# Workshop Notes 31/08/2024

There are sample solutions for each problem in the strings.js file at \_\_\_

## Q1: Given a string which consists of only “>”, ”^”, ”V”, ”<”, e.g “VV^<>><^”, how many characters do you need to change to make them all the same.

1. Don’t make the problem harder than it is
   * You only need to return the number of characters you need to change and not specify which characters.
   * There are only 4 possible characters, so it would be acceptable for the solution to maintain a record of the counts for each
2. Develop a simplifying theory
   * Here, the theory is you only need to find the character which appears most often, and the flip the others to be the same, i.e.

Answer = length of string – number of occurrences of most frequent character

1. Break the problem down into smaller steps
   * Turn the string into an array of characters:

let array\_copy = input.split("");

* + Count the number of times each character appears:

let sums = array\_copy.reduce(

(a,b) => {a[b] ? a[b]++ : a[b]=1; return a;},

{}

);

* + Subtract the highest count from the length of the input:

input.length - Math.max(...Object.values(sums)))

## Q2: Given a string containing upper and lower case characters, find how many of the letters follow the pattern << at least one lower case followed by at least one upper case>>.

Example: “aaaBABACcccdDDAA”. Only “a” and “d” follow this pattern, answer = 2

1. Don’t make the problem harder than it is:

Although the string itself might be very long, there are only 26 letters of the alphabet. This makes it possible to keep track of each letter, without the possibility of unlimited memory usage. We can cache or “memoise” results as the program runs

1. Develop a simplifying theory:
   1. The first occurrence of any letter must be the lowercase.
   2. There must be an uppercase version of the letter later in the string.
   3. The must not be a lowercase version after the first occurrence of the uppercase version
2. Develop a possible strategy for the solution
   * 1. Loop over each character in the string and once a letter is seen for the first time, determine whether the subsequent occurrences of that letter follow the pattern. At first glance this looks like it’s O(N2) but actually, since there are only 26 distinct letters it’s 26\*O(N).
     2. For each of the 26 letters, determine whether it follows the pattern in the string. 26\*O(N)
     3. Loop over each character in the string and maintain a “state” for each character (“memoise”) which tells us whether:
        1. The character has not yet been
        2. The lowercase version has been seen
        3. The uppercase version has been seen
        4. The pattern has been broken

Then at the end of the loop, count the number of letters which are in the required state (3). Since this solution only involves scanning the input string once, it could be said it’s more efficient than i) and ii) but on the other hand it needs to do the extra work of maintaining the state of each letter in every iteration, whereas i) can be optimised by exiting early once all 26 letters have been seen, and ii) is constrained to test a max of 26 letters anyway.

1. Use built-in Javascript to simplify coding:

I.e. check that input.lastIndeOf(lowercase) < input.indexOf(uppercase)

## Q3: Given a string like “one+two-one+two+one-two…”, write the code to calculate the value of the equation if “one” represents numerical 1 and “two” represents numerical 2.

This was probably intended as test of python knowledge as much as problem solving.

1. Don’t make the problem harder than it is

It’s often the case that strings need to be converted into arrays of characters because the language libraries typically offer more supporting functions for Arrays than Strings. But in this case, the String functions are fine:

eval (input.replaceAll(“one”,”1”).replaceAll(two,”2)) // JS

eval (input.replace(“one”,”1”).replace(two,”2)) // Python