'aaa':3,'bbb':2}

    dict **=** Counter(input)

    # Get the list of all values and sort it in ascending order

    value **=** sorted(dict.values(), reverse**=**True)

    # Pick second largest element

    secondLarge **=** value[1]

    # Traverse dictionary and print key whose

    # value is equal to second large element

**for** (key, val) **in** dict.items():

**if** val **==** secondLarge:

            print(key)

**return**

# Driver program

**if** \_\_name\_\_ **==** "\_\_main\_\_":

    input **=** ['aaa', 'bbb', 'ccc', 'bbb', 'aaa', 'aaa']

    secondFrequent(input)

**Output**

bbb

**Time complexity: O(nlogn)** where n is the length of the input list

**Auxiliary space: O(n)**where n is the length of the input list

**Alternate Implementation :**

**Python3**

# returns the second most repeated word

**from** collections **import** Counter

**class** Solution:

**def** secFrequent(self, arr, n):

        all\_freq **=** dict(Counter(arr))

        store **=** []

**for** w **in** sorted(all\_freq, key**=**all\_freq.get):

            # if add key=all\_freq.get will sort according to values

            # without key=all\_freq.get will sort according to keys

**if** w **not in** store:

                store.append(w)

**return** store[**-**2]

# driver code or main function

**if** \_\_name\_\_ **==** '\_\_main\_\_':

    # no. of test cases

    t **=** 1

**for** \_ **in** range(t):

        # no of words

        n **=** 7

        # String of words

        arr **=** ["cat","mat","cat","mat","cat",'ball',"tall"]

        ob **=** Solution()

        ans **=** ob.secFrequent(arr,n)

**print**(ans)

**Output**

mat

**Time complexity:** **O(nlogn)**

**Auxiliary space: O(n)**

**Regex in Python to put spaces between words starting with capital letters**

Given an array of characters, which is basically a sentence. However, there is no space between different words and the first letter of every word is in uppercase. You need to print this sentence after the following amendments:

1. Put a single space between these words.
2. Convert the uppercase letters to lowercase

**Examples:**

Input : BruceWayneIsBatman  
Output : bruce wayne is batman

Input : GeeksForGeeks  
Output : geeks for geeks

We have an existing solution for this problem, please refer [Put spaces between words starting with capital letters](https://www.geeksforgeeks.org/put-spaces-words-starting-capital-letters/) link.

[Recommended: Please try your approach on ***{IDE}*** first, before moving on to the solution.](https://ide.geeksforgeeks.org/)

We can solve this problem quickly in python using [findall()](https://www.geeksforgeeks.org/regular-expressions-python-set-1-search-match-find/) method of [re (regex) library](https://www.geeksforgeeks.org/regular-expression-python-examples-set-1/).

**Approach :**

1. Split each word starting with a capital letter using **re.findall(expression, str)** method.
2. Now change the capital letter of each word to lowercase and concatenate each word with space.

**Implementation:**

**Python3**

**import** re

**def** putSpace(input):

    # regex [A-Z][a-z]\* means any string starting

    # with capital character followed by many

    # lowercase letters

    words **=** re.findall('[A-Z][a-z]\*', input)

    # Change first letter of each word into lower

    # case

**for** i **in** range(0,len(words)):

      words[i]**=**words[i][0].lower()**+**words[i][1:]

**print**(' '.join(words))

# Driver program

**if** \_\_name\_\_ **==** "\_\_main\_\_":

    input **=** 'BruceWayneIsBatman'

    putSpace(input)

**Output**

bruce wayne is batman

**Time Complexity: O(n)**

**Auxiliary Space: O(n)**

**Python code to move spaces to front of string in single traversal**

Given a string that has set of words and spaces, write a program to move all spaces to front of string, by traversing the string only once. Examples:

Input : str = "geeks for geeks"  
Output : str = " geeksforgeeks"

Input : str = "move these spaces to beginning"  
Output : str = " movethesespacestobeginning"  
There were four space characters in input,  
all of them should be shifted in front.

[Recommended: Please try your approach on ***{IDE}*** first, before moving on to the solution.](https://ide.geeksforgeeks.org/)

This problem has existing solution, please refer [Move spaces to front of string in single traversal](https://www.geeksforgeeks.org/move-spaces-front-string-single-traversal/) link. We will solve this problem quickly in Python using [List Comprehension](https://www.geeksforgeeks.org/python-list-comprehension-and-slicing/).

***Approach 1****:*

1. *Traverse input string and create a string without any space character using list comprehension.*
2. *Now to know how many space characters were there in original string just take a difference of length of original string and new string.*
3. *Now create another string and append space characters at the beginning.*

**Implementation**

**Python3**

# Function to move spaces to front of string

# in single traversal in Python

**def** moveSpaces(input):

    # Traverse string to create string without spaces

    noSpaces **=** [ch **for** ch **in** input **if** ch!**=**' ']

    # calculate number of spaces

    space**=** len(input) **-** len(noSpaces)

    # create result string with spaces

    result **=** ' '**\***space

    # concatenate spaces with string having no spaces

    result **=** '"'+result + ''.join(noSpaces)+'"'

**print** (result)

# Driver program

**if** \_\_name\_\_ **==** "\_\_main\_\_":

    input **=** 'geeks for geeks'

    moveSpaces(input)

**Output:**

" geeksforgeeks"

**Time Complexity: O(n)**

**Auxiliary Space: O(n)**

**Approach 2 :**Using **count()** and **replace()** method

**Implementation**

**Python3**

# Function to move spaces to front of string

# in single traversal in Python

**def** moveSpaces(input):

    c**=**input.count(' ')

    input**=**input.replace(' ','')

    input**=**' '**\***c**+**input

    print(input)

# Driver program

**if** \_\_name\_\_ **==** "\_\_main\_\_":

    input **=** 'geeks for geeks'

    moveSpaces(input)

**Output**

geeksforgeeks

**String slicing in Python to rotate a string**

Given a string of size n, write functions to perform following operations on string.

1. Left (Or anticlockwise) rotate the given string by d elements (where d <= n).
2. Right (Or clockwise) rotate the given string by d elements (where d <= n).

**Examples:**

Input : s = "GeeksforGeeks"  
 d = 2  
Output : Left Rotation : "eksforGeeksGe"   
 Right Rotation : "ksGeeksforGee"

Input : s = "qwertyu"   
 d = 2  
Output : Left rotation : "ertyuqw"  
 Right rotation : "yuqwert"

[Recommended: Please try your approach on ***{IDE}*** first, before moving on to the solution.](https://ide.geeksforgeeks.org/)

**Method 1:**We have existing solution for this problem please refer [Left Rotation and Right Rotation of a String](https://www.geeksforgeeks.org/left-rotation-right-rotation-string-2/) link. We will solve this problem quickly in python using [String Slicing](https://www.geeksforgeeks.org/interesting-facts-about-strings-in-python-set-2/). Approach is very simple,

1. Separate string in two parts **first & second**, for **Left rotation** Lfirst = str[0 : d] and Lsecond = str[d :]. For **Right rotation** Rfirst = str[0 : len(str)-d] and Rsecond = str[len(str)-d : ].
2. Now concatenate these two parts **second + first** accordingly.

**Implementation:**

**Python3**

# Function to rotate string left and right by d length

**def** rotate(input,d):

    # slice string in two parts for left and right

    Lfirst **=** input[0 : d]

    Lsecond **=** input[d :]

    Rfirst **=** input[0 : len(input)**-**d]

    Rsecond **=** input[len(input)**-**d : ]

    # now concatenate two parts together

**print** ("Left Rotation : ", (Lsecond **+** Lfirst) )

**print** ("Right Rotation : ", (Rsecond **+** Rfirst))

# Driver program

**if** \_\_name\_\_ **==** "\_\_main\_\_":

    input **=** 'GeeksforGeeks'

    d**=**2

    rotate(input,d)

**Output:**

Left Rotation : eksforGeeksGe   
Right Rotation : ksGeeksforGee

**Method 2:**We use extended string to rotate the string. We will solve this problem quickly in python by slicing extended string. Approach is very simple,

Use extended string Extend\_str, for Left rotation Lfirst = Extended\_str[n : l1+n] . For Right rotation Rfirst = str[l1-n : l2-n].

Now print this string.

**Implementation:**

**Python3**

# Function to rotate string left and right by d length

**def** rotate(str1,n):

    # Create the extended string and index of for rotation

    temp **=** str1 **+** str1

    l1 **=** len(str1)

    l2 **=** len(temp)

    Lfirst **=** temp[n  : l1**+**n]

    Lfirst **=** temp[l1**-**n : l2**-**n]

    # now printing the string

**print** ("Left Rotation : ", Lfirst)

    print ("Right Rotation : ", Lfirst )

# Driver program

**if** \_\_name\_\_ **==** "\_\_main\_\_":

    input **=** 'GeeksforGeeks'

    d**=**2

    rotate(input,d)

**Output**

Left Rotation : ksGeeksforGee  
Right Rotation : ksGeeksforGee

*From <*[*https://www.geeksforgeeks.org/string-slicing-python-rotate-string/*](https://www.geeksforgeeks.org/string-slicing-python-rotate-string/)*>*

**String slicing in Python to check if a string can become empty by recursive deletion**

Given a string “str” and another string “sub\_str”. We are allowed to delete “sub\_str” from “str” any number of times. It is also given that the “sub\_str” appears only once at a time. The task is to find if “str” can become empty by removing “sub\_str” again and again. Examples:

Input : str = "GEEGEEKSKS", sub\_str = "GEEKS"  
Output : Yes  
Explanation : In the string GEEGEEKSKS, we can first   
 delete the substring GEEKS from position 4.  
 The new string now becomes GEEKS. We can   
 again delete sub-string GEEKS from position 1.   
 Now the string becomes empty.

Input : str = "GEEGEEKSSGEK", sub\_str = "GEEKS"  
Output : No  
Explanation : In the string it is not possible to make the  
 string empty in any possible manner.

[Recommended: Please try your approach on ***{IDE}*** first, before moving on to the solution.](https://ide.geeksforgeeks.org/)

We have existing solution for this problem please refer [Check if a string can become empty by recursively deleting a given sub-string](https://www.geeksforgeeks.org/check-string-can-become-empty-recursively-deleting-given-sub-string/) link. We will solve this problem in python using [String Slicing](https://www.geeksforgeeks.org/how-to-split-a-string-in-cc-python-and-java/). Approach is very simple,

1. Use [find()](https://www.geeksforgeeks.org/python-string-methods-set-1-find-rfind-startwith-endwith-islower-isupper-lower-upper-swapcase-title/) method of string to search given pattern sub-string.
2. If sub-string lies in main string then find function will return index of it’s first occurrence.
3. Now slice string in two parts, (i) from start of string till index-1 of founded sub-string, (ii) (start from first index of founded sub-string + length of sub-string) till end of string.
4. Concatenate these two sliced part and repeat from step 1 until original string becomes empty or we don’t find sub-string anymore.

**Implementation:**

* Python3

**def** checkEmpty(input, pattern):

    # If both are empty

**if** len(input)**==** 0 **and** len(pattern)**==** 0:

**return** 'true'

    # If only pattern is empty

**if** len(pattern)**==** 0:

**return** 'true'

**while** (len(input) !**=** 0):

        # find sub-string in main string

        index **=** input.find(pattern)

        # check if sub-string founded or not

**if** (index **==**(**-**1)):

**return** 'false'

        # slice input string in two parts and concatenate

        input **=** input[0:index] **+** input[index **+** len(pattern):]

**return** 'true'

# Driver program

**if** \_\_name\_\_ **==** "\_\_main\_\_":

    input **=**'GEEGEEKSKS'

    pattern **=**'GEEKS'

    print (checkEmpty(input, pattern))

**Output**

true

**Time Complexity: O(n/m)**, where n is the length of string, and m is the length of substring

**Auxiliary Space: O(1)**

**Reverse words in a given String in Python**

We are given a string and we need to reverse words of a given string

**Examples:**

Input : str =" geeks quiz practice code"  
Output : str = code practice quiz geeks    
Input : str = "my name is laxmi"  
output : str= laxmi is name my

reverse the words in the given string program

**Python3**

# Python code

# To reverse words in a given string

# input string

string **=** "geeks quiz practice code"

# reversing words in a given string

s **=** string.split()[::**-**1]

l **=** []

**for** i **in** s:

    # apending reversed words to l

    l.append(i)

# printing reverse words

print(" ".join(l))

**Output**

code practice quiz geeks

**Time Complexity:** **O(n)**, where n is the length of the string

**Auxiliary Space: O(n)**, where n is the length of the string

We are given a string and we need to reverse words of a given string? Examples:

Input : str = geeks quiz practice code  
Output : str = code practice quiz geeks

[Recommended: Please try your approach on ***{IDE}*** first, before moving on to the solution.](https://ide.geeksforgeeks.org/)

This problem has an existing solution please refer [Reverse words in a given String](https://www.geeksforgeeks.org/reverse-words-in-a-given-string/) link. We will solve this problem in python. Given below are the steps to be followed to solve this problem.

* Separate each word in a given string using [split()](https://www.geeksforgeeks.org/how-to-split-a-string-in-cc-python-and-java/) method of string data type in python.
* Reverse the word separated list.
* Print words of the list, in string form after joining each word with space using [” “.join()](https://www.geeksforgeeks.org/python-string-methods-set-2-len-count-center-ljust-rjust-isalpha-isalnum-isspace-join/) method in python.

**Implementation:**

**Python3**

# Function to reverse words of string

**def** rev\_sentence(sentence):

    # first split the string into words

    words **=** sentence.split(' ')

    # then reverse the split string list and join using space

    reverse\_sentence **=** ' '.join(reversed(words))

    # finally return the joined string

**return** reverse\_sentence

**if** \_\_name\_\_ **==** "\_\_main\_\_":

    input **=** 'geeks quiz practice code'

    print (rev\_sentence(input))

**Output**

code practice quiz geeks

**Time Complexity:** **O(n),** where n is the length of the string

**Auxiliary Space: O(n),** where n is the length of the string

This solution use the same procedure but have different methods to reverse the words in string with the help backward iteration and regular expression module. Following are the steps of our approach:

* Find all the words in string with the help of [re.findall()](https://www.geeksforgeeks.org/python-regex-re-search-vs-re-findall/) function.
* Iterate over the list in backward manner.
* Join the words in a string with the help of join() function.

**Python3**

# Function to reverse words of string

**import** re

**def** rev\_sentence(sentence):

    # find all the words in sentence

    words **=** re.findall('\w+', sentence)

    # Backward iterate over list of words and join using space

    reverse\_sentence **=** ' '.join(words[i] **for** i **in** range(len(words)**-**1, **-**1, **-**1))

    # finally return the joined string

**return** reverse\_sentence

**if** \_\_name\_\_ **==** "\_\_main\_\_":

    input **=** 'geeks quiz practice code'

    print (rev\_sentence(input))

**Output**

code practice quiz geeks

Given an input string, write a function that returns the [Run Length Encoded](https://en.wikipedia.org/wiki/Run-length_encoding) string for the input string. For example, if the input string is ‘wwwwaaadexxxxxx’, then the function should return ‘w4a3d1e1x6’.

**Examples:**

Input : str = 'wwwwaaadexxxxxx'  
Output : 'w4a3d1e1x6'

[Recommended: Please try your approach on ***{IDE}*** first, before moving on to the solution.](https://ide.geeksforgeeks.org/)

This problem has existing solution please refer [Run Length Encoding](https://www.geeksforgeeks.org/run-length-encoding/) link. Here we will solve this problem quickly in python using [OrderedDict](https://www.geeksforgeeks.org/remove-duplicates-given-string-python/). Approach is very simple, first we create a ordered dictionary which contains characters of input string as key and 0 as their default value, now we run a loop to count frequency of each character and will map it to it’s corresponding key.

**Implementation:**

**Python3**

# Python code for run length encoding

**from** collections **import** OrderedDict

**def** runLengthEncoding(input):

    # Generate ordered dictionary of all lower

    # case alphabets, its output will be

    # dict = {'w':0, 'a':0, 'd':0, 'e':0, 'x':0}

    dict**=**OrderedDict.fromkeys(input, 0)

    # Now iterate through input string to calculate

    # frequency of each character, its output will be

    # dict = {'w':4,'a':3,'d':1,'e':1,'x':6}

**for** ch **in** input:

        dict[ch] **+=** 1

    # now iterate through dictionary to make

    # output string from (key,value) pairs

    output **=** ''

**for** key,value **in** dict.items():

        output **=** output **+** key **+** str(value)

**return** output

# Driver function

**if** \_\_name\_\_ **==** "\_\_main\_\_":

    input**=**"wwwwaaadexxxxxx"

    print (runLengthEncoding(input))

**Output**

w4a3d1e1x6

**Another code:**

**Python3**

**def** encode(message):

    encoded\_message **=** ""

    i **=** 0

**while** (i <**=** len(message)**-**1):

        count **=** 1

        ch **=** message[i]

        j **=** i

**while** (j < len(message)**-**1):

**if** (message[j] **==** message[j**+**1]):

                count **=** count**+**1

                j **=** j**+**1

**else**:

**break**

        encoded\_message**=**encoded\_message**+**str(count)**+**ch

        i **=** j**+**1

**return** encoded\_message

#Provide different values for message and test your program

encoded\_message**=**encode("ABBBBCCCCCCCCAB")

print(encoded\_message)

**Output**

1A4B8C1A1B

**Anagram checking in Python using collections.Counter()**

Write a function to check whether two given strings are anagram of each other or not. An anagram of a string is another string that contains same characters, only the order of characters can be different. For example, “abcd” and “dabc” are anagram of each other.

**Examples:**

Input : str1 = “abcd”, str2 = “dabc”  
Output : True

Input : str1 = “abcf”, str2 = “kabc”  
Output : False

[Recommended: Please try your approach on ***{IDE}*** first, before moving on to the solution.](https://ide.geeksforgeeks.org/)

This problem has existing solution please refer [Check whether two strings are anagram of each other](https://www.geeksforgeeks.org/check-whether-two-strings-are-anagram-of-each-other/) link. We will solve this problem in python in a single line using [collections.Counter() module](https://www.geeksforgeeks.org/count-frequencies-elements-array-python-using-collections-module/).

**Implementation:**

**Python**

# Python code to check if two strings are

# anagram

**from** collections **import** Counter

**def** anagram(input1, input2):

    # Counter() returns a dictionary data

    # structure which contains characters

    # of input as key and their frequencies

    # as it's corresponding value

**return** Counter(input1) **==** Counter(input2)

# Driver function

**if** \_\_name\_\_ **==** "\_\_main\_\_":

    input1 **=** 'abcd'

    input2 **=** 'dcab'

    print anagram(input1, input2)

**Output**

True

**Time complexity: O(n)**

**Auxiliary Space: O(1)**

**How dictionary comparison works in python ?**

If we have two dictionary data structures in python dict1 = {‘a’:2,’b’:3,’c’:1} and dict2 = {‘b’:3,’c’:1,’a’:2} and we compare both of them like **dict1=dict2** then it will result **True**. In python ordinary dictionary data structure does not follow any ordering of keys, when we compare two dictionaries then it compares three checks in order **number of keys (if they don’t match, the dicts are not equal)**, **names of keys (if they don’t match, they’re not equal)** and **value of each key (they have to be ‘==’, too)**.

*From <*[*https://www.geeksforgeeks.org/anagram-checking-python-collections-counter/*](https://www.geeksforgeeks.org/anagram-checking-python-collections-counter/)*>*

**Remove all duplicates from a given string in Python**

We are given a string and we need to remove all duplicates from it? What will be the output if the order of character matters? **Examples:**

***Input :****geeksforgeeks*

***Output :****efgkors*

[Recommended: Please try your approach on ***{IDE}*** first, before moving on to the solution.](https://ide.geeksforgeeks.org/)

This problem has existing solution please refer [Remove all duplicates from a given string](https://www.geeksforgeeks.org/remove-all-duplicates-from-the-input-string/).

**Method 1:**

* Python3

**from** collections **import** OrderedDict

# Function to remove all duplicates from string

# and order does not matter

**def** removeDupWithoutOrder(str):

    # set() --> A Set is an unordered collection

    #         data type that is iterable, mutable,

    #         and has no duplicate elements.

    # "".join() --> It joins two adjacent elements in

    #             iterable with any symbol defined in

    #             "" ( double quotes ) and returns a

    #             single string

**return** "".join(set(str))

# Function to remove all duplicates from string

# and keep the order of characters same

**def** removeDupWithOrder(str):

**return** "".join(OrderedDict.fromkeys(str))

# Driver program

**if** \_\_name\_\_ **==** "\_\_main\_\_":

    str **=** "geeksforgeeks"

**print** ("Without Order = ",removeDupWithoutOrder(str))

**print** ("With Order = ",removeDupWithOrder(str))

**Output**

Without Order = foskerg  
With Order = geksfor

**Time complexity: O(n)**

**Auxiliary Space: O(n)**

**Method 2:**

* Python3

**def** removeDuplicate(str):

    s**=**set(str)

    s**=**"".join(s)

**print**("Without Order:",s)

    t**=**""

**for** i **in** str:

**if**(i **in** t):

**pass**

**else**:

            t**=**t**+**i

        print("With Order:",t)

str**=**"geeksforgeeks"

removeDuplicate(str)

**Output**

Without Order: kogerfs  
With Order: g  
With Order: ge  
With Order: ge  
With Order: gek  
With Order: geks  
With Order: geksf  
With Order: geksfo  
With Order: geksfor  
With Order: geksfor  
With Order: geksfor  
With Order: geksfor  
With Order: geksfor  
With Order: geksfor

**Time complexity:** **O(n)**

**Auxiliary Space: O(n)**

**What do OrderedDict and fromkeys() do ?**

An OrderedDict is a dictionary that remembers the order of the keys that were inserted first. If a new entry overwrites an existing entry, the original insertion position is left unchanged.

For example see below code snippet :

* Python3

**from** collections **import** OrderedDict

ordinary\_dictionary **=** {}

ordinary\_dictionary['a'] **=** 1

ordinary\_dictionary['b'] **=** 2

ordinary\_dictionary['c'] **=** 3

ordinary\_dictionary['d'] **=** 4

ordinary\_dictionary['e'] **=** 5

# Output = {'a': 1, 'c': 3, 'b': 2, 'e': 5, 'd': 4}

**print** (ordinary\_dictionary)

ordered\_dictionary **=** OrderedDict()

ordered\_dictionary['a'] **=** 1

ordered\_dictionary['b'] **=** 2

ordered\_dictionary['c'] **=** 3

ordered\_dictionary['d'] **=** 4

ordered\_dictionary['e'] **=** 5

# Output = {'a':1,'b':2,'c':3,'d':4,'e':5}

print (ordered\_dictionary)

**Output**

{'a': 1, 'b': 2, 'c': 3, 'd': 4, 'e': 5}  
OrderedDict([('a', 1), ('b', 2), ('c', 3), ('d', 4), ('e', 5)])

**Time complexity: O(n)**

**Auxiliary Space: O(1)**

**fromkeys()** creates a new dictionary with keys from seq and values set to value and returns list of keys, **fromkeys(seq[, value])** is the syntax for fromkeys() method. **Parameters :**

* **seq :** This is the list of values which would be used for dictionary keys preparation.
* **value :** This is optional, if provided then value would be set to this value.

For example see below code snippet :

* Python3

**from** collections **import** OrderedDict

seq **=** ('name', 'age', 'gender')

dict **=** OrderedDict.fromkeys(seq)

# Output = {'age': None, 'name': None, 'gender': None}

**print** (str(dict))

dict **=** OrderedDict.fromkeys(seq, 10)

# Output = {'age': 10, 'name': 10, 'gender': 10}

**print** (str(dict))

**Output**

OrderedDict([('name', None), ('age', None), ('gender', None)])  
OrderedDict([('name', 10), ('age', 10), ('gender', 10)])

**Time complexity: O(n)**

**Auxiliary Space: O(1)**

*From <*[*https://www.geeksforgeeks.org/remove-duplicates-given-string-python/*](https://www.geeksforgeeks.org/remove-duplicates-given-string-python/)*>*

**Remove all consecutive duplicates from the string**

Given a string **S**, The task is to remove all the consecutive duplicate characters of the string and return the resultant string.

**Note:**that this problem is different from [Recursively remove all adjacent duplicates](https://www.geeksforgeeks.org/recursively-remove-adjacent-duplicates-given-string/). Here we keep one character and remove all subsequent same characters.

**Examples:**

***Input:****S= “aaaaabbbbbb”*

***Output:****ab*

***Input:****S = “geeksforgeeks”*

***Output:****geksforgeks*

***Input:****S = “aabccba”*

***Output:****abcba*

*Recommended Practice*

[*Remove Consecutive Characters*](https://practice.geeksforgeeks.org/problems/consecutive-elements2306/1)

[*Try It!*](https://practice.geeksforgeeks.org/problems/consecutive-elements2306/1)

***Remove all consecutive duplicates from the string using***[*recursion*](http://www.geeksforgeeks.org/recursion/)***:***

***Approach:***

*If the string is not empty compare the adjacent characters of the string. If they are the same then shift the characters one by one to the left. Call recursion on string S otherwise, call recursion from S+1 string.*

*Follow the below steps to implement the idea:*

* *If the string is empty, return.*
* *Else compare the adjacent characters of the string. If they are the same then shift the characters one by one to the left. Call recursion on string S*
* *If they are not same then call recursion from S+1 string.*

***Illustration:***

*The recursion tree for the string****S****= aabcca is shown below.*

*aabcca    S = aabcca*

*/*

*abcca            S = abcca*

*/*

*bcca                 S = abcca*

*/*

*cca                        S = abcca*

*/*

*ca                                S = abca*

*/*

*a                                     S = abca (Output String)*

*/*

*empty string*

*Below is the implementation of the above approach:*

# Recursive Program to remove consecutive

# duplicates from string S.

**def** removeConsecutiveDuplicates(s):

**if** len(s) < 2:

**return** s

**if** s[0] !**=** s[1]:

**return** s[0]**+**removeConsecutiveDuplicates(s[1:])

**return** removeConsecutiveDuplicates(s[1:])

# This code is contributed by direwolf707

s1 **=** 'geeksforgeeks'

**print**(removeConsecutiveDuplicates(s1))  # geksforgeks

s2 **=** 'aabcca'

print(removeConsecutiveDuplicates(s2))  # ab

# This code is contributed by rahulsood707.

***Output***

*geksforgeks  
abca*

***Time complexity:****O(N2)*

***Auxiliary Space:****O(1)*

***Remove all consecutive duplicates from the string using Iterative approach:***

*The idea is to check if current character is equal to the next character or not . If current character is equal to the next character we’ll ignore it and when it is not equal we will add it to our answer. At last add the last character.*

*Follow the below steps to implement the idea:*

* *Create an****str****string to store the result*
* *Iterate from 0 to string length – 2.*
* *If the current character is not equal to next character then add it to answer string.*
* *Else continue.*
* *return****str****.*

*Below is the implementation of above approach:*

# Python3 program to remove consecutive

# duplicates from a given string.

# A iterative function that removes

# consecutive duplicates from string S

**def** removeDuplicates(S):

    n **=** len(S)

    # We don't need to do anything for

    # empty or single character string.

**if** (n < 2):

**return**

    # j is used to store index is result

    # string (or index of current distinct

    # character)

    j **=** 0

    # Traversing string

**for** i **in** range(n):

        # If current character S[i]

        # is different from S[j]

**if** (S[j] !**=** S[i]):

            j **+=** 1

            S[j] **=** S[i]

    # Putting string termination

    # character.

    j **+=** 1

    S **=** S[:j]

**return** S

# Driver Code

**if** \_\_name\_\_ **==** '\_\_main\_\_':

    S1 **=** "geeksforgeeks"

    S1 **=** list(S1.rstrip())

    S1 **=** removeDuplicates(S1)

    print(**\***S1, sep**=**"")

    S2 **=** "aabcca"

    S2 **=** list(S2.rstrip())

    S2 **=** removeDuplicates(S2)

    print(**\***S2, sep**=**"")

# This code is contributed by

# Shubham Singh(SHUBHAMSINGH10)

***Output***

*geksforgeks  
abca*

***Time Complexity:****O(N)*

***Auxiliary Space:****O(1)*

**Python program to check if a string is palindrome or not**

Given a string, write a python function to check if it is palindrome or not. A string is said to be palindrome if the reverse of the string is the same as string. For example, “radar” is a palindrome, but “radix” is not a palindrome.

**Examples:**

**Input :** malayalam  
**Output :** Yes

**Input :** geeks  
**Output :** No

[Recommended: Please try your approach on ***{IDE}*** first, before moving on to the solution.](https://ide.geeksforgeeks.org/)

**Method #1**

1. Find reverse of string
2. Check if reverse and original are same or not.

**Python**

# function which return reverse of a string

**def** isPalindrome(s):

**return** s **==** s[::**-**1]

# Driver code

s **=** "malayalam"

ans **=** isPalindrome(s)

**if** ans:

    print("Yes")

**else**:

    print("No")

**Output :**

Yes

***Time complexity:****O(n)*

***Auxiliary Space:****O(1)*

**Iterative Method:** This method is contributed by ***Shariq Raza***. Run a loop from starting to length/2 and check the first character to the last character of the string and second to second last one and so on …. If any character mismatches, the string wouldn’t be a palindrome.

Below is the implementation of above approach:

**Python**

# function to check string is

# palindrome or not

**def** isPalindrome(str):

    # Run loop from 0 to len/2

**for** i **in** range(0, int(len(str)**/**2)):

**if** str[i] !**=** str[len(str)**-**i**-**1]:

**return** False

**return** True

# main function

s **=** "malayalam"

ans **=** isPalindrome(s)

**if** (ans):

    print("Yes")

**else**:

**print**("No")

**Output:**

Yes

***Time complexity:****O(n)*

***Auxiliary Space:****O(1)*

**Method using the** **inbuilt function to reverse a string:**

In this method, predefined function **‘ ‘.join(reversed(string))**is used to reverse string.

Below is the implementation of the above approach:

**Python**

# function to check string is

# palindrome or not

**def** isPalindrome(s):

    # Using predefined function to

    # reverse to string print(s)

    rev **=** ''.join(reversed(s))

    # Checking if both string are

    # equal or not

**if** (s **==** rev):

**return** True

**return** False

# main function

s **=** "malayalam"

ans **=** isPalindrome(s)

**if** (ans):

    print("Yes")

**else**:

**print**("No")

**Output:**

Yes

**Time complexity:**O(n)

**Auxiliary Space:**O(n)

**Method using one extra variable:** In this method, the user takes a character of string one by one and store it in an empty variable. After storing all the characters user will compare both the string and check whether it is palindrome or not.

**Python**

# Python program to check

# if a string is palindrome

# or not

x **=** "malayalam"

w **=** ""

**for** i **in** x:

    w **=** i **+** w

**if** (x **==** w):

    print("Yes")

**else**:

    print("No")

**Output:**

Yes

***Time complexity:****O(n)*

***Auxiliary Space:****O(n)*

**Method using flag:** In this method, the user compares each character from starting and ending in a for loop and if the character does not match then it will change the status of the flag. Then it will check the status of the flag and accordingly and print whether it is a palindrome or not.

**Python**

# Python program to check

# if a string is palindrome

# or not

st **=** 'malayalam'

j **= -**1

flag **=** 0

**for** i **in** st:

**if** i !**=** st[j]:

        flag **=** 1

**break**

    j **=** j **-** 1

**if** flag **==** 1:

    print("NO")

**else**:

    print("Yes")

**Output:**

Yes

***Time complexity:****O(n)*

***Auxiliary Space:****O(1)*

**Method using recursion**:

This method compares the first and the last element of the string and gives the rest of the substring to a recursive call to itself.

**Python3**

# Recursive function to check if a

# string is palindrome

**def** isPalindrome(s):

    # to change it the string is similar case

    s **=** s.lower()

    # length of s

    l **=** len(s)

    # if length is less than 2

**if** l < 2:

**return** True

    # If s[0] and s[l-1] are equal

**elif** s[0] **==** s[l **-** 1]:

        # Call is palindrome form substring(1,l-1)

**return** isPalindrome(s[1: l **-** 1])

**else**:

**return** False

# Driver Code

s **=** "MalaYaLam"

ans **=** isPalindrome(s)

**if** ans:

    print("Yes")

**else**:

**print**("No")

**Output:**

Yes

***Time complexity:****O(n)*

***Auxiliary Space:****O(n)*

**Method : Using extend() and reverse() methods**

**Python3**

#Using lists and reverse() method

**def** isPalindrome(s):

    x**=**list(s)

    y**=**[]

    y.extend(x)

    x.reverse()

**if**(x**==**y):

**return** True

**return** False

# Driver Code

s **=** "malayalam"

ans **=** isPalindrome(s)

**if** ans:

**print**("Yes")

**else**:

**print**("No")

**Output**

Yes

-------------------------------------------------------------------------------------------------------------------------------------------------------------

**Arthimetic Operation in String :**

1. [Smallest number with sum of digits as N and divisible by 10^N](https://www.geeksforgeeks.org/smallest-number-sum-digits-n-divisible-10n/)
2. [Minimum sum of squares of character counts in a given string after removing k characters](https://www.geeksforgeeks.org/minimum-sum-squares-characters-counts-given-string-removing-k-characters/)
3. [Maximum and minimum sums from two numbers with digit replacements](https://www.geeksforgeeks.org/maximum-minimum-sums-two-numbers-digit-replacements/)
4. [Check if a given string is sum-string](https://www.geeksforgeeks.org/check-given-string-sum-string/)
5. [Sum of two large numbers](https://www.geeksforgeeks.org/sum-two-large-numbers/)
6. [Calculate sum of all numbers present in a string](https://www.geeksforgeeks.org/calculate-sum-of-all-numbers-present-in-a-string/)
7. [Extract maximum numeric value from a given string](https://www.geeksforgeeks.org/extract-maximum-numeric-value-given-string-set-2-regex-approach/)
8. [Calculate maximum value using ‘+’ or ‘\*’ sign between two numbers in a string](https://www.geeksforgeeks.org/calculate-maximum-value-using-sign-two-numbers-string/)
9. [Maximum segment value after putting k breakpoints in a number](https://www.geeksforgeeks.org/maximum-segment-value-putting-k-breakpoints-number/)
10. [Difference of two large numbers](https://www.geeksforgeeks.org/difference-of-two-large-numbers/)
11. [Check if a large number is divisible by 4 or not](https://www.geeksforgeeks.org/check-large-number-divisible-4-not/)
12. [Check if a large number is divisible by 11 or not](https://www.geeksforgeeks.org/check-large-number-divisible-11-not/)
13. [Number of substrings divisible by 6 in a string of integers](https://www.geeksforgeeks.org/number-substrings-divisible-6-string-integers/)
14. [Decimal representation of given binary string is divisible by 5 or not](https://www.geeksforgeeks.org/decimal-representation-given-binary-string-divisible-5-not/)
15. [Number of substrings divisible by 8 but not by 3](https://www.geeksforgeeks.org/number-substrings-divisible-8-not-3/)
16. [To check divisibility of any large number by 999](https://www.geeksforgeeks.org/check-divisibility-large-number-999/)
17. [Multiply Large Numbers represented as Strings](https://www.geeksforgeeks.org/multiply-large-numbers-represented-as-strings/)
18. [Divide large number represented as string](https://www.geeksforgeeks.org/divide-large-number-represented-string/)
19. [Remainder with 7 for large numbers](https://www.geeksforgeeks.org/remainder-7-large-numbers/)
20. [Given two numbers as strings, find if one is a power of other](https://www.geeksforgeeks.org/given-two-numbers-strings-find-one-power/)
21. [Check whether a given number is even or odd](https://www.geeksforgeeks.org/check-whether-given-number-even-odd/)
22. [Product of nodes at k-th level in a tree represented as string](https://www.geeksforgeeks.org/product-nodes-k-th-level-tree-represented-string/)
23. [Program to find remainder when large number is divided by 11](https://www.geeksforgeeks.org/program-find-remainder-large-number-divided-11/)
24. [Ways to remove one element from a binary string so that XOR becomes zero](https://www.geeksforgeeks.org/ways-remove-one-element-binary-string-xor-becomes-zero/)
25. [Find the maximum subarray XOR in a given array](https://www.geeksforgeeks.org/find-the-maximum-subarray-xor-in-a-given-array/)
26. [Calculate the difficulty of a sentence](https://www.geeksforgeeks.org/calculate-difficulty-sentence/)
27. [Minimum Index Sum for Common Elements of Two Lists](https://www.geeksforgeeks.org/minimum-index-sum-common-elements-two-lists/)

**Smallest number with sum of digits as N and divisible by 10^N**

Find the smallest number such that the sum of its digits is N and it is divisible by

.

**Examples :**

Input : N = 5  
Output : 500000  
500000 is the smallest number divisible  
by 10^5 and sum of digits as 5.

Input : N = 20  
Output : 29900000000000000000000

Recommended Practice

[Smallest number with sum of digits as N and divisible by 10^N](https://practice.geeksforgeeks.org/problems/smallest-number-with-sum-of-digits-as-n-and-divisible-by-10n4032/1/)

[Try It!](https://practice.geeksforgeeks.org/problems/smallest-number-with-sum-of-digits-as-n-and-divisible-by-10n4032/1/)

**Explanation:**To make a number divisible by

we need at least N zeros at the end of the number. To make the number smallest, we append exactly N zeros to the end of the number. Now, we need to ensure the sum of the digits is N. For this, we will try to make the length of the number as small as possible to get the answer. Thus we keep on inserting 9 into the number till the sum doesn’t exceed N. If we have any remainder left, then we keep it as the first digit (a most significant one) so that the resulting number is minimized.

The approach works well for all subtasks but there are 2 corner cases:

1. The first is that the final number may not fit into the data types present in C++/Java. Since we only need to output the number, we can use strings to store the answer.
2. The only corner case where the answer is 0 is N = 0.
3. There are no cases where the answer doesn’t exist.

**Implementation:**

# Python program to find smallest

# number to find smallest number

# with N as sum of digits and

# divisible by 10^N.

**import** math

**def** digitsNum(N):

    # If N = 0 the string will be 0

**if** (N **==** 0) :

**print**("0", end **=** "")

    # If n is not perfectly divisible

    # by 9 output the remainder

**if** (N **%** 9 !**=** 0):

        print (N **%** 9, end **=**"")

    # Print 9 N/9 times

**for** i **in** range( 1, int(N **/** 9) **+** 1) :

**print**("9", end **=** "")

    # Append N zero's to the number so

    # as to make it divisible by 10^N

**for** i **in** range(1, N **+** 1) :

**print**("0", end **=** "")

**print**()

# Driver Code

N **=** 5

print("The number is : ",end**=**"")

digitsNum(N)

# This code is contributed by Gitanjali.

**Output**

The number is : 500000

**Time Complexity:** O(N), where N is the given input.

**Auxiliary Space:**O(1), no extra space is required, so it is a constant.

*From <*[*https://www.geeksforgeeks.org/smallest-number-sum-digits-n-divisible-10n/*](https://www.geeksforgeeks.org/smallest-number-sum-digits-n-divisible-10n/)*>*

**Minimum sum of squares of character counts in a given string after removing k characters**

Given a string of lowercase alphabets and a number k, the task is to print the minimum value of the string after removal of ‘k’ characters. The value of a string is defined as the sum of squares of the count of each distinct character.

For example consider the string “saideep”, here frequencies of characters are s-1, a-1, i-1, e-2, d-1, p-1 and value of the string is 1^2 + 1^2 + 1^2 + 1^2 + 1^2 + 2^2 = 9.

Expected Time Complexity: O(k\*logn)

**Examples:**

***Input :****str = abccc, K = 1*

***Output :****6*

***Explanation:****We remove c to get the value as 12 + 12 + 22*

***Input :****str = aaab, K = 2*

***Output :****2*

**Asked In: Amazon**

Recommended Problem

Game with String

One clear observation is that we need to remove character with highest frequency. One trick is the character ma

A **Simple solution** is to use sorting technique through all current highest frequency reduce up to k times. For After every reduce again sort frequency array.

A **Better Solution** used to Priority Queue which has to the highest element on top.

1. Initialize empty priority queue.
2. Count frequency of each character and Store into temp array.
3. Remove K characters which have highest frequency from queue.
4. Finally Count Sum of square of each element and return it.

Below is the implementation of the above idea.

# Python3 program to find min sum of

# squares of characters after k removals

**from** queue **import** PriorityQueue

MAX\_CHAR **=** 26

# Main Function to calculate min sum of

# squares of characters after k removals

**def** minStringValue(str, k):

    l **=** len(str) # find length of string

    # if K is greater than length of string

    # so reduced string will become 0

**if**(k >**=** l):

**return** 0

    # Else find Frequency of each

    # character and store in an array

    frequency **=** [0] **\*** MAX\_CHAR

**for** i **in** range(0, l):

        frequency[ord(str[i]) **-** 97] **+=** 1

    # Push each char frequency negative

    # into a priority\_queue as the queue

    # by default is minheap

    q **=** PriorityQueue()

**for** i **in** range(0, MAX\_CHAR):

        q.put(**-**frequency[i])

    # Removal of K characters

**while**(k > 0):

        # Get top element in priority\_queue

        # multiply it by -1 as temp is negative

        # remove it. Increment by 1 and again

        # push into priority\_queue

        temp **=** q.get()

        temp **=** temp **+** 1

        q.put(temp, temp)

        k **=** k **-** 1

    # After removal of K characters find

    # sum of squares of string Value

    result **=** 0; # initialize result

**while not** q.empty():

        temp **=** q.get()

        temp **=** temp **\*** (**-**1)

        result **+=** temp **\*** temp

**return** result

# Driver Code

**if** \_\_name\_\_ **==** "\_\_main\_\_":

    str **=** "abbccc"

    k **=** 2

**print**(minStringValue(str, k))

    str **=** "aaab"

    k **=** 2

    print(minStringValue(str, k))

# This code is contributed

# by Sairahul Jella

**Output**

6  
2

**Time Complexity:** **O(k\*logn)**

**Auxiliary Space: O(1)**because constant size array and priority\_queue are being used

----------------------------------------------------------------------------------------------------------------------------------------------------------------

**Character Counting Based Problems :**

1. [Count Uppercase, Lowercase, special character and numeric values](https://www.geeksforgeeks.org/count-uppercase-lowercase-special-character-numeric-values/)
2. [Find the smallest window in a string containing all characters of another string](https://www.geeksforgeeks.org/find-the-smallest-window-in-a-string-containing-all-characters-of-another-string/)
3. [Smallest window that contains all characters of string itself](https://www.geeksforgeeks.org/smallest-window-contains-characters-string/)
4. [Count number of substrings with exactly k distinct characters](https://www.geeksforgeeks.org/count-number-of-substrings-with-exactly-k-distinct-characters/)
5. [Number of substrings with count of each character as k](https://www.geeksforgeeks.org/number-substrings-count-character-k/)
6. [String with k distinct characters and no same characters adjacent](https://www.geeksforgeeks.org/string-k-distinct-characters-no-characters-adjacent/)
7. [Number of substrings of a string](https://www.geeksforgeeks.org/number-substrings-string/)
8. [Distinct strings with odd and even changes allowed](https://www.geeksforgeeks.org/distinct-strings-odd-even-changes-allowed/)
9. [Find k’th character of decrypted string](https://www.geeksforgeeks.org/find-kth-character-of-decrypted-string/)
10. [Count characters at same position as in English alphabets](https://www.geeksforgeeks.org/count-characters-position-english-alphabets/)
11. [Count words in a given string](https://www.geeksforgeeks.org/count-words-in-a-given-string/)
12. [Count words present in a string](https://www.geeksforgeeks.org/count-words-present-in-a-string/)
13. [Count of words whose i-th letter is either (i-1)-th, i-th, or (i+1)-th letter of given word](https://www.geeksforgeeks.org/count-words-whose-th-letter-either-1-th-th-i1-th-letter-given-word/)
14. [Program to find Smallest and Largest Word in a String](https://www.geeksforgeeks.org/program-find-smallest-largest-word-string/)
15. [Count substrings with same first and last characters](https://www.geeksforgeeks.org/count-substrings-with-same-first-and-last-characters/)
16. [Recursive solution to count substrings with same first and last characters](https://www.geeksforgeeks.org/recursive-solution-count-substrings-first-last-characters/)
17. [Count of distinct substrings of a string using Suffix Array](https://www.geeksforgeeks.org/count-distinct-substrings-string-using-suffix-array/)
18. [Count of distinct substrings of a string using Suffix Trie](https://www.geeksforgeeks.org/count-distinct-substrings-string-using-suffix-trie/)
19. [Count number of strings (made of R, G and B) using given combination](https://www.geeksforgeeks.org/count-number-of-strings-made-of-r-g-and-b-using-given-combination/)
20. [Count of strings that can be formed using a, b and c under given constraints](https://www.geeksforgeeks.org/count-strings-can-formed-using-b-c-given-constraints/)
21. [Count of substrings of a binary string containing K ones](https://www.geeksforgeeks.org/count-substrings-binary-string-containing-k-ones/)
22. [Group words with same set of characters](https://www.geeksforgeeks.org/print-words-together-set-characters/)
23. [Print all distinct characters of a string in order (3 Methods)](https://www.geeksforgeeks.org/print-all-distinct-characters-of-a-string-in-order-3-methods/)
24. [Print common characters of two Strings in alphabetical order](https://www.geeksforgeeks.org/print-common-characters-two-strings-alphabetical-order-2/)
25. [Common characters in n strings](https://www.geeksforgeeks.org/common-characters-n-strings/)
26. [Find uncommon characters of the two strings](https://www.geeksforgeeks.org/find-uncommon-characters-two-strings/)
27. [Concatenated string with uncommon characters of two strings](https://www.geeksforgeeks.org/concatenated-string-uncommon-characters-two-strings/)
28. [Program to remove vowels from a String](https://www.geeksforgeeks.org/program-remove-vowels-string/)
29. [Remove consecutive vowels from string](https://www.geeksforgeeks.org/remove-consecutive-vowels-string/)
30. [Program to count vowels in a string (Iterative and Recursive)](https://www.geeksforgeeks.org/program-count-vowels-string-iterative-recursive/)
31. [Count consonants in a string (Iterative and recursive methods)](https://www.geeksforgeeks.org/count-consonants-string-iterative-recursive-methods/)
32. [Alternate vowel and consonant string](https://www.geeksforgeeks.org/alternate-vowel-consonant-string/)
33. [Given a binary string, count number of substrings that start and end with 1](https://www.geeksforgeeks.org/given-binary-string-count-number-substrings-start-end-1/)
34. [Number of distinct permutation a String can have](https://www.geeksforgeeks.org/number-distinct-permutation-string-can/)
35. [Time complexity of all permutations of a string](https://www.geeksforgeeks.org/time-complexity-permutations-string/)
36. [Permutations of a given string using STL](https://www.geeksforgeeks.org/permutations-of-a-given-string-using-stl/)
37. [Check if both halves of the string have same set of characters](https://www.geeksforgeeks.org/check-half-string-character-frequency-character/)
38. [Count words that appear exactly two times in an array of words](https://www.geeksforgeeks.org/count-words-appear-exactly-two-times-array-words/)
39. [Check if frequency of all characters can become same by one removal](https://www.geeksforgeeks.org/check-if-frequency-of-all-characters-can-become-same-by-one-removal/)
40. [Check if a string has all characters with same frequency with one variation allowed](https://www.geeksforgeeks.org/check-if-a-string-has-all-characters-with-same-frequency-with-one-variation-allowed/)
41. [Count ways to increase LCS length of two strings by one](https://www.geeksforgeeks.org/count-ways-increase-lcs-length-two-strings-one/)
42. [Find the character in first string that is present at minimum index in second string](https://www.geeksforgeeks.org/find-character-first-string-present-minimum-index-second-string/)
43. [Remove characters from the first string which are present in the second string](https://www.geeksforgeeks.org/remove-characters-from-the-first-string-which-are-present-in-the-second-string/)
44. [Length of Longest sub-string that can be removed](https://www.geeksforgeeks.org/length-longest-sub-string-can-make-removed/)
45. [Count of character pairs at same distance as in English alphabets](https://www.geeksforgeeks.org/count-characters-string-distance-english-alphabets/)
46. [Count number of equal pairs in a string](https://www.geeksforgeeks.org/number-equal-pairs-string/)
47. [Count of strings where adjacent characters are of difference one](https://www.geeksforgeeks.org/count-strings-adjacent-characters-difference-one/)
48. [Print number of words, vowels and frequency of each character](https://www.geeksforgeeks.org/print-number-words-vowels-frequency-character/)

**Subsequence & Substring :**

1. [Longest subsequence where every character appears at-least k times](https://www.geeksforgeeks.org/longest-subsequence-where-every-character-appears-at-least-k-times/)
2. [Given two strings, find if first string is a subsequence of second](https://www.geeksforgeeks.org/given-two-strings-find-first-string-subsequence-second/)
3. [Number of subsequences of the form a^i b^j c^k](https://www.geeksforgeeks.org/number-subsequences-form-ai-bj-ck/)
4. [Number of subsequences in a string divisible by n](https://www.geeksforgeeks.org/number-subsequences-string-divisible-n/)
5. [Find number of times a string occurs as a subsequence in given string](https://www.geeksforgeeks.org/find-number-times-string-occurs-given-string/)
6. [Number of subsequences as “ab” in a string repeated K times](https://www.geeksforgeeks.org/number-subsequences-ab-string-repeated-k-times/)
7. [Count of ‘GFG’ Subsequences in the given string](https://www.geeksforgeeks.org/count-gfg-subsequences-given-string/)
8. [Count Distinct Subsequences](https://www.geeksforgeeks.org/count-distinct-subsequences/)
9. [Count distinct occurrences as a subsequence](https://www.geeksforgeeks.org/count-distinct-occurrences-as-a-subsequence/)
10. [Longest common subsequence with permutations allowed](https://www.geeksforgeeks.org/longest-common-subsequence-with-permutations-allowed/)
11. [Repeated subsequence of length 2 or more](https://www.geeksforgeeks.org/repeated-subsequence-length-2/)
12. [Print all longest common sub-sequences in lexicographical order](https://www.geeksforgeeks.org/print-longest-common-sub-sequences-lexicographical-order/)
13. [Printing Longest Common Subsequence | Set 2](https://www.geeksforgeeks.org/printing-longest-common-subsequence-set-2-printing/)
14. [Given number as string, find number of contiguous subsequences which recursively add up to 9 | Set 2](https://www.geeksforgeeks.org/given-number-string-find-number-contiguous-subsequences-recursively-add-9-set-2/)
15. [Shortest Uncommon Subsequence](https://www.geeksforgeeks.org/shortest-uncommon-subsequence/)
16. [Shortest Superstring Problem](https://www.geeksforgeeks.org/shortest-superstring-problem/)
17. [Printing Shortest Common Supersequence](https://www.geeksforgeeks.org/print-shortest-common-supersequence/)
18. [Shortest possible combination of two strings](https://www.geeksforgeeks.org/shortest-possible-combination-two-strings/)
19. [A Space Optimized Solution of LCS](https://www.geeksforgeeks.org/space-optimized-solution-lcs/)
20. [Sort a string according to the order defined by another string](https://www.geeksforgeeks.org/sort-string-according-order-defined-another-string/)
21. [Shortest Common Supersequence](https://www.geeksforgeeks.org/shortest-common-supersequence/)
22. [Longest Repeating Subsequence](https://www.geeksforgeeks.org/longest-repeating-subsequence/)
23. [Find largest word in dictionary by deleting some characters of given string](https://www.geeksforgeeks.org/find-largest-word-dictionary-deleting-characters-given-string/)
24. [Dynamic Programming | Set 12 (Longest Palindromic Subsequence)](https://www.geeksforgeeks.org/dynamic-programming-set-4-longest-common-subsequence/)

[More >>](https://www.geeksforgeeks.org/data-structures/string-subsequence-substring/)

**Reverse & Rotation :**

1. [Perfect reversible string](https://www.geeksforgeeks.org/perfect-reversible-string/)
2. [Reversing an Equation](https://www.geeksforgeeks.org/reversing-an-equation/)
3. [Left Rotation and Right Rotation of a String](https://www.geeksforgeeks.org/left-rotation-right-rotation-string-2/)
4. [Generate all rotations of a given string](https://www.geeksforgeeks.org/generate-rotations-given-string/)
5. [Minimum rotations required to get the same string](https://www.geeksforgeeks.org/minimum-rotations-required-get-string/)
6. [Check if strings are rotations of each other or not](https://www.geeksforgeeks.org/check-strings-rotations-not-set-2/)
7. [Check if a string can be obtained by rotating another string 2 places](https://www.geeksforgeeks.org/check-string-can-obtained-rotating-another-string-2-places/)
8. [Count rotations divisible by 4](https://www.geeksforgeeks.org/count-rotations-divisible-4/)
9. [Check if all rows of a matrix are circular rotations of each other](https://www.geeksforgeeks.org/check-rows-matrix-circular-rotations/)
10. [Print reverse of a string using recursion](https://www.geeksforgeeks.org/reverse-a-string-using-recursion/)
11. [Print words of a string in reverse order](https://www.geeksforgeeks.org/print-words-string-reverse-order/)
12. [Program to reverse a string (Iterative and Recursive)](https://www.geeksforgeeks.org/program-reverse-string-iterative-recursive/)
13. [Write a program to reverse an array or string](https://www.geeksforgeeks.org/write-a-program-to-reverse-an-array-or-string/)
14. [Reverse an array without affecting special characters](https://www.geeksforgeeks.org/reverse-an-array-without-affecting-special-characters/)
15. [Reverse words in a given string](https://www.geeksforgeeks.org/reverse-words-in-a-given-string/)
16. [Reverse individual words](https://www.geeksforgeeks.org/reverse-individual-words/)
17. [Reverse a string preserving space positions](https://www.geeksforgeeks.org/reverse-string-preserving-space-positions/)
18. [Reverse string without using any temporary variable](https://www.geeksforgeeks.org/reverse-string-without-using-any-temporary-variable/)
19. [Print reverse string after removing vowels](https://www.geeksforgeeks.org/print-reverse-string-removing-vowels/)
20. [Reverse vowels in a given string](https://www.geeksforgeeks.org/reverse-vowels-given-string/)
21. [Reverse String according to the number of words](https://www.geeksforgeeks.org/reverse-string-according-number-words/)
22. [Reverse each word in a linked list node](https://www.geeksforgeeks.org/reverse-word-linked-list-node/)
23. [Find if an array of strings can be chained to form a circle](https://www.geeksforgeeks.org/find-array-strings-can-chained-form-circle-set-2/)

**Sorting & Searching :**

1. [Sort an array of strings according to string lengths](https://www.geeksforgeeks.org/sort-array-strings-according-string-lengths/)
2. [Sorting array of strings (or words) using Trie](https://www.geeksforgeeks.org/sorting-array-strings-words-using-trie/)
3. [Sort string of characters](https://www.geeksforgeeks.org/sort-string-characters/)
4. [Alternate Lower Upper String Sort](https://www.geeksforgeeks.org/alternate-lower-upper-string-sort/)
5. [Program to sort string in descending order](https://www.geeksforgeeks.org/program-sort-string-descending-order/)
6. [Print array of strings in sorted order without copying one string into another](https://www.geeksforgeeks.org/print-array-strings-sorted-order-without-copying-one-string-another/)
7. [Sort the given string using character search](https://www.geeksforgeeks.org/sort-given-string-using-character-search/)
8. [Given a sorted dictionary of an alien language, find order of characters](https://www.geeksforgeeks.org/given-sorted-dictionary-find-precedence-characters/)
9. [Rearrange a string in sorted order followed by the integer sum](https://www.geeksforgeeks.org/rearrange-a-string-in-sorted-order-followed-by-the-integer-sum/)
10. [Print distinct sorted permutations with duplicates allowed in input](https://www.geeksforgeeks.org/print-distinct-sorted-permutations-with-duplicates-allowed/)
11. [Minimum cost to sort strings using reversal operations of different costs](https://www.geeksforgeeks.org/minimum-cost-sort-strings-using-reversal-operations-different-costs/)
12. [Print number in ascending order which contains 1, 2 and 3 in their digits.](https://www.geeksforgeeks.org/print-number-ascending-order-contains-1-2-3-digits/)
13. [Search in an array of strings where non-empty strings are sorted](https://www.geeksforgeeks.org/search-in-an-array-of-strings-where-non-empty-strings-are-sorted/)
14. [Sparse Search](https://www.geeksforgeeks.org/sparse-search/)

**Case Sensitive String :**

1. [Lower case to upper case – An interesting fact](https://www.geeksforgeeks.org/lower-case-upper-case-interesting-fact/)
2. [isupper() and islower() and their application in C++](https://www.geeksforgeeks.org/isupper-islower-application-c/)
3. [Case conversion (Lower to Upper and Vice Versa) of a string using BitWise operators in C/C++](https://www.geeksforgeeks.org/case-conversion-lower-upper-vice-versa-string-using-bitwise-operators-cc/)
4. [Maximum distinct lowercase alphabets between two uppercase](https://www.geeksforgeeks.org/maximum-distinct-lowercase-alphabets-two-uppercase/)
5. [First uppercase letter in a string (Iterative and Recursive)](https://www.geeksforgeeks.org/first-uppercase-letter-in-a-string-iterative-and-recursive/)
6. [Convert characters of a string to opposite case](https://www.geeksforgeeks.org/convert-alternate-characters-string-upper-case/)
7. [gOOGLE cASE of a given sentence](https://www.geeksforgeeks.org/google-case-given-sentence/)
8. [Print all words matching a pattern in CamelCase Notation Dictonary](https://www.geeksforgeeks.org/print-words-matching-pattern-camelcase-notation-dictonary/)
9. [Camel case of a given sentence](https://www.geeksforgeeks.org/camel-case-given-sentence/)
10. [Permute a string by changing case](https://www.geeksforgeeks.org/permute-string-changing-case/)
11. [Toggle case of a string using Bitwise Operators](https://www.geeksforgeeks.org/toggle-case-string-using-bitwise-operators/)
12. [How to design a tiny URL or URL shortener?](https://www.geeksforgeeks.org/how-to-design-a-tiny-url-or-url-shortener/)

**Occurrence Based String :**

1. [Given a string, find its first non-repeating character](https://www.geeksforgeeks.org/given-a-string-find-its-first-non-repeating-character/)
2. [Print all permutations with repetition of characters](https://www.geeksforgeeks.org/print-all-permutations-with-repetition-of-characters/)
3. [Find the first non-repeating character from a stream of characters](https://www.geeksforgeeks.org/find-first-non-repeating-character-stream-characters/)
4. [Convert to a string that is repetition of a substring of k length](https://www.geeksforgeeks.org/convert-string-repetition-substring-k-length/)
5. [Smallest length string with repeated replacement of two distinct adjacent](https://www.geeksforgeeks.org/smallest-length-string-with-repeated-replacement-of-two-distinct-adjacent/)
6. [Distributing all balls without repetition](https://www.geeksforgeeks.org/distributing-all-balls-without-repetition/)
7. [Maximum consecutive repeating character in string](https://www.geeksforgeeks.org/maximum-consecutive-repeating-character-string/)
8. [Minimum number of deletions so that no two consecutive are same](https://www.geeksforgeeks.org/queue-based-approach-for-first-non-repeating-character-in-a-stream/)
9. [K’th Non-repeating Character](https://www.geeksforgeeks.org/kth-non-repeating-character/)
10. [Find repeated character present first in a string](https://www.geeksforgeeks.org/find-repeated-character-present-first-string/)
11. [Find the first repeated word in a string](https://www.geeksforgeeks.org/find-first-repeated-word-string/)
12. [Find the first repeated character in a string](https://www.geeksforgeeks.org/find-the-first-repeated-character-in-a-string/)
13. [Second most repeated word in a sequence](https://www.geeksforgeeks.org/second-repeated-word-sequence/)
14. [Most frequent word in an array of strings](https://www.geeksforgeeks.org/frequent-word-array-strings/)
15. [Efficiently find first repeated character in a string without using any additional data structure in one traversal](https://www.geeksforgeeks.org/efficiently-find-first-repeated-character-string-without-using-additional-data-structure-one-traversal/)
16. [Queries for characters in a repeated string](https://www.geeksforgeeks.org/queries-for-characters-in-a-repeated-string/)
17. [Return maximum occurring character in an input string](https://www.geeksforgeeks.org/return-maximum-occurring-character-in-the-input-string/)
18. [Generate two output strings depending upon occurrence of character in input string.](https://www.geeksforgeeks.org/generate-two-output-strings-depending-upon-occurrence-character-input-string/)
19. [Print characters and their frequencies in order of occurrence](https://www.geeksforgeeks.org/print-characters-frequencies-order-occurrence/)
20. [Program to count occurrence of a given character in a string](https://www.geeksforgeeks.org/program-count-occurrence-given-character-string/)
21. [Check if all occurrences of a character appear together](https://www.geeksforgeeks.org/check-occurrences-character-appear-together/)
22. [Group all occurrences of characters according to first appearance](https://www.geeksforgeeks.org/group-occurrences-characters-according-first-appearance/)
23. [Print the string by ignoring alternate occurrences of any character](https://www.geeksforgeeks.org/print-string-ignoring-alternate-occurrences-character/)
24. [Print the string after the specified character has occurred given no. of times](https://www.geeksforgeeks.org/print-string-specified-character-occurred-given-no-times/)
25. [Find all occurrences of a given word in a matrix](https://www.geeksforgeeks.org/find-all-occurrences-of-the-word-in-a-matrix/)
26. [Replace all occurrences of string AB with C without using extra space](https://www.geeksforgeeks.org/replace-occurrences-string-ab-c-without-using-extra-space/)
27. [Rearrange a binary string as alternate x and y occurrences](https://www.geeksforgeeks.org/rearrange-binary-string-alternate-x-y-occurrences/)
28. [Remove recurring digits in a given number](https://www.geeksforgeeks.org/remove-recurring-digits-in-a-given-number/)
29. [Find the most frequent digit without using array/string](https://www.geeksforgeeks.org/find-the-most-frequent-digit-without-using-arraystring/)

**Spacing :**

1. [Remove spaces from a given string](https://www.geeksforgeeks.org/remove-spaces-from-a-given-string/)
2. [Move spaces to front of string in single traversal](https://www.geeksforgeeks.org/move-spaces-front-string-single-traversal/)
3. [Put spaces between words starting with capital letters](https://www.geeksforgeeks.org/put-spaces-words-starting-capital-letters/)
4. [Removing spaces from a string using Stringstream](https://www.geeksforgeeks.org/removing-spaces-string-using-stringstream/)
5. [Remove extra spaces from a string](https://www.geeksforgeeks.org/remove-extra-spaces-string/)
6. [URLify a given string (Replace spaces is %20)](https://www.geeksforgeeks.org/urlify-given-string-replace-spaces/)
7. [String containing first letter of every word in a given string with spaces](https://www.geeksforgeeks.org/string-containing-first-letter-every-word-given-string-spaces/)
8. [Print all possible strings that can be made by placing spaces](https://www.geeksforgeeks.org/print-possible-strings-can-made-placing-spaces-2/)
9. [Print all possible strings that can be made by placing spaces](https://www.geeksforgeeks.org/print-possible-strings-can-made-placing-spaces/)

**Anagram :**

1. [Check whether two strings are anagram of each other](https://www.geeksforgeeks.org/check-whether-two-strings-are-anagram-of-each-other/)
2. [Given a sequence of words, print all anagrams together | Set 2](https://www.geeksforgeeks.org/given-a-sequence-of-words-print-all-anagrams-together-set-2/)
3. [Anagram Substring Search](https://www.geeksforgeeks.org/anagram-substring-search-search-permutations/)
4. [Print all pairs of anagrams in a given array of strings](https://www.geeksforgeeks.org/print-pairs-anagrams-given-array-strings/)
5. [Remove minimum number of characters so that two strings become anagram](https://www.geeksforgeeks.org/remove-minimum-number-characters-two-strings-become-anagram/)
6. [Check if two strings are k-anagrams or not](https://www.geeksforgeeks.org/check-two-strings-k-anagrams-not/)
7. [Check if binary representations of two numbers are anagram](https://www.geeksforgeeks.org/check-binary-representations-two-numbers-anagram/)
8. [Given a sequence of words, print all anagrams together using STL](https://www.geeksforgeeks.org/given-a-sequence-of-words-print-all-anagrams-together-using-stl/)
9. [Check if all levels of two trees are anagrams or not](https://www.geeksforgeeks.org/check-if-all-levels-of-two-trees-are-anagrams-or-not/)
10. [Count of total anagram substrings](https://www.geeksforgeeks.org/count-total-anagram-substrings/)
11. [Minimum Number of Manipulations required to make two Strings Anagram Without Deletion of Character](https://www.geeksforgeeks.org/minimum-number-of-manipulations-required-to-make-two-strings-anagram-without-deletion-of-character/)

[More >>](https://www.geeksforgeeks.org/tag/anagram/)

**Palindrome :**

1. [C Program to Check if a Given String is Palindrome](https://www.geeksforgeeks.org/c-program-check-given-string-palindrome/)
2. [Check if a given string is a rotation of a palindrome](https://www.geeksforgeeks.org/check-given-string-rotation-palindrome/)
3. [C++ Program to print all palindromes in a given range](https://www.geeksforgeeks.org/c-program-print-palindromes-given-range/)
4. [Check if characters of a given string can be rearranged to form a palindrome](https://www.geeksforgeeks.org/check-characters-given-string-can-rearranged-form-palindrome/)
5. [Dynamic Programming | Set 28 (Minimum insertions to form a palindrome)](https://www.geeksforgeeks.org/dynamic-programming-set-28-minimum-insertions-to-form-a-palindrome/)
6. [Longest Palindromic Substring | Set 2](https://www.geeksforgeeks.org/longest-palindromic-substring-set-2/)
7. [Find all palindromic sub-strings of a given string](https://www.geeksforgeeks.org/find-palindromic-sub-strings-given-string-set-2/)
8. [Online algorithm for checking palindrome in a stream](https://www.geeksforgeeks.org/online-algorithm-for-checking-palindrome-in-a-stream/)
9. [Given a string, print all possible palindromic partitions](https://www.geeksforgeeks.org/given-a-string-print-all-possible-palindromic-partition/)
10. [Print all palindromic partitions of a string](https://www.geeksforgeeks.org/print-palindromic-partitions-string/)
11. [Dynamic Programming | Set 17 (Palindrome Partitioning)](https://www.geeksforgeeks.org/dynamic-programming-set-17-palindrome-partitioning/)
12. [Count All Palindromic Subsequence in a given String](https://www.geeksforgeeks.org/count-palindromic-subsequence-given-string/)
13. [Minimum characters to be added at front to make string palindrome](https://www.geeksforgeeks.org/minimum-characters-added-front-make-string-palindrome/)
14. [Palindrome Substring Queries](https://www.geeksforgeeks.org/palindrome-substring-queries/)
15. [Suffix Tree Application 6 – Longest Palindromic Substring](https://www.geeksforgeeks.org/suffix-tree-application-6-longest-palindromic-substring/)
16. [Palindrome pair in an array of words (or strings)](https://www.geeksforgeeks.org/palindrome-pair-in-an-array-of-words-or-strings/)
17. [Make largest palindrome by changing at most K-digits](https://www.geeksforgeeks.org/make-largest-palindrome-changing-k-digits/)
18. [Lexicographically first palindromic string](https://www.geeksforgeeks.org/lexicographically-first-palindromic-string/)
19. [Recursive function to check if a string is palindrome](https://www.geeksforgeeks.org/recursive-function-check-string-palindrome/)
20. [Minimum number of Appends needed to make a string palindrome](https://www.geeksforgeeks.org/minimum-number-appends-needed-make-string-palindrome/)
21. [Longest Non-palindromic substring](https://www.geeksforgeeks.org/longest-non-palindromic-substring/)
22. [Minimum number of deletions to make a string palindrome](https://www.geeksforgeeks.org/minimum-number-deletions-make-string-palindrome/)
23. [Minimum steps to delete a string after repeated deletion of palindrome substrings](https://www.geeksforgeeks.org/minimum-steps-to-delete-a-string-after-repeated-deletion-of-palindrome-substrings/)
24. [Count of Palindromic substrings in an Index range](https://www.geeksforgeeks.org/count-of-palindromic-substrings-in-an-index-range/)
25. [Minimum insertions to form a palindrome with permutations allowed](https://www.geeksforgeeks.org/minimum-insertions-to-form-a-palindrome-with-permutations-allowed/)
26. [Nth Even length Palindrome](https://www.geeksforgeeks.org/nth-even-length-palindrome/)

[More >>](https://www.geeksforgeeks.org/string-palindrome/)

**Binary String :**

1. [Count of operations to make a binary string”ab” free](https://www.geeksforgeeks.org/count-operations-make-stringab-free/)
2. [Change if all bits can be made same by single flip](https://www.geeksforgeeks.org/change-bits-can-made-one-flip/)
3. [Length of Longest sub-string that can be removed](https://www.geeksforgeeks.org/length-longest-sub-string-can-make-removed/)
4. [Number of flips to make binary string alternate](https://www.geeksforgeeks.org/number-flips-make-binary-string-alternate/)
5. [1’s and 2’s complement of a Binary Number](https://www.geeksforgeeks.org/1s-2s-complement-binary-number/)
6. [Efficient method for 2’s complement of a binary string](https://www.geeksforgeeks.org/efficient-method-2s-complement-binary-string/)
7. [Count binary strings with k times appearing adjacent two set bits](https://www.geeksforgeeks.org/count-binary-strings-k-times-appearing-adjacent-two-set-bits/)
8. [Count strings with consecutive 1’s](https://www.geeksforgeeks.org/count-strings-with-consecutive-1s/)
9. [Generate all binary strings from given pattern](https://www.geeksforgeeks.org/generate-all-binary-strings-from-given-pattern/)
10. [Add two bit strings](https://www.geeksforgeeks.org/add-two-bit-strings/)
11. [Count number of binary strings without consecutive 1’s](https://www.geeksforgeeks.org/count-number-binary-strings-without-consecutive-1s/)
12. [Generate all binary permutations such that there are more or equal 1’s than 0’s before every point in all permutations](https://www.geeksforgeeks.org/generate-binary-permutations-1s-0s-every-point-permutations/)
13. [Check if a string follows a^nb^n pattern or not](https://www.geeksforgeeks.org/check-string-follows-anbn-pattern-not/)
14. [Binary representation of next number](https://www.geeksforgeeks.org/binary-representation-of-next-number/)
15. [Binary representation of next greater number with same number of 1’s and 0’s](https://www.geeksforgeeks.org/binary-representation-next-greater-number-number-1s-0s/)
16. [Maximum difference of zeros and ones in binary string](https://www.geeksforgeeks.org/maximum-difference-zeros-ones-binary-string-set-2-time/)
17. [Check if a binary string has a 0 between 1s or not | Set 2](https://www.geeksforgeeks.org/check-binary-string-0-1s-not-set-2-regular-expression-approach/)
18. [Min flips of continuous characters to make all characters same in a string](https://www.geeksforgeeks.org/min-flips-of-continuous-characters-to-make-all-characters-same-in-a-string/)
19. [Concatenation of two strings in PHP](https://www.geeksforgeeks.org/concatenation-two-string-php/)
20. [Program to add two binary strings](https://www.geeksforgeeks.org/program-to-add-two-binary-strings/)
21. [Convert String into Binary Sequence](https://www.geeksforgeeks.org/convert-string-binary-sequence/)
22. [Generate all binary strings without consecutive 1’s](https://www.geeksforgeeks.org/generate-binary-strings-without-consecutive-1s/)
23. [Minimum number of characters to be removed to make a binary string alternate](https://www.geeksforgeeks.org/minimum-number-characters-removed-make-binary-string-alternate/)
24. [Check divisibility of binary string by 2^k](https://www.geeksforgeeks.org/check-divisibility-binary-string-2k/)
25. Removing elements between the two zeros
26. [Find i’th Index character in a binary string obtained after n iterations](https://www.geeksforgeeks.org/find-ith-index-character-in-a-binary-string-obtained-after-n-iterations/)
27. [Number of substrings with odd decimal value in a binary string](https://www.geeksforgeeks.org/number-of-substrings-with-odd-decimal-value-in-a-binary-string/)
28. [Generate n-bit Gray Codes](https://www.geeksforgeeks.org/given-a-number-n-generate-bit-patterns-from-0-to-2n-1-so-that-successive-patterns-differ-by-one-bit/)
29. [Print N-bit binary numbers having more 1’s than 0’s in all prefixes](https://www.geeksforgeeks.org/print-n-bit-binary-numbers-1s-0s-prefixes/)
30. [Add n binary strings](https://www.geeksforgeeks.org/add-n-binary-strings/)

[More >>](https://www.geeksforgeeks.org/tag/binary-string/)

**Lexicographic pattern :**

1. [Powet Set in Lexicographic order](https://www.geeksforgeeks.org/powet-set-lexicographic-order/)
2. [Lexicographically n-th permutation of string](https://www.geeksforgeeks.org/lexicographically-n-th-permutation-string/)
3. [Lexicographic rank of string using stl](https://www.geeksforgeeks.org/lexicographic-rank-string-using-stl/)
4. [Lexicographically minimum string rotation | Set 1](https://www.geeksforgeeks.org/lexicographically-minimum-string-rotation/)
5. [Generating distinct subsequences of a given string in lexicographic order](https://www.geeksforgeeks.org/generating-distinct-subsequences-of-a-given-string-in-lexicographic-order/)
6. [Lexicographically smallest string obtained after concatenating array](https://www.geeksforgeeks.org/lexicographically-smallest-string-obtained-concatenating-array/)
7. [Lexicographical Maximum substring of string](https://www.geeksforgeeks.org/lexicographical-maximum-substring-string/)
8. [Lexicographical concatenation of all substrings of a string](https://www.geeksforgeeks.org/lexicographical-concatenation-substrings-string/)
9. [Construct lexicographically smallest palindrome](https://www.geeksforgeeks.org/construct-lexicographically-smallest-palindrome/)
10. [Lexicographically smallest string whose hamming distance from given string is exactly K](https://www.geeksforgeeks.org/lexicographically-smallest-string-whose-hamming-distance-given-string-exactly-k/)
11. [Lexicographically next string](https://www.geeksforgeeks.org/lexicographically-next-string/)
12. [Lexicographically largest subsequence such that every character occurs at least k times](https://www.geeksforgeeks.org/lexicographically-largest-subsequence-every-character-occurs-least-k-times/)
13. [Lexicographically first alternate vowel and consonant string](https://www.geeksforgeeks.org/lexicographically-first-alternate-vowel-consonant-string/)
14. [Find a string in lexicographic order which is in between given two strings](https://www.geeksforgeeks.org/find-string-lexicographic-order-given-two-strings/)
15. [Print all permutations in sorted (lexicographic) order](https://www.geeksforgeeks.org/lexicographic-permutations-of-string/)
16. [How to find Lexicographically previous permutation?](https://www.geeksforgeeks.org/lexicographically-previous-permutation-in-c/)
17. [Find n-th lexicographically permutation of a string | Set 2](https://www.geeksforgeeks.org/find-n-th-lexicographically-permutation-string-set-2/)
18. [Lexicographic rank of a string](https://www.geeksforgeeks.org/lexicographic-rank-of-a-string/)

**Pattern Searching :**

1. [Searching for Patterns | Set 5 (Finite Automata)](https://www.geeksforgeeks.org/searching-for-patterns-set-5-finite-automata/)
2. [Pattern Searching | Set 7 (Boyer Moore Algorithm – Bad Character Heuristic)](https://www.geeksforgeeks.org/pattern-searching-set-7-boyer-moore-algorithm-bad-character-heuristic/)
3. [Manacher’s Algorithm – Linear Time Longest Palindromic Substring – Part 4](https://www.geeksforgeeks.org/manachers-algorithm-linear-time-longest-palindromic-substring-part-4/)
4. [Z algorithm](https://www.geeksforgeeks.org/z-algorithm-linear-time-pattern-searching-algorithm/)
5. [Search a Word in a 2D Grid of characters](https://www.geeksforgeeks.org/search-a-word-in-a-2d-grid-of-characters/)
6. [Printing string in plus ‘+’ pattern in the matrix](https://www.geeksforgeeks.org/printing-string-plus-pattern-matrix/)
7. [Wildcard Pattern Matching](https://www.geeksforgeeks.org/wildcard-pattern-matching/)
8. [Dynamic Programming | Wildcard Pattern Matching | Linear Time and Constant Space](https://www.geeksforgeeks.org/dynamic-programming-wildcard-pattern-matching-linear-time-constant-space/)
9. [Replace a character c1 with c2 and c2 with c1 in a string S](https://www.geeksforgeeks.org/replace-character-c1-c2-c2-c1-string-s/)
10. [Aho-Corasick Algorithm](https://www.geeksforgeeks.org/aho-corasick-algorithm-pattern-searching/)
11. [Count of occurrences of a “1(0+)1” pattern in a string](https://www.geeksforgeeks.org/count-of-occurrences-of-a-101-pattern-in-a-string/)
12. [Find all the patterns of “1(0+)1” in a given string | SET 2](https://www.geeksforgeeks.org/find-patterns-101-given-string-set-2regular-expression-approach/)
13. [In-place replace multiple occurrences of a pattern](https://www.geeksforgeeks.org/place-replace-multiple-occurrences-pattern/)
14. [Find all strings that match specific pattern in a dictionary](https://www.geeksforgeeks.org/find-all-strings-that-match-specific-pattern-in-a-dictionary/)
15. [Check if string follows order of characters defined by a pattern or not](https://www.geeksforgeeks.org/check-if-string-follows-order-of-characters-defined-by-a-pattern-or-not-set-3/)
16. [Find nth term of the Dragon Curve Sequence](https://www.geeksforgeeks.org/find-nth-term-dragon-curve-sequence/)
17. [Count of number of given string in 2D character array](https://www.geeksforgeeks.org/find-count-number-given-string-present-2d-character-array/)

[More >>](https://www.geeksforgeeks.org/category/algorithm/pattern-searching/)

**Split String :**

1. [Tokenizing a string in C++](https://www.geeksforgeeks.org/tokenizing-a-string-cpp/)
2. [Split a sentence into words in C++](https://www.geeksforgeeks.org/split-a-sentence-into-words-in-cpp/)
3. [How to split a string in C/C++, Python and Java?](https://www.geeksforgeeks.org/how-to-split-a-string-in-cc-python-and-java/)
4. [Check if given string can be split into four distinct strings](https://www.geeksforgeeks.org/check-given-string-can-split-four-distinct-strings/)
5. [Split numeric, alphabetic and special symbols from a String](https://www.geeksforgeeks.org/split-numeric-alphabetic-and-special-symbols-from-a-string/)
6. [Splitting a Numeric String](https://www.geeksforgeeks.org/splitting-numeric-string/)
7. [Ways to split string such that each partition starts with distinct character](https://www.geeksforgeeks.org/ways-split-string-partition-starts-distinct-character/)
8. [Partition a number into two divisble parts](https://www.geeksforgeeks.org/partition-number-two-divisble-parts/)
9. [Partition given string in such manner that i’th substring is sum of (i-1)’th and (i-2)’th substring](https://www.geeksforgeeks.org/partition-given-string-manner-ith-substring-sum-1th-2th-substring/)
10. [Breaking a number such that first part is integral division of second by a power of 10](https://www.geeksforgeeks.org/breaking-number-first-part-integral-division-second-power-10/)
11. [Divide a string in N equal parts](https://www.geeksforgeeks.org/divide-a-string-in-n-equal-parts/)
12. [Minimum Word Break](https://www.geeksforgeeks.org/minimum-word-break/)
13. [Word Break Problem](https://www.geeksforgeeks.org/word-break-problem-trie-solution/)
14. [Word Break Problem using Backtracking](https://www.geeksforgeeks.org/word-break-problem-using-backtracking/)

**Balance Parentheses & Bracket Evaluation :**

1. [Identify and mark unmatched parenthesis in an expression](https://www.geeksforgeeks.org/identify-mark-unmatched-parenthesis-expression/)
2. [Cost to Balance the parentheses](https://www.geeksforgeeks.org/cost-balance-parantheses/)
3. [Check for balanced parentheses in an expression | O(1) space](https://www.geeksforgeeks.org/check-balanced-parentheses-expression-o1-space/)
4. [Check for balanced parentheses in an expression](https://www.geeksforgeeks.org/check-for-balanced-parentheses-in-an-expression/)
5. [Length of Longest Balanced Subsequence](https://www.geeksforgeeks.org/length-longest-balanced-subsequence/)
6. [Balanced expression with replacement](https://www.geeksforgeeks.org/balanced-expression-replacement/)
7. [Evaluate a boolean expression represented as string](https://www.geeksforgeeks.org/evaluate-a-boolean-expression-represented-as-string/)
8. [Find maximum depth of nested parenthesis in a string](https://www.geeksforgeeks.org/find-maximum-depth-nested-parenthesis-string/)
9. [Print all ways to break a string in bracket form](https://www.geeksforgeeks.org/print-ways-break-string-bracket-form/)
10. [Find an equal point in a string of brackets](https://www.geeksforgeeks.org/find-equal-point-string-brackets/)
11. [Minimum Swaps for Bracket Balancing](https://www.geeksforgeeks.org/minimum-swaps-bracket-balancing/)
12. [Check if two expressions with brackets are same](https://www.geeksforgeeks.org/check-two-expressions-brackets/)
13. [Expression contains redundant bracket or not](https://www.geeksforgeeks.org/expression-contains-redundant-bracket-not/)
14. [Range Queries for Longest Correct Bracket Subsequence](https://www.geeksforgeeks.org/range-queries-longest-correct-bracket-subsequence/)
15. [Evaluate an array expression with numbers, + and –](https://www.geeksforgeeks.org/evaluate-an-array-expression-with-numbers-and/)
16. [Print Bracket Number](https://www.geeksforgeeks.org/print-bracket-number/)
17. [Find index of closing bracket for a given opening bracket in an expression](https://www.geeksforgeeks.org/find-index-closing-bracket-given-opening-bracket-expression/)
18. [Binary tree to string with brackets](https://www.geeksforgeeks.org/binary-tree-string-brackets/)
19. [Construct Binary Tree from String with bracket representation](https://www.geeksforgeeks.org/construct-binary-tree-string-bracket-representation/)
20. [Minimum number of bracket reversals needed to make an expression balanced](https://www.geeksforgeeks.org/minimum-number-of-bracket-reversals-needed-to-make-an-expression-balanced/)

**Conversion :**

1. [Convert all substrings of length ‘k’ from base ‘b’ to decimal](https://www.geeksforgeeks.org/convert-substrings-length-k-base-b-decimal/)
2. [Convert Binary fraction to Decimal](https://www.geeksforgeeks.org/convert-binary-fraction-decimal/)
3. [Convert decimal fraction to binary number](https://www.geeksforgeeks.org/convert-decimal-fraction-binary-number/)
4. [Convert a sentence into its equivalent mobile numeric keypad sequence](https://www.geeksforgeeks.org/convert-sentence-equivalent-mobile-numeric-keypad-sequence/)
5. [Check if it is possible to convert one string into another with given constraints](https://www.geeksforgeeks.org/check-if-it-is-possible-to-convert-one-string-into-another/)
6. [Converting one string to other using append and delete last operations](https://www.geeksforgeeks.org/converting-one-string-using-append-delete-last-operations/)
7. [Converting Decimal Number lying between 1 to 3999 to Roman Numerals](https://www.geeksforgeeks.org/converting-decimal-number-lying-between-1-to-3999-to-roman-numerals/)
8. [Converting Roman Numerals to Decimal lying between 1 to 3999](https://www.geeksforgeeks.org/converting-roman-numerals-decimal-lying-1-3999/)
9. [Inverting the Move to Front Transform](https://www.geeksforgeeks.org/inverting-move-front-transform/)
10. [Burrows – Wheeler Data Transform Algorithm](https://www.geeksforgeeks.org/burrows-wheeler-data-transform-algorithm/)
11. [Check if it is possible to transform one string to another](https://www.geeksforgeeks.org/check-possible-transform-one-string-another/)
12. [Transform the string](https://www.geeksforgeeks.org/transform-the-string/)
13. [An in-place algorithm for String Transformation](https://www.geeksforgeeks.org/an-in-place-algorithm-for-string-transformation/)
14. [Ways of transforming one string to other by removing 0 or more characters](https://www.geeksforgeeks.org/ways-transforming-one-string-removing-0-characters/)
15. [Transform One String to Another using Minimum Number of Given Operation](https://www.geeksforgeeks.org/transform-one-string-to-another-using-minimum-number-of-given-operation/)
16. [Convert Ternary Expression to a Binary Tree](https://www.geeksforgeeks.org/convert-ternary-expression-binary-tree/)
17. [Prefix to Infix Conversion](https://www.geeksforgeeks.org/prefix-infix-conversion/)
18. [Prefix to Postfix Conversion](https://www.geeksforgeeks.org/prefix-postfix-conversion/)
19. [Postfix to Prefix Conversion](https://www.geeksforgeeks.org/postfix-prefix-conversion/)
20. [Postfix to Infix](https://www.geeksforgeeks.org/postfix-to-infix/)

**Misc :**

1. [Word Wrap problem ( Space optimized solution )](https://www.geeksforgeeks.org/word-wrap-problem-space-optimized-solution/)
2. [Form minimum number from given sequence](https://www.geeksforgeeks.org/form-minimum-number-from-given-sequence/)
3. [Maximum number of characters between any two same character in a string](https://www.geeksforgeeks.org/maximum-number-characters-two-character-string/)
4. [Print shortest path to print a string on screen](https://www.geeksforgeeks.org/print-shortest-path-print-string-screen/)
5. [Minimum number of stops from given path](https://www.geeksforgeeks.org/minimum-number-stops-given-path/)
6. [Check whether second string can be formed from characters of first string](https://www.geeksforgeeks.org/check-whether-second-string-can-formed-first-string-using-count-array/)
7. [Mirror characters of a string](https://www.geeksforgeeks.org/mirror-characters-string/)
8. [Find words which are greater than given length k](https://www.geeksforgeeks.org/find-words-greater-given-length-k/)
9. [Find last index of a character in a string](https://www.geeksforgeeks.org/find-last-index-character-string/)
10. [Find position of the given number among the numbers made of 4 and 7](https://www.geeksforgeeks.org/find-position-given-number-among-numbers-made-4-7/)
11. [Find winner of an election where votes are represented as candidate names](https://www.geeksforgeeks.org/find-winner-election-votes-represented-candidate-names/)
12. [Compare Version Numbers with large inputs allowed](https://www.geeksforgeeks.org/compare-version-numbers-large-inputs-allowed/)
13. [Possibility of moving out of maze](https://www.geeksforgeeks.org/possibility-moving-maze/)
14. [Possibility of a word from a given set of characters](https://www.geeksforgeeks.org/possibility-of-a-word-from-a-given-set-of-characters/)
15. [Find the arrangement of queue at given time](https://www.geeksforgeeks.org/find-arrangement-queue-given-time/)
16. [Program to generate all possible valid IP addresses from given string](https://www.geeksforgeeks.org/program-generate-possible-valid-ip-addresses-given-string/)
17. [Program to validate an IP address](https://www.geeksforgeeks.org/program-to-validate-an-ip-address/)
18. [Program to check for a Valid IMEI Number](https://www.geeksforgeeks.org/program-check-valid-imei-number/)
19. [Decode a median string to the original string](https://www.geeksforgeeks.org/decode-median-string-original-string/)
20. [Decode a string recursively encoded as count followed by substring](https://www.geeksforgeeks.org/decode-string-recursively-encoded-count-followed-substring/)
21. [Minimal operations to make a number magical](https://www.geeksforgeeks.org/minimal-operations-make-number-magical/)
22. [Program to check for ISBN](https://www.geeksforgeeks.org/program-check-isbn/)
23. [Program for credit card number validation](https://www.geeksforgeeks.org/program-credit-card-number-validation/)
24. [Maximize a number considering permutations with values smaller than limit](https://www.geeksforgeeks.org/maximize-a-number-considering-permutations-with-values-smaller-than-limit/)
25. [Find if a string starts and ends with another given string](https://www.geeksforgeeks.org/program-find-string-start-end-geeks/)

[More >>](https://www.geeksforgeeks.org/data-structures/string-miscellaneous/)

**Quick Links :**

* [‘Practice Problems’ on Strings](https://practice.geeksforgeeks.org/topics/String/)
* [‘Quizzes’ on Strings](https://www.geeksforgeeks.org/c-language-2-gq/string-gq/)