1. **Java String Introduction**

The elements of a *String* are called *characters*. The number of *characters* in a *String* is called the *length*, and it can be retrieved with the *String.length()* method.

Given two strings of lowercase English letters,  and , perform the following operations:

1. Sum the lengths of  and .
2. Determine if  is lexicographically larger than  (i.e.: does  come before  in the dictionary?).
3. Capitalize the first letter in  and  and print them on a single line, separated by a space.

**Input Format**

The first line contains a string . The second line contains another string . The strings are comprised of only lowercase English letters.

**Output Format**

There are three lines of output:  
For the first line, sum the lengths of  and .  
For the second line, write Yes if  is lexicographically greater than  otherwise print No instead.  
For the third line, capitalize the first letter in both  and  and print them on a single line, separated by a space.

import java.io.\*;

import java.util.\*;

public class Solution {

    public static void main(String[] args) {

        Scanner sc=new Scanner(System.in);

        String A=sc.next();

        String B=sc.next();

        /\* Enter your code here. Print output to STDOUT. \*/

        int totalLength = A.length()+B.length();

        System.out.println(totalLength);

        int compareResult = A.compareTo(B);

        if(compareResult>0)

            System.out.println("Yes");

        else if(compareResult<0)

            System.out.println("No");

        char ch1 = A.charAt(0);

        ch1 = Character.toTitleCase(ch1);

        String newA = String.valueOf(ch1);

        for(int i=1; i<A.length(); i++)

        {

           newA +=  A.charAt(i);

        }

        char ch2 = B.charAt(0);

        ch2 = Character.toTitleCase(ch2);

        String newB = String.valueOf(ch2);

        for(int i=1; i<B.length(); i++)

        {

           newB +=  B.charAt(i);

        }

        System.out.println(newA+" "+newB);

    }

}

1. **Java Substring**

Given a string, , and two indices,  and , print a [substring](https://en.wikipedia.org/wiki/Substring) consisting of all characters in the inclusive range from  to . You'll find the *String* class' [substring method](https://docs.oracle.com/javase/8/docs/api/java/lang/String.html#substring-int-int-) helpful in completing this challenge.

**Input Format**

The first line contains a single string denoting .  
The second line contains two space-separated integers denoting the respective values of  and .

**Constraints**

* String  consists of English alphabetic letters (i.e., ) only.

**Output Format**

Print the substring in the inclusive range from  to .

import java.io.\*;

import java.util.\*;

import java.text.\*;

import java.math.\*;

import java.util.regex.\*;

public class Solution {

    public static void main(String[] args) {

        Scanner in = new Scanner(System.in);

        String S = in.next();

        int start = in.nextInt();

        int end = in.nextInt();

        System.out.println(S.substring(start, end));

    }

}

1. **Java Substring Comparisons**

We define the following terms:

* [Lexicographical Order](https://en.wikipedia.org/wiki/Lexicographical_order), also known as alphabetic or dictionary order, orders characters as follows:

For example, ball < cat, dog < dorm, Happy < happy, Zoo < ball.

* A [substring](https://en.wikipedia.org/wiki/Substring) of a string is a contiguous block of characters in the string. For example, the substrings of abc are a, b, c, ab, bc, and abc.

Given a string, , and an integer, , complete the function so that it finds the lexicographically smallest and largest substrings of length .

**Function Description**

Complete the getSmallestAndLargest function in the editor below.

getSmallestAndLargest has the following parameters:

* string s: a string
* int k: the length of the substrings to find

**Returns**

* string: the string ' + "\n" + ' where and are the two substrings

**Input Format**

The first line contains a string denoting .  
The second line contains an integer denoting .

**Constraints**

* consists of English alphabetic letters only (i.e., [a-zA-Z]).

**Sample Input 0**

welcometojava

3

**Sample Output 0**

ava

wel

**Explanation 0**

String  has the following lexicographically-ordered substrings of length :

We then return the first (lexicographically smallest) substring and the last (lexicographically largest) substring as two newline-separated values (i.e., ava\nwel).

The stub code in the editor then prints ava as our first line of output and wel as our second line of output.

import java.util.Scanner;

public class Solution {

    public static String getSmallestAndLargest(String s, int k) {

        String smallest = s.substring(0,k);

        String largest = s.substring(0,k);

        // Complete the function

        // 'smallest' must be the lexicographically smallest substring of length 'k'

        // 'largest' must be the lexicographically largest substring of length 'k'i));

        for(int i=1; i<s.length()-k+1; i++)

        {

            String substr = s.substring(i, i+k);

            if(smallest.compareTo(substr)>0)

                smallest = substr;

            else if(largest.compareTo(substr)<0)

                largest = substr;

        }

        return smallest + "\n" + largest;

    }

1. **Java String Reverse**

A palindrome is a word, phrase, number, or other sequence of characters which reads the same backward or forward.

Given a string , print Yes if it is a palindrome, print No otherwise.

**Constraints**

* will consist at most  lower case english letters.

import java.io.\*;

import java.util.\*;

public class Solution {

    public static void main(String[] args) {

        Scanner sc=new Scanner(System.in);

        String A=sc.next();

        String result = "";

        if(A.length() == 1)

            result = "Yes";

        else{

            int mid = A.length()/2;

            int last = A.length()-1;

            for(int i=0,j=last; i<mid; i++, j--)

            {

                if(A.charAt(i) == A.charAt(j))

                    result = "Yes";

                else

                    result = "No";

            }

        }

        System.out.println(result);

    }

}

1. **Java String Tokens**

Given a string, , matching the regular expression [A-Za-z !,?.\_'@]+, split the string into tokens. We define a token to be one or more consecutive English alphabetic letters. Then, print the number of tokens, followed by each token on a new line.

**Note:** You may find the [String.split](https://docs.oracle.com/javase/8/docs/api/java/lang/String.html" \l "split-java.lang.String-) method helpful in completing this challenge.

**Input Format**

A single string, .

**Constraints**

* is composed of any of the following: English alphabetic letters, blank spaces, exclamation points (!), commas (,), question marks (?), periods (.), underscores (\_), apostrophes ('), and at symbols (@).

**Output Format**

On the first line, print an integer, , denoting the number of tokens in string  (they do not need to be unique). Next, print each of the  tokens on a new line in the same order as they appear in input string .

import java.io.\*;

import java.util.\*;

public class Solution {

    public static void main(String[] args) {

        Scanner scan = new Scanner(System.in);

        if(!scan.hasNext())

            System.out.println("0");

        else{

                String s = scan.nextLine();

                String []tokens = s.trim().split("[ ,'!?.\_@]+");

                System.out.println(tokens.length);

                for(String token:tokens)

                    System.out.println(token);

        }

        scan.close();

    }

}