<https://ca-magic-hours.github.io/js_exercises/>

Exercise #1

Write a function that takes a string as an argument (input) and returns an array containing the position of each vowel in the string. Note: We want the ‘actual’ position of each vowel, not its index.

EX: If given the string ‘super’ we want an array that returns [2,4] (the second and fourth letter in the string).

function vowelIndices(word) {   
var vowels = ['a', 'e', 'i', 'o', 'u', 'y'];   
var answer = [];   
for (i = 0; i < word.length; i++) {   
for (j = 0; j < vowels.length; j++) {   
if (word[i] === vowels[j]) {   
answer.push(i + 1);   
};   
};   
};   
return answer;   
};

     Exercise #2

Write a function that accepts two arrays as arguments. The first array is a collection of items, and the second array is a collection of integers. The output should be an array that contains the items from the first array at the index as denoted by the integer from the second array. If *either* array is empty, return an empty array.

EX1: [‘a’, ‘b’, 'c', ‘d'] and [2,0], output: ['c', ‘a’]   
EX2: [1,2,3,4,5] and [2,4,0] output: [3, 5, 1]   
EX3: [0,3,4] and [2,6]output: [4]

function findArray(arr1, arr2) {   
var find\_array = [];   
if (arr1.length > 0 && arr2.length > 0) {   
for (var i = 0; i < arr2.length; i++) {   
find\_array.push(arr1[arr2[i]]);   
};   
};   
return find\_array;   
};

     Exercise #3

The Western Suburbs Croquet Club has two categories of membership, Senior and Open. They would like your help with an application form that will tell prospective members which category they will be placed.

To be a senior, a member must be at least 55 years old and have a handicap greater than 7. In this croquet club, handicaps range from -2 to +26; the better the player the lower the handicap.

Input will consist of a list of lists containing two items each. Each list contains information for a single potential member. Information consists of an integer for the person's age and an integer for the person's handicap.

Output will consist of a list of string values stating whether the respective member is to be placed in the senior or open category.

EX:   
Given this input: [[18, 20],[45, 2],[61, 12],[37, 6],[21, 21],[78, 9]]   
This is the result: ["Open", "Open", "Senior", "Open", "Open", "Senior"]

function openOrSenior(data){   
var result = [];   
for(i = 0; i < data.length; i++) {   
var age = data[i][0];   
var handicap = data[i][1];   
if (age >= 55 && handicap > 7) {   
result.push("Senior");   
} else {   
result.push("Open");   
};   
};   
return result;

};

     Exercise #4

Write a function that returns the sum of all numbers up to and including 'n' that are divisible by 3 or 5.

EX1: if 'n' is 5, then output is 8 (because in 1,2,3,4,5 only 3 and 5 are divisible by 3 or 5, and 3 + 5 = 8)   
EX2: if 'n' is 10, then output is 33 (because 1,2,3,4,5,6,7,8,9,10, only 3,5,6,9,10 are divisible by 3 or 5, and 3 + 5 + 6 + 9 + 10 = 33)

function findSum(n) {   
var acc = 0;   
for (i = 0; i <= n; i++) {   
if(i % 3 === 0 || i % 5 === 0) {   
acc = acc+i;   
};   
};   
return acc;   
};

     Exercise #5

Given an array of numbers, find the smallest and largest numbers and then difference between the two. Output should be an array: [smallest, largest, difference]

EX: Given [1,2,3,4,5,6], Output [1,6,5]

function differenceInNums(array) {   
var sorted = array.sort(function(a,b) {   
return a - b;   
};   
return [sorted[0], sorted[sorted.length-1], sorted[sorted.length-1] - sorted[0]];   
};

     Exercise #6

Write a function that takes 2 parameters. The first is a number, the second is a limit. Your goal is to produce the SUM of ALL multiples of the given number up to but NOT including the limit.

EX1: If you are given 2 (number) and 9 (limit), your multiples are 2,4,6,8 and your result should be the SUM of these numbers (20).   
EX2: If given 3 (number) and 13 (limit), your multiples are 3,6,9,12 and your result should be the SUM of these numbers (30).

function sumMul(num, lim){   
var acc = 0;   
if (lim <= 0 ) {   
return 'INVALID';   
};   
for (i = 0; i <= Math.floor(lim / num); i++) {   
acc = acc + (num \* i);   
};   
return acc;   
};

     Exercise #7

You are given a 3 digit number. Re-arrange the digits so that you produce the largest possible number combination. If you're given an input that isn't a 3-digit number return 'null'.

EX1: Input: 123, Output: 321   
EX2: Input: 'abc', Output: null   
EX3: Input: 1234, Output: null

function maxRedigit(num) {   
if (num <= 0 || isNaN(num) || num.toString().length <= 2 || num.toString().length >= 4) {   
return null;   
};   
return parseInt(num.toString().split('').sort(function(a,b) {   
return b - a;   
}).join(''));   
};

     Exercise #8

Write a function that accepts a list of strings. Each letter in the string has a value equal to its position in the alphabet ( a = 1, b = 2, c = 3, ... , z = 26). Spaces have no value.

Your output should be an array containing the value of the string (sum of its letter values) multiplied by its position in the input array (ie: 1st, 2nd, not its index).

The string 'abc' has a value of 6 ( a = 1, b = 2, c = 3 and 1 + 2 + 3 = 6).   
  
EX1: ['abc', 'abc'] should return [6, 12] because (6 \* 1, 6 \* 2)   
EX2: ['abc', 'abc abc'] should return [6, 24] because ( 6 \* 1, (6 + 6) \* 2)

function wordValue(arr) {   
var alphabet = 'abcdefghijklmnopqrstuvwxyz';   
var output = [];   
var pointObj = {};   
  
for (i = 1; i <= alphabet.length; i++) {   
var letter = alphabet.split('')[i - 1];   
pointObj[letter] = i;   
};   
  
for (i = 0; i < arr.length; i++) {   
var value = 0;   
var item = arr[i].split('');   
for (j = 0; j < item.length; j++) {   
if (!pointObj[item[j]]) {   
continue;   
};   
value += pointObj[item[j]];   
};   
value \*= (i + 1);   
output.push(value);   
};   
return output;   
};

     Exercise #9

Write a function that takes 2 parameters: a sum and a product. The goal is to find the numbers that when ADDED together give you the sum AND when multiplied together give you the product. Output should be an array with the two correct numbers. If no combination exists, return 'null'.

EX1: Input: 13 (sum) and 12 (product), Output: [1,12] (because 1 \* 12 = 12, and 1 + 12 = 13)   
EX2: Input 12 (sum) and 32 (product), Output: [8,4] (because 8 \* 4 = 32, and 8 + 4 = 12)

function sumAndMultiply(sum, multiply) {   
for(var num1 = 0; num1 < sum; num1++) {   
var ans = [];   
var num2 = sum - num1;   
  
if(num1 \* num2 === multiply) {   
ans.push(num1);   
ans.push(num2);   
return ans;   
};   
};   
return null;   
};

     Exercise #10

Write a function that accepts two lists as arguments. The first is a list of prices, the second is a list of fruits to purchase. In the case of duplicate fruits, the same price is to be used (ie: if you have to buy 2 Apples, you buy them for the same price - the prices are per TYPE of fruit, not per ITEM). You want to return an array of 2 items: the BEST possible outcome, and the WORST possible outcome.

EX1:   
Inputs: You have prices of [4,2,1,10,5] and a list of ["apple", "orange", "mango"]   
Your BEST case scenario is that each fruit costs as little as possible, so apple = 1, orange = 2, mango = 4, 1+2+4 = 7. So, BEST CASE SCENARIO you get everything you want for $7.   
Your WORST case scenario is that each fruit costs as much as possible, so apple = 10, orange = 5, mango = 4 (the fact mango is 4 in both cases is not intentional), 10 + 5 + 4 = 19. So, WORST CASE SCENARIO you get everything you want for $19.   
Output would be [7,19]

EX2:   
Inputs: You have prices of [3,5,1,6,8,1] and a list of ["peach", "grapefruit", "banana", "orange", "orange"] (note 'orange' appears twice)   
Your best case scenario: orange = 1, peach = 1, grapefruit = 3, banana = 5. (1 \* 2) + 1 + 3 + 5 = 11.   
Your worst case scenario: orange = 8, peach = 6, grapefruit = 5, banana = 3. (8 \* 2) + 6 + 5 + 3 = 30.   
Output would be [11,30]

function buyFruits(priceLabels,fruitsList){   
var uniqueFruits = [];   
var fruitCount = {};   
var best = 0;   
var worst = 0;   
  
// We use this loop to create an array that contains ONLY unique items (ie: no duplicates)   
for (i = 0; i < fruitsList.length; i++) {   
if (uniqueFruits.indexOf(fruitsList[i]) === -1) {   
uniqueFruits.push(fruitsList[i]);   
};   
};   
  
// We use this loop create an object with properties equal to the names of the fruits, and values equal to the number of times the fruit appears in the original list.   
for (i = 0; i < fruitsList.length; i++) {   
if (!fruitCount[fruitsList[i]]) {   
fruitCount[fruitsList[i]];   
fruitCount[fruitsList[i]] = 1;   
} else {   
fruitCount[fruitsList[i]]++;   
};   
};   
  
// We use this loop to figure out what the best case scenario is for the total bill   
for (i = 0; i < uniqueFruits.length; i++) {   
// We arrange the prices in ascending order (small to big)   
var ascendingPrices = priceLabels.sort(function(a,b) {   
return a - b;   
});   
// We arrange the array of unique fruits in descending order, using the values of fruitCount as our comparators   
// ie: if 'orange' appears 3 times in our primary list, it's value in fruitCount will be 3, and we'll want that at the beginning of our array   
// We want that because our prices are assorted small -> big, and out fruits are assorted most common -> least common   
// If a fruit is more common, we want it to be as cheap as possible   
// We arrange the arrays this way so it's easy to use 'i' to target the various things we'll need to target   
var descendingFruitCount = uniqueFruits.sort(function(a,b) {   
return fruitCount[b] - fruitCount[a];   
});   
// fruitCount[descendingFruitCount[i]] = the property on fruitCount that associates with the value of descendingFruitCount[i]   
// If descendingFruitCount[i] = 'orange', then fruitCount[descendingFruitCount[i]] = fruitCount['orange'], which is equal to fruitCount.orange   
best += (fruitCount[descendingFruitCount[i]] \* ascendingPrices[i]);   
};   
  
for (i = 0; i < uniqueFruits.length; i++) {   
// We arrange the prices in descending order (big to small)   
var descendingPrices = priceLabels.sort((a,b) => b - a);   
// We arrange the array of unique fruits in descending order, using the values of fruitCount as our comparators   
// ie: if 'orange' appears 3 times in our primary list, it's value in fruitCount will be 3, and we'll want that at the beginning of our array   
// We want that because our prices are assorted big -> small, and out fruits are assorted most common -> least common   
// If a fruit is more common, we want it to be as expensive as possible   
// We arrange the arrays this way so it's easy to use 'i' to target the various things we'll need to target   
var descendingFruitCount = uniqueFruits.sort(function(a,b) {   
return fruitCount[b] - fruitCount[a];   
});   
// fruitCount[descendingFruitCount[i]] = the property on fruitCount that associates with the value of descendingFruitCount[i]   
// If descendingFruitCount[i] = 'orange', then fruitCount[descendingFruitCount[i]] = fruitCount['orange'], which is equal to fruitCount.orange   
worst += (fruitCount[descendingFruitCount[i]] \* descendingPrices[i]);   
};   
// Return an array that lists the best possible outcome and the worst possible outcome   
return [best, worst];   
};

John Sutherland

jrms999@gmail.com