**String**

**Special Coding questions**:

1. **Note: a palindrome is a sequence of characters that reads the same forwards and backwards. Ex: Given the following strings... "A man, a plan, a canal: Panama.", return true**

import java.util.Scanner;

class String02{

    static public boolean isPalindrome(String str){

        char[] arr=str.toCharArray();

        for(int i=0;i<arr.length/2;i++){

            if(arr[i]==arr[arr.length-1-i]){

                continue;

            }

            return false;

        }

        return true;

    }

    public static void main(String[] args) {

        Scanner sc=new Scanner(System.in);

        String str=sc.next();

        System.out.println(isPalindrome(str));

    }

}

1. **This question is asked by Google. Given a string, return whether or not it uses capitalization correctly. A string correctly uses capitalization if all letters are capitalized, no letters are capitalized, or only the first letter is capitalized. Ex: Given the following strings... "USA", return true "Calvin", return true "compUter", return false "coding", return true**

import java.util.Scanner;

class String03{

    static public boolean capitalization(String word) {

        if(word.length() < 2) return true;

        if(Character.isUpperCase(word.charAt(0)) && Character.isUpperCase(word.charAt(1))){

            for(int i = 2; i < word.length(); i++){

                if(Character.isLowerCase(word.charAt(i))) return false;

            }

        }

        else{

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

                if(Character.isUpperCase(word.charAt(i))) return false;

            }

        }

        return true;

    }

    public static void main(String[] args) {

        Scanner sc=new Scanner(System.in);

        System.out.println("ENter a String");

        String str=sc.next();

        System.out.println(capitalization(str));

    }

}

1. **This question is asked by Amazon. Given a string representing the sequence of moves a robot vacuum makes, return whether or not it will return to its original position. The string will only contain L, R, U, and D characters, representing left, right, up, and down respectively. Ex: Given the following strings... "LR", return true "URURD", return false "RUULLDRD", return true**
2. import java.util.Scanner;
3. class String04{
4. public static boolean judgeCircle(String moves) {
5. int x = 0;
6. int y = 0;
7. for (char ch : moves.toCharArray()) {
8. if (ch == 'U') y++;
9. else if (ch == 'D') y--;
10. else if (ch == 'R') x++;
11. else if (ch == 'L') x--;
12. }
13. return x == 0 && y == 0;
14. }
15. public static void main(String[] args) {
16. Scanner sc=new Scanner(System.in);
17. String str1=sc.next();
19. System.out.println(judgeCircle(str1));
20. }
21. }

**4. Given two binary strings (strings containing only 1s and 0s) return their sum (also as a binary string).**

**5. Note: neither binary string will contain leading 0s unless the string itself is 0 Ex: Given the following binary strings... "100" + "1", return "101" "11" + "1", return "100" "1" + "0", return "1"**

import java.util.Scanner;

class String05{

    static public String addBinary(String a, String b) {

        int n=a.length();

        int m=b.length();

        int carray=0;

        String ans="";

        int i=0;

        while(i<n||i<m || carray!=0){

            int x=0;

            if(i<n && a.charAt(n-1-i)=='1'){

                 x=1;

            }

            int y=0;

             if(i<m && b.charAt(m-1-i)=='1'){

                 y=1;

            }

             ans=(x+y+carray)%2+ans;

            carray=(x+y+carray)/2;

            i++;

        }

                return ans;

    }

    public static void main(String[] args) {

        Scanner sc=new Scanner(System.in);

        String str1=sc.next();

        String str2=sc.next();

        System.out.println(addBinary(str1,str2));

    }

}

**6. Given an array of strings, return the longest common prefix that is shared amongst all strings.  
7. Note: you may assume all strings only contain lowercase alphabetical characters. Ex: Given the following arrays... ["colorado", "color", "cold"], return "col" ["a", "b", "c"], return "" ["spot", "spotty", "spotted"], return "spot"**

import java.util.Scanner;

// 6. Given an array of strings, return the longest common prefix that is shared amongst all strings

import java.util.Scanner;

public class String06 {

    static public String longestCommonPrefix(String[] arr) {

        if(arr.length<=0)   return "";

        String str=arr[0];

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

            int j=0;

            for(j=0;j<str.length() && j<arr[i].length();j++){

                if(str.charAt(j)==arr[i].charAt(j)){

                    continue;

                }else{

                    break;

                }

            }

             str=str.substring(0,j);

        }

        return str;

    }

    public static void main(String[] args) {

        Scanner sc= new Scanner(System.in);

        System.out.println("Enter a number of elemeent in array");

        int n=sc.nextInt();

        String[] arr=new String[n];

        for(int i=0;i<n;i++){

            arr[i]=sc.next();

        }

        System.out.println(longestCommonPrefix(arr));

    }

}

**8. Given a string and the ability to delete at most one character, return whether or not it can form a palindrome.**

**9. Note: a palindrome is a sequence of characters that reads the same forwards and backwards. Ex: Given the following strings... "abcba", return true "foobof", return true (remove the first 'o', the second 'o', or 'b') "abccab", return false**

import java.util.Scanner;

public class String07 {

    public static void main(String[] args) {

        Scanner sc=new Scanner(System.in);

        String str=sc.next();

        System.out.println(validPalindrome(str));

    }

    public static  boolean validPalindrome(String s) {

        //Declare two pointers

        int left = 0;

        int right = s.length()-1;

       /\*

        Run a loop while left pointer is less than or equal to right pointer.

       \*/

        while(left <= right) {

            if(s.charAt(left) != s.charAt(right)) {

                return isPalindrome(s, left+1, right)

                       ||isPalindrome(s,left,right-1);

            }

            left++;

            right--;

        }

        return true;

    }

    public static boolean isPalindrome(String s, int start, int end) {

        while(start <= end) {

            if(s.charAt(start) != s.charAt(end)) {

                return false;

            }

            start++;

            end--;

        }

        return true;

    }

}

**10. Given a string representing your stones and another string representing a list of jewels, return the number of stones that you have that are also jewels. Ex: Given the following jewels and stones... jewels = "abc", stones = "ac", return 2 jewels = "Af", stones = "AaaddfFf", return 3 jewels = "AYOPD", stones = "ayopd", return 0**

import java.util.\*;

public class String08 {

    public static void main(String[] args) {

        Scanner sc=new Scanner(System.in);

        String str1=sc.next();

        String str2=sc.next();

        System.out.println(numJewelsInStones(str1, str2));

    }

   static  public int numJewelsInStones(String J, String S) {

        int res = 0;

        Set setJ = new HashSet();

        for (char j: J.toCharArray())

            setJ.add(j);

        for (char s: S.toCharArray())

            if (setJ.contains(s)) res++;

        return res;

    }

}

**11. Given two strings, s and t, merge the two strings together alternating characters starting with s. Note: If one string is longer than the other, append the remaining characters of the longer string at the end of the merged string. s = "abc", t = "def", return "adbecf"**

import java.util.Scanner;

public class String09 {

    public static void main(String[] args) {

        Scanner sc=new Scanner(System.in);

        String s1=sc.next();

        String s2=sc.next();

        System.out.println(mergeAlternately(s1, s2));

    }

    public static String mergeAlternately(String w1, String w2) {

        int n = w1.length(), m = w2.length(), i = 0, j = 0;

        StringBuilder res = new StringBuilder();

        while (i < n || j < m) {

            if (i < w1.length())

                res.append(w1.charAt(i++));

            if (j < w2.length())

                res.append(w2.charAt(j++));

        }

        return res.toString();

    }

}

**12. Given a string, s, return the total number of substring within s that contain the same character. Note: You may assume that s only contains lowercase alphabetical characters. Ex: Given the following string s… s = "aabca", return 6 ("a" appears 3 times, "aa" appears 1 time, "b" appears 1 time, and "c" appears 1 time).**

import java.util.Scanner;

public class String10 {

    public static void main(String[] args) {

        Scanner sc=new Scanner(System.in);

        String str=sc.next();

        System.out.println(numberOfSubstrings(str));

    }

    public static  int numberOfSubstrings(String s) {

        int count[] = {0, 0, 0}, res = 0 , i = 0, n = s.length();

        for (int j = 0; j < n; ++j) {

            ++count[s.charAt(j) - 'a'];

            while (count[0] > 0 && count[1] > 0 && count[2] > 0)

                --count[s.charAt(i++) - 'a'];

            res += i;

        }

        return res;

    }

}

**13. Given an encoded string, s, return its decoded representation. The string s will be encoded as follows N[letters], meaning that the letters should be repeated N times in the decoded representation. Note: You may assume s always encoded correctly (i.e. correct formatting of brackets, only positive values outside the brackets, and always lowercase alphabetical characters inside the brackets). Ex: Given the following string s… s = "3[a]2[b]1[c]", return "aaabbc" ("a" is repeated 3 times, "b" is repeated 2 times, and "c" is repeated 1 time).**

import java.util.\*;

public class String11 {

    public static void main(String[] args) {

        Scanner sc=new Scanner(System.in);

        String str=sc.next();

        System.out.println(decodeString(str));

    }

    public static String decodeString(String s) {

        String res = "";

        Stack<Integer> countStack = new Stack<>();

        Stack<String> resStack = new Stack<>();

        int idx = 0;

        while (idx < s.length()) {

            if (Character.isDigit(s.charAt(idx))) {

                int count = 0;

                while (Character.isDigit(s.charAt(idx))) {

                    count = 10 \* count + (s.charAt(idx) - '0');

                    idx++;

                }

                countStack.push(count);

            }

            else if (s.charAt(idx) == '[') {

                resStack.push(res);

                res = "";

                idx++;

            }

            else if (s.charAt(idx) == ']') {

                StringBuilder temp = new StringBuilder (resStack.pop());

                int repeatTimes = countStack.pop();

                for (int i = 0; i < repeatTimes; i++) {

                    temp.append(res);

                }

                res = temp.toString();

                idx++;

            }

            else {

                res += s.charAt(idx++);

            }

        }

        return res;

    }

}

**14. You are given a list of strings, times, where each string represent a timestamp of a twenty-four hour clock (i.e. hours and minutes - “HH:MM”). Return the minimum difference, in minutes, between any two of the timestamps in the list. Ex: Given the following times… times = ["01:00", "01:10"], return 10 (i.e. there are 10 minutes between the two times). Ex: Given the following times… times = ["00:00", "12:23", "05:50", "23:12"], return 48.**

import java.util.Scanner;

public class String12 {

    public static void main(String[] args) {

        Scanner sc=new Scanner(System.in);

        String str=sc.next();

        String str1=sc.next();

        System.out.println(timeGap(str, str1));

    }

    static int getTimeInSeconds(String str) {

        String[] curr\_time = str.split(":");

        int t = Integer.parseInt(curr\_time[0]) \* 60 \* 60 + Integer.parseInt(curr\_time[1]) \* 60

                + Integer.parseInt(curr\_time[2]);

        return t;

    }

    static String convertSecToTime(int t) {

        int hours = t / 3600;

        String hh = hours < 10 ? "0" + Integer.toString(hours)

                : Integer.toString(hours);

        int min = (t % 3600) / 60;

        String mm = min < 10 ? "0" + Integer.toString(min)

                : Integer.toString(min);

        int sec = ((t % 3600) % 60);

        String ss = sec < 10 ? "0" + Integer.toString(sec)

                : Integer.toString(sec);

        String ans = hh + ":" + mm + ":" + ss;

        return ans;

    }

    static String timeGap(String st, String et) {

        int t1 = getTimeInSeconds(st);

        int t2 = getTimeInSeconds(et);

        int time\_diff = (t1 - t2 < 0) ? t2 - t1 : t1 - t2;

        return convertSecToTime(time\_diff);

    }

}

**15. Given a string, s, return the length of the longest substring that contains every vowel occurring an even number of times. Note: You may assume s only contains lowercase alphabetical characters and the vowels you must account for are a, e, i, o, and u. Ex: Given the following string s… s = "aeiouaeioua", return 10 (the last 'a' cannot count). Ex: Given the following string s… s = "bbb", return 3 (all vowels occur an even number of times, i.e. zero times each).**

import java.util.\*;

public class String13 {

    public static void main(String[] args) {

        Scanner sc=new Scanner(System.in);

        String str=sc.next();

        System.out.println(findTheLongestSubstring(str));

    }

     public static int findTheLongestSubstring(String s) {

        int res = 0 , cur = 0, n = s.length();

        HashMap<Integer, Integer> seen = new HashMap<>();

        seen.put(0, -1);

        for (int i = 0; i < n; ++i) {

            cur ^= 1 << ("aeiou".indexOf(s.charAt(i)) + 1 ) >> 1;

            seen.putIfAbsent(cur, i);

            res = Math.max(res, i - seen.get(cur));

        }

        return res;

    }

}

**16. You are given a list of words and asked to find the longest chain. Two words (or more) form a chain if a single letter can be added anywhere in a word, s, to form a word, t (where s and t are both words within the list of words you’re given). Return the length of the longest chain you can form. Ex: Given the following words… words = ["a", "ab", "abc"], return 3 ("a" can be transformed to "ab" by adding a "b" and "ab" can be transformed by adding a "c" giving a chain length of 3). Ex: Given the following words… words = ["a", "abc"], return 1 (both "a" or "abc" form their own chains of size one, they cannot be added together).**

import java.util.\*;

public class String14 {

    public static void main(String[] args) {

        Scanner sc=new Scanner(System.in);

        String str=sc.next();

        System.out.println(longestStrChain(args));

    }

    public static int longestStrChain(String[] words) {

        Map<String, Integer> dp = new HashMap<>();

        Arrays.sort(words, (a, b)->a.length() - b.length());

        int res = 0;

        for (String word : words) {

            int best = 0;

            for (int i = 0; i < word.length(); ++i) {

                String prev = word.substring(0, i) + word.substring(i + 1);

                best = Math.max(best, dp.getOrDefault(prev, 0) + 1);

            }

            dp.put(word, best);

            res = Math.max(res, best);

        }

        return res;

    }

}

**17. You are given two string arrays, queries and words. For any string, s, f(s) is equal to the number of times the smallest lexicographical characters occurs in s. For each query, queries[i] count the total number of words where f(queries[i]) < f(word) and return the answer as an array. Note: Both queries and words will only contain lowercase alphabetical characters and contain at most ten strings each. Ex: Given the following queries and words… queries = ["abc"], words = ["def"], return 0 ('a' and 'd' both occur once so f("abc") and f("def") are equal). Ex: Given the following queries and words… queries = ["abc"], words = ["ddef", "xxyz"], return 2 ('a' appears once and 'd' and 'x' appear twice so f("abc") is less than both f("ddef") and f("xxyz")).**