**Secuencias de comandos de algoritmos intermedios**

Suma todos los números en un rango

Le pasaremos una matriz de dos números. Devuelve la suma de esos dos números más la suma de todos los números entre ellos. El número más bajo no siempre vendrá primero.

Por ejemplo, sumAll([4,1])debe regresar 10porque la suma de todos los números entre 1 y 4 (ambos inclusive) es 10.

**function sumAll(arr) {**

**let max = Math.max(arr[0], arr[1]);**

**let min = Math.min(arr[0], arr[1]);**

**let sumBetween = 0;**

**for (let i = min; i <= max; i++) {**

**sumBetween += i;**

**}**

**return sumBetween;**

**}**

**sumAll([1, 4]);**

**Diferenciar dos matrices**

Compare dos matrices y devuelva una nueva matriz con cualquier elemento que solo se encuentre en una de las dos matrices dadas, pero no en ambas. En otras palabras, devuelva la diferencia simétrica de las dos matrices.

Nota: puede devolver la matriz con sus elementos en cualquier orden.

**function diffArray(arr1, arr2) {**

**const newArr = [];**

**function onlyInFirst(first, second) {**

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

**if (second.indexOf(first[i]) === -1) {**

**newArr.push(first[i]);**

**}**

**}**

**}**

**onlyInFirst(arr1, arr2);**

**onlyInFirst(arr2, arr1);**

**return newArr;**

**}**

**diffArray([1, 2, 3, 5], [1, 2, 3, 4, 5]);**

**Buscar y destruir**

Se le proporcionará una matriz inicial (el primer argumento de la destroyerfunción), seguida de uno o más argumentos. Elimina todos los elementos de la matriz inicial que tengan el mismo valor que estos argumentos.

Nota: Tienes que usar el argumentsobjeto.

**function destroyer(arr) {**

**const valsToRemove = Object.values(arguments).slice(1);**

**const filteredArray = [];**

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

**let removeElement = false;**

**for (let j = 0; j < valsToRemove.length; j++) {**

**if (arr[i] === valsToRemove[j]) {**

**removeElement = true;**

**}**

**}**

**if (!removeElement) {**

**filteredArray.push(arr[i]);**

**}**

**}**

**return filteredArray;**

**}**

**Donde estás**

Cree una función que mire a través de una matriz de objetos (primer argumento) y devuelva una matriz de todos los objetos que tienen pares de nombre y valor coincidentes (segundo argumento). Cada par de nombre y valor del objeto de origen debe estar presente en el objeto de la colección si se va a incluir en la matriz devuelta.

Por ejemplo, si el primer argumento es [{ first: "Romeo", last: "Montague" }, { first: "Mercutio", last: null }, { first: "Tybalt", last: "Capulet" }]y el segundo argumento es { last: "Capulet" }, entonces debe devolver el tercer objeto de la matriz (el primer argumento), porque contiene el nombre y su valor, que se pasó como el segundo argumento.

**function whatIsInAName(collection, source) {**

**// What's in a name?**

**const arr = [];**

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

**let found = true;**

**for (const sourceProp in source) {**

**if (collection[i][sourceProp] !== source[sourceProp]) {**

**found = false;**

**break;**

**}**

**}**

**if (found) arr.push(collection[i]);**

**}**

**return arr;**

**}**

**whatIsInAName([{ first: "Romeo", last: "Montague" }, { first: "Mercutio", last: null }, { first: "Tybalt", last: "Capulet" }], { last: "Capulet" });**

**Caso de punción lumbar**

Convierte una cuerda en una caja espinal. El caso espinal es todo-palabras-en-minúsculas-unidas-por-guiones.

**function spinalCase(str) {**

**// Create a variable for the white space and underscores.**

**var regex = /\s+|\_+/g;**

**// Replace low-upper case to low-space-uppercase**

**str = str.replace(/([a-z])([A-Z])/g, "$1 $2");**

**// Replace space and underscore with -**

**return str.replace(regex, "-").toLowerCase();**

**}**

**// test here**

**spinalCase("This Is Spinal Tap");**

**Jerga**

Pig Latin es una forma de alterar palabras en inglés. Las reglas son las siguientes:

- Si una palabra comienza con una consonante, tome la primera consonante o grupo de consonantes, muévala al final de la palabra y añádala ay.

- Si una palabra comienza con una vocal, solo agregue wayal final.

**function translatePigLatin(str) {**

**// Create variables to be used**

**var pigLatin = "";**

**var regex = /[aeiou]/gi;**

**// Check if the first character is a vowel**

**if (str[0].match(regex)) {**

**pigLatin = str + "way";**

**} else if (str.match(regex) === null) {**

**// Check if the string contains only consonants**

**pigLatin = str + "ay";**

**} else {**

**// Find how many consonants before the first vowel.**

**var vowelIndice = str.indexOf(str.match(regex)[0]);**

**// Take the string from the first vowel to the last char**

**// then add the consonants that were previously omitted and add the ending.**

**pigLatin = str.substr(vowelIndice) + str.substr(0, vowelIndice) + "ay";**

**}**

**return pigLatin;**

**}**

**// test here**

**translatePigLatin("consonant");**

**Buscar y reemplazar**

Realice una búsqueda y reemplace en la oración utilizando los argumentos proporcionados y devuelva la nueva oración.

El primer argumento es la oración para realizar la búsqueda y reemplazar.

El segundo argumento es la palabra que reemplazará (antes).

El tercer argumento es con lo que reemplazará el segundo argumento (después).

Nota: Preserve las mayúsculas y minúsculas del primer carácter de la palabra original cuando lo reemplace. Por ejemplo, si desea reemplazar la palabra Bookcon la palabra dog, debe reemplazarse comoDog

**function myReplace(str, before, after) {**

**// Find index where before is on string**

**var index = str.indexOf(before);**

**// Check to see if the first letter is uppercase or not**

**if (str[index] === str[index].toUpperCase()) {**

**// Change the after word to be capitalized before we use it.**

**after = after.charAt(0).toUpperCase() + after.slice(1);**

**} else {**

**// Change the after word to be uncapitalized before we use it.**

**after = after.charAt(0).toLowerCase() + after.slice(1);**

**}**

**// Now replace the original str with the edited one.**

**str = str.replace(before, after);**

**return str;**

**}**

**// test here**

**myReplace("A quick brown fox jumped over the lazy dog", "jumped", "leaped");**

**Emparejamiento de ADN**

Los pares de cadenas de ADN consisten en pares de nucleobases. Los pares de bases están representados por los caracteres AT y CG , que forman bloques de construcción de la doble hélice del ADN.

A la hebra de ADN le falta el elemento de emparejamiento. Escribe una función para hacer coincidir los pares de bases que faltan para la hebra de ADN proporcionada. Para cada carácter de la cadena proporcionada, busque el carácter del par base. Devuelve los resultados como una matriz 2d.

Por ejemplo, para la entrada GCG, devuelva[["G", "C"], ["C","G"], ["G", "C"]]

El carácter y su pareja se emparejan en una matriz, y todas las matrices se agrupan en una matriz encapsulada.

**function pairElement(str) {**

**// create object for pair lookup**

**const pairs = {**

**A: "T",**

**T: "A",**

**C: "G",**

**G: "C"**

**};**

**// map character to array of character and matching pair**

**return str**

**.split("")**

**.map(x => [x, pairs[x]]);**

**}**

**// test here**

**pairElement("GCG");**

**Cartas perdidas**

Encuentre la letra que falta en el rango de letras aprobadas y devuélvala.

Si todas las letras están presentes en el rango, devuelve undefined.

**function fearNotLetter(str) {**

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

**/\* code of current character \*/**

**const charCode = str.charCodeAt(i);**

**/\* if code of current character is not equal to first character + no of iteration**

**then a letter was skipped \*/**

**if (charCode !== str.charCodeAt(0) + i) {**

**/\* if current character skipped past a character find previous character and return \*/**

**return String.fromCharCode(charCode - 1);**

**}**

**}**

**return undefined;**

**}**

**// test here**

**fearNotLetter("abce");**

**Unión ordenada**

Escriba una función que tome dos o más matrices y devuelva una nueva matriz de valores únicos en el orden de las matrices proporcionadas originalmente.

En otras palabras, todos los valores presentes de todas las matrices deben incluirse en su orden original, pero sin duplicados en la matriz final.

Los números únicos deben clasificarse según su orden original, pero la matriz final no debe clasificarse en orden numérico.

Consulte las pruebas de aserción para ver ejemplos.

**function uniteUnique(arr1, arr2, arr3) {**

**// Creates an empty array to store our final result.**

**const finalArray = [];**

**// Loop through the arguments object to truly make the program work with two or more arrays**

**// instead of 3.**

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

**const arrayArguments = arguments[i];**

**// Loops through the array at hand**

**for (let j = 0; j < arrayArguments.length; j++) {**

**let indexValue = arrayArguments[j];**

**// Checks if the value is already on the final array.**

**if (finalArray.indexOf(indexValue) < 0) {**

**finalArray.push(indexValue);**

**}**

**}**

**}**

**return finalArray;**

**}**

**uniteUnique([1, 3, 2], [5, 2, 1, 4], [2, 1]);**

**Convertir entidades HTML**

Convierta los caracteres &, <, >, "(comillas dobles) y '(apóstrofe) de una cadena en sus entidades HTML correspondientes.

**function convertHTML(str) {**

**// Use Object Lookup to declare as many HTML entities as needed.**

**const htmlEntities = {**

**"&": "&amp;",**

**"<": "&lt;",**

**">": "&gt;",**

**'"': "&quot;",**

**"'": "&apos;"**

**};**

**// Using a regex, replace characters with it's corresponding html entity**

**return str.replace(/([&<>\"'])/g, match => htmlEntities[match]);**

**}**

**// test here**

**convertHTML("Dolce & Gabbana");**

**Suma todos los números impares de Fibonacci**

Dado un entero positivo num, devuelve la suma de todos los números impares de Fibonacci que son menores o iguales que num.

Los primeros dos números en la secuencia de Fibonacci son 0 y 1. Cada número adicional en la secuencia es la suma de los dos números anteriores. Los primeros siete números de la sucesión de Fibonacci son 0, 1, 1, 2, 3, 5 y 8.

Por ejemplo, sumFibs(10)debe volver 10porque todos los números impares de Fibonacci menores o iguales que 10son 1, 1, 3 y 5.

**function sumFibs(num) {**

**// Perform checks for the validity of the input**

**if (num <= 0) return 0;**

**// Create an array of fib numbers till num**

**const arrFib = [1, 1];**

**let nextFib = 0;**

**// We put the new Fibonacci numbers to the front so we**

**// don't need to calculate the length of the array on each**

**// iteration**

**while ((nextFib = arrFib[0] + arrFib[1]) <= num) {**

**arrFib.unshift(nextFib);**

**}**

**// We filter the array to get the odd numbers and reduce them to get their sum.**

**return arrFib.filter(x => x % 2 != 0).reduce((a, b) => a + b);**

**}**

**// test here**

**sumFibs(4);**

**Suma todos los números primos**

Un número primo es un número entero mayor que 1 con exactamente dos divisores: 1 y él mismo. Por ejemplo, el 2 es un número primo porque solo es divisible por 1 y 2. En cambio, el 4 no es primo porque es divisible por 1, 2 y 4.

Reescribe sumPrimespara que devuelva la suma de todos los números primos que son menores o iguales que num.

**function sumPrimes(num) {**

**// Check all numbers for primality**

**let primes = [];**

**for (let i = 2; i <= num; i++) {**

**if (primes.every((prime) => i % prime !== 0))**

**primes.push(i);**

**}**

**return primes.reduce((sum, prime) => sum + prime, 0);**

**}**

**Mínimo común múltiplo**

Encuentre el múltiplo común más pequeño de los parámetros proporcionados que se puede dividir uniformemente por ambos, así como por todos los números secuenciales en el rango entre estos parámetros.

El rango será una matriz de dos números que no necesariamente estarán en orden numérico.

Por ejemplo, si se dan 1 y 3, encuentre el múltiplo común más pequeño de 1 y 3 que también sea divisible por todos los números entre 1 y 3. La respuesta aquí sería 6.

**function smallestCommons(arr) {**

**// Setup**

**const [min, max] = arr.sort((a, b) => a - b);**

**const range = Array(max - min + 1)**

**.fill(0)**

**.map((\_, i) => i + min);**

**// Largest possible value for SCM**

**const upperBound = range.reduce((prod, curr) => prod \* curr);**

**// Test all multiples of 'max'**

**for (let multiple = max; multiple <= upperBound; multiple += max) {**

**// Check if every value in range divides 'multiple'**

**const divisible = range.every((value) => multiple % value === 0);**

**if (divisible) {**

**return multiple;**

**}**

**}**

**}**

**smallestCommons([1, 5]);**

**Déjalo caer**

Dada la matriz arr, itere y elimine cada elemento comenzando desde el primer elemento (el índice 0) hasta que la función funcregrese truecuando el elemento iterado pase a través de él.

Luego, devuelva el resto de la matriz una vez que se cumpla la condición; de lo contrario, arrdebe devolverse como una matriz vacía.

**function dropElements(arr, func) {**

**let sliceIndex = arr.findIndex(func);**

**return arr.slice(sliceIndex >= 0 ? sliceIndex : arr.length);**

**}**

**// test here**

**dropElements([1, 2, 3, 4], function(n) {**

**return n >= 3;**

**});**

**Aplanadora**

Aplane una matriz anidada. Debe tener en cuenta los diferentes niveles de anidamiento.

**function steamrollArray(arr) {**

**const flattenedArray = [];**

**// Loop over array contents**

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

**if (Array.isArray(arr[i])) {**

**// Recursively flatten entries that are arrays**

**// and push into the flattenedArray**

**flattenedArray.push(...steamrollArray(arr[i]));**

**} else {**

**// Copy contents that are not arrays**

**flattenedArray.push(arr[i]);**

**}**

**}**

**return flattenedArray;**

**};**

**// test here**

**steamrollArray([1, [2], [3, [[4]]]]);**

**Agentes binarios**

Devuelve una oración traducida al inglés de la cadena binaria pasada.

La cadena binaria estará separada por espacios.

**function binaryAgent(str) {**

**var biString = str.split(" ");**

**var uniString = [];**

**/\*using the radix (or base) parameter in parseInt, we can convert the binary**

**number to a decimal number while simultaneously converting to a char\*/**

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

**uniString.push(String.fromCharCode(parseInt(biString[i], 2)));**

**}**

**// we then simply join the string**

**return uniString.join("");**

**}**

**// test here**

**binaryAgent(**

**"01000001 01110010 01100101 01101110 00100111 01110100 00100000 01100010 01101111 01101110 01100110 01101001 01110010 01100101 01110011 00100000 01100110 01110101 01101110 00100001 00111111"**

**);**

**todo sea verdad**

Comprueba si el predicado (segundo argumento) es verdadero en todos los elementos de una colección (primer argumento).

En otras palabras, se le proporciona una colección de matriz de objetos. El predicado preserá una propiedad de objeto y deberá devolver truesi su valor es truthy. De lo contrario, regresa false.

En JavaScript, truthylos valores son valores que se traducen truecuando se evalúan en un contexto booleano.

Recuerde, puede acceder a las propiedades de los objetos a través de la notación de puntos o []la notación.

**function truthCheck(collection, pre) {**

**return collection.every(function (element) {**

**return element.hasOwnProperty(pre) && Boolean(element[pre]);**

**});**

**}**

**truthCheck([{ name: "Quincy", role: "Founder", isBot: false }, { name: "Naomi", role: "", isBot: false }, { name: "Camperbot", role: "Bot", isBot: true }], "isBot");**

**Argumentos Opcional**

Cree una función que sume dos argumentos. Si solo se proporciona un argumento, devuelve una función que espera un argumento y devuelve la suma.

Por ejemplo, addTogether(2, 3)debe devolver 5y addTogether(2)debe devolver una función.

Llamar a esta función devuelta con un solo argumento devolverá la suma:

**var sumTwoAnd = addTogether(2);**

**sumTwoAnd(3)regresa 5\_**

Si alguno de los argumentos no es un número válido, devuelve indefinido.

**function addTogether() {**

**const [first, second] = arguments;**

**// First argument is not a number**

**if (typeof(first) !== "number") {**

**return undefined;**

**}**

**// First argument is a number**

**// and second argument is not defined**

**else if (arguments.length === 1) {**

**function addSecond(second) {**

**// New argument is not a number**

**if (typeof(second) !== "number") {**

**return undefined;**

**}**

**// New argument is a number**

**else {**

**return first + second;**

**}**

**}**

**// Note: returning a \*function\***

**return addSecond;**

**}**

**// First argument is a number**

**// and second argument is not a number**

**else if (typeof(second) !== "number") {**

**return undefined;**

**}**

**// First argument is a number**

**// and second argument is a number**

**else {**

**return first + second;**

**}**

**}**

**hacer una persona**

Complete el constructor de objetos con los siguientes métodos a continuación:

**getFirstName()**

**getLastName()**

**getFullName()**

**setFirstName(first)**

**setLastName(last)**

**setFullName(first, last)**

Ejecute las pruebas para ver el resultado esperado para cada método. Estos métodos deben ser los únicos medios disponibles para interactuar con el objeto. Cada prueba declarará una nueva Personinstancia como new Person('Bob', 'Ross').

**const Person = function(first, last) {**

**let firstName = first;**

**let lastName = last;**

**this.getFirstName = function() {**

**return firstName;**

**};**

**this.getLastName = function() {**

**return lastName;**

**};**

**this.getFullName = function() {**

**return this.getFirstName() + " " + this.getLastName();**

**};**

**this.setFirstName = function(first) {**

**return firstName = first;**

**};**

**this.setLastName = function(last) {**

**return lastName = last;**

**};**

**this.setFullName = function(first, last) {**

**this.setFirstName(first);**

**this.setLastName(last);**

**return this.getFullName();**

**};**

**};**

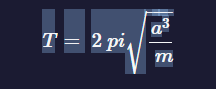
**const bob = new Person("Bob", "Ross");**

**console.log(bob.getFullName());**

**Mapa de los escombros**

De acuerdo con la Tercera Ley de Kepler, el período orbital T

de dos masas puntuales que orbitan entre sí en una órbita circular o elíptica es:



a

es el semieje mayor de la órbita

μ =GRAMOMETRO

es el parámetro gravitacional estándar

GRAMO

es la constante gravitatoria,

METRO

es la masa del cuerpo más masivo.

Devuelve una nueva matriz que transforma la altitud promedio de los elementos en sus períodos orbitales (en segundos).

La matriz contendrá objetos en el formato {name: 'name', avgAlt: avgAlt}.

Los valores deben redondearse al número entero más cercano. El cuerpo orbitado es la Tierra.

El radio de la Tierra es 6367,4447 kilómetros y el valor GM de la Tierra es 398600,4418 km 3 s -2 .