Interview

Step 1: Understand the Problem

### Write a function which takes two numbers and returns their sum.

1. Can I restate the problem in my own words?

“Implement addition”

1. What are the inputs that go into the problem?

* Ints?
* Floats?
* What about string for large numbers?

1. What are the outputs that should come from the solution to the problem?

* Int?
* Float?
* String?

1. Can the outputs be determined from the inputs? In other words. Do I have enough information to solve the problem? (You may not be able to answer this question until you set about solving the problem. That’s okay; it’s still worth considering the question at this early stage.)
2. How should I label the important pieces of data that are a part of the problem?

## Step 2: Explore Concrete Examples

Exploring examples one in two for sure

### Write a function which takes in a string and returns counts of each character in the string.

* Start with Simple Examples

charCount(“aaaa”) // {a:4}

charCount(“hello”) // {h:1, e:1, l:2, o:1 }

* Progress to More Complex Examples

“My phone number is 182763”

* Explore Examples with Empty Inputs

“Hello hi”

charCount(“”)

* Explore Examples with Invalid Inputs

charCount(“”)

## Step 3: Break it Down

Write a function which takes in a string and returns counts of each character in the string.

charCount ("aaaa")

*// {*

*// a:4*

*// }*

charCount ("hello")

*// {*

*// h:1,*

*// e:1,*

*// L:2,*

*// O:1*

*// }*

charCount ("Your PIN number is 1234!")

*/\*{*

*1:1,*

*2:1,*

*3:1,*

*4:1,*

*b:1,*

*e:1,*

*i:2,*

*m:1,*

*n:2,*

*o:1,*

*p:1,*

*r:2,*

*s:1,*

*u:2,*

*y:1*

*}\*/*

function charCount(str) {

*// do something*

*// return an object with keys that are lowercase alphanumeric characters in the string ; values should be the counts for those characters*

};

function charCount(str) {

*// make object to return at end*

*// loop over string, for each character...*

*// if the char is a number/letter and is a key in object , add one to count*

*// if the char is a number/letter and not in object, add it to object and set value to 1*

*// if character is something else (space, period, etc.) don't do anything*

*// return object at end*

}

## Step 4: Solve Or Simplify

### Write a function which takes in a string and returns counts of each character in the string.

function charCount(str) {

*// make object to return at end*

var result ={};

*// loop over string, for each character...*

for (var i=0; i< str.length; i++) {

var char = str[i].toLowerCase()

*// if the char is a number/letter and is a key in object , add one to count*

if (result[char] > 0) {

result[char] ++;

}

*// if the char is a number/letter and not in object, add it to object and set value to 1*

else {

result[char] = 1;

}

}

*// if character is something else (space, period, etc.) don't do anything*

*// return object at end*

return result;

}

## Step 5: Look Back and Refactor

Alphanumeric (charCode vs RegExp)

RegExp

function charCount(str) {

var obj = {};

for (var char of str) {

char = char.toLowerCase();

if (/[a-z0-9]/.test(char)) {

obj[char] = ++obj[char] || 1;

}

}

return obj;

}

charCode (fastest)

function charCount(str) {

var obj = {};

for (var char of str) {

if (isAlphaNumeric(char)) {

char = char.toLowerCase();

obj[char] = ++obj[char] || 1;

}

}

return obj;

}

function isAlphaNumeric(char) {

var code= char.charCodeAt(0);

if (!(code >47 && code<58) && *// numeric (0-9)*

!(code > 64 && code < 91) && *//upper alpha (A-Z)*

!(code > 96 && code < 123)){ *// lower alpha (a-z)*

return false;

}

return true;

}

AN EXAMPLE

Write a function called same, which accepts two arrays. The function should return true if every value in the array has its corresponding value squared in the second array. The frequency of values must be the same.

##### same\_naive\_solution

function same(arr1, arr2){

if(arr1.length !== arr2.length){

return false;

}

for(let i = 0; i < arr1.length; i++){

let correctIndex = arr2.indexOf(arr1[i] \*\* 2)

if(correctIndex === -1) {

return false;

}

console.log(arr2);

arr2.splice(correctIndex,1)

}

return true;

}

same([1,2,3,2], [9,1,4,4])

##### same\_refactored\_solution

function same(arr1, arr2){

if(arr1.length !== arr2.length){

return false;

}

let frequencyCounter1 = {}

let frequencyCounter2 = {}

for(let val of arr1){

frequencyCounter1[val] = (frequencyCounter1[val] || 0) + 1

}

for(let val of arr2){

frequencyCounter2[val] = (frequencyCounter2[val] || 0) + 1

}

console.log(frequencyCounter1);

console.log(frequencyCounter2);

for(let key in frequencyCounter1){

if(!(key \*\* 2 in frequencyCounter2)){

return false

}

if(frequencyCounter2[key \*\* 2] !== frequencyCounter1[key]){

return false

}

}

return true

}

same([1,2,3,2,5], [9,1,4,4,11])

Coding Exercise

Coding Exercise 1: Frequency Counter - validAnagram

### Frequency Counter: Anagram Challenge

Given two strings, write a function to determine if the second string is an anagram of the first. An anagram is a word, phrase, or name formed by rearranging the letters of another, such as cinema , formed from iceman.

Examples:

validAnagram('', '') *// true*

validAnagram('aaz', 'zza') *// false*

validAnagram('anagram','nagaram') *//true*

validAnagram('rat', 'car') *// false*

validAnagram('awesome', 'awesom') *// false*

validAnagram('amanaplanacana;panama', 'acanalmanplanpamana') *//false*

validAnagram('qwerty','qeywrt') *//true*

validAnagram('rexttwisttime','timetwisttext')*//true*

Note :You may assume the string contains only lowercase alphabets.

Time Complexity - O(n)

function validAnagram(first, second) {

if (first.length !== second.length) {

return false;

}

const lookup = {};

for (let i = 0; i < first.length; i++) {

let letter = first[i];

// if letter exists, increment, otherwise set to 1

lookup[letter] ? lookup[letter] += 1 : lookup[letter] = 1;

}

console.log(lookup)

for (let i = 0; i < second.length; i++) {

let letter = second[i];

// can't find letter or letter is zero then it's not an anagram

if (!lookup[letter]) {

return false;

} else {

lookup[letter] -= 1;

}

}

return true;

}

// {a: 0, n: 0, g: 0, r: 0, m: 0,s:1}

validAnagram('anagrams', 'nagaramm')

## Coding Exercise 2: Multiple Pointers: CountUniqueValues

### Multiple Pointers: CountUniqueValues

Implement a function called **countUniqueValues**, which accepts a sorted array, and counts the unique values in the array. There can be negative numbers in the array , but it will always be sorted.

countUniqueValues([1,1,1,1,1,2]) *// 2*

countUniqueValues([1,2,3,4,4,4,7,7,12,12,13]) *//7*

countUniqueValues([]) *//0*

countUniqueValues([-2,-1,-1,0,1]) *// 4*

Time Complexity - O(n)

Space Complexity - O(n)

**Bonus**

You must do this with constant or O(1) space and O(n) time.

function countUniqueValues(arr){

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

var i = 0;

for(var j = 1; j < arr.length; j++){

if(arr[i] !== arr[j]){

i++;

arr[i] = arr[j]

}

}

return i + 1;

}

countUniqueValues([1,2,2,5,7,7,99])

## Coding Exercise 3: Frequency Counter - sameFrequency

### Frequency Counter - sameFrequency

Write a function called sameFrequency. Given two positive integers, find out if the two numbers have the same frequency of digits.

Your solution MUST have the following complexities:

Time: O(N)

sameFrequency(182,281) *//true*

sameFrequency(34,14) *//false*

sameFrequency(3589578, 5879385) *//true*

sameFrequency(22, 222) *// false*