Programación de Algoritmos intermedios:

1.- Suma todos los números de un rango:

function sumAll(arr) {
 	
  const sorted = arr.sort((a, b) => a - b)
  let suma = 0
 	
  for(let i = arr[0]; i <= arr[1]; i++){
  suma += i
 	
  }
  return suma
}
 	
sumAll([1, 4]);
	Te pasaremos un arreglo de dos números. Devuelve la suma de esos dos números más la suma de todos los números dentro del rango. Pueden no estar ordenados.

2.- Diferencia entre dos arreglos:

function diffArray(arr1, arr2) {
 	
  const newArr = []
 	
  for(let i = 0; i < arr1.length; i++){
  if(!arr2.includes(arr1[i])){
  newArr.push(arr1[i])
  }
  }
 	
  for(let i = 0; i < arr2.length; i++){
  if(!arr1.includes(arr2[i])){
  newArr.push(arr2[i])
  }
  }
 	
  return newArr
}
diffArray([1, 2, 3, 6], [1, 2, 3, 4, 5]);
	Compara dos arreglos y devuelve un nuevo arreglo con los elementos que solo se encuentren en uno de los dos arreglos. Es decir, debes devolver la diferencia simétrica.

3.- Busca y destruye:

function destroyer(arr) {
 	
  const newArray = []
 	
  const one = Array.from(arguments)[0]
  const two = Array.from(arguments).slice(1,one.length+1)
	Se proporciona una función destroyer, como argumento recibe un array más un número indeterminado de argumentos. Elimina todos los elementos del array inicial que coincidan con estos argumentos.

for(let i = 0; i < one.length; i++){
 	
if(!two.includes(one[i])){
newArray.push(one[i])
}
 	
}
return newArray
}
destroyer([1, 2, 3, 1, 2, 3], 2, 3);	

4.- Dónde estás:

function whatIsInAName(collection, source) {
 	
const sourceKeys = Object.keys(source)
 	
return collection.filter((person) => {
 	
for (let i = 0; i < sourceKeys.length; i++){
 	
if(!person.hasOwnProperty(sourceKeys[i]) ‖
person[sourceKeys[i]] !== source[sourceKeys[i]]){
return false
}
 	
}
return true
})
 	
}
 	
whatIsInAName([{ first: "Romeo", last: "Montague" }, {
first: "Mercutio", last: null }, { first: "Tybalt", last: "Capulet"
}], { last: "Capulet" });
	Crea una función que busque a través de un arreglo de objetos y devuelva un arreglo de todos los objetos que tengan pares de nombre y valor coincidentes al segundo argumento.

5.- Spinal Case:

function spinalCase(str) {
 	
return str
 	
.trim()
.replace(/[\s_]+/g, "-")
.replace(/([a-z])([A-Z])/g, "$1-$2")
.toLowerCase()
 	
}
spinalCase("TheAndy_Griffith Show");
	Convierte una cadena a spinal case. Spinal case significa todas las palabras en minúscula y unidas por guiones.

La expresión regular /([a-z])([A-Z])/g es una expresión regular en JavaScript que se utiliza para buscar una letra minúscula [a-z] seguida de una letra mayúscula [A-Z] en una cadena de texto. Los paréntesis () se utilizan para capturar cada letra como un grupo separado para poder utilizarlos en el reemplazo.

El segundo parámetro de la función replace() "$1-$2" indica cómo se debe reemplazar la letra minúscula seguida de la letra mayúscula. En este caso, se coloca un guión - entre las dos letras y se las convierte a minúsculas. $1 y $2 corresponden a los dos grupos capturados por los paréntesis en la expresión regular. $1 se refiere a la letra minúscula y $2 se refiere a la letra mayúscula.

6.- Pig Latin (Latin de los cerdos):

function translatePigLatin(str) {
 	
  if(/^[^aeiou]+/i.test(str)){
 	
  const firstMayus = str.match((/^[^aeiou]+/i))
 	
  const a = str
  .replace((/^[^aeiou]+/i), ' ')
  .concat(firstMayus + 'ay')
  .trim()
 	
  return a
 	
  } else {
 	
  const a = str.concat('way')
 	
  return a
  }
 	
}
 	
translatePigLatin("california");
	Pig Latin es una manera de alterar palabras. Si la palabra comienza por consonante, toma el primer grupo de consonantes, muévelo al final y agrega ‘ay’. Si la palabra comienza con vocal, agrega ‘way’ al final.

7.- Busca y Reemplaza:

function myReplace(str, before, after) {
  const result = str
  .split(/\s+/)
 	
  for(let i = 0; i < result.length; i++){
  if(result[i] === before){
 	
  if(before[0] === before[0].toUpperCase()){
  after = after.charAt(0).toUpperCase() +
after.slice(1)
  } else if(after[0] === after[0].toUpperCase()) {
	Busca en la oración y reemplaza usando los argumentos proporcionados y devuelve la nueva oración. El primer argumento es la oración, el segundo el valor a reemplazar y el tercero el valor por el cual se reemplazará el segundo. Respeta la capitalización en la oración original.

after = after.charAt(0).toLowerCase() + after.slice(1) } result[i] = after } } return result.join(' ').trim() } myReplace("A quick brown fox jumped over the lazy dog", "jumped", "Leaped");	

8.- Emparejamiento de ADN:

function pairElement(str) {
const result = [];
 	
for (let i = 0; i < str.length; i++) {
switch (str[i]) {
case 'G':
result.push(['G', 'C']);
break;
case 'C':
result.push(['C', 'G']);
break;
case 'A':
result.push(['A', 'T']);
break;
case 'T':
result.push(['T', 'A']);
break;
default:
console.log(`Invalid character ${str[i]}`);
}
}
 	
return result;
}
 	
pairElement('TTGAG');
	Escribe una función que coincida con los pares de base faltantes para la hebra de ADN proporcionada.

9.- Letras faltantes:

function fearNotLetter(str) {
  for (let i = 0; i < str.length; i++) {
  const charCode = str.charCodeAt(i);
 	
  if (charCode !== str.charCodeAt(0) + i) {
  return String.fromCharCode(charCode - 1);
  }
  }
  return undefined;
}
 	
fearNotLetter("abce");
	Encuentra la letra que falta en la siguiente cadena de letras y devuelvela. Si todas las letras están presentes devuelve undefined.

10.- Unión Ordenada:

//function uniteUnique(...arrays) {return [...new Set(arrays.flat())];}
 	
function uniteUnique(...arrays) {
  // La función toma cualquier número de arrays como argumentos usando el operador de propagación "..." y crea un nuevo array "arrays"
  const result = []; // Crea un nuevo array vacío llamado "result"
  arrays.forEach(array => { // Itera sobre cada array en "arrays" usando el método "forEach"
  array.forEach(element => { // Itera sobre cada elemento en el array usando el método "forEach"
  if (!result.includes(element)) { // Verifica si el elemento no está presente en el array "result" usando el método "includes"
  result.push(element); // Si el elemento no está presente, se agrega al array "result" usando el método "push"
  }
  });
  });
  return result; // Devuelve el array "result" que contiene todos los elementos de los arrays de entrada sin duplicados
}
	Escribe una función que tome como argumento dos o más arreglos y devuelva un arreglo de valores únicos respetando el orden original.

11.- Convierte entidades HTML:

function convertHTML(str) {
let result = []
  for (let i = 0; i < str.length; i++) {
  switch (str[i]) {
  case '&':
  result.push('&');
  break;
  case '<':
  result.push('<');
  break;
  case '>':
  result.push('>');
  break;
  case '"':
  result.push('"');
  break;
  case '\'':
  result.push(''');
  break;
  default:
  result.push(str[i])
  }
  }
  console.log(result.join(""))
  return result.join("")
}
 	
convertHTML("Hamburgers < Pizza < Tacos")
	Convierte los caracteres &, <, >, " (dobles comillas), y ' (apóstrofo), en un cadena con su correspondiente entidad HTML.

12.- Suma todos los números impares de fibonacci:

function sumFibs(num) {
 	
  let acumulator = 0
  let count = 0
 	
  const secuence = [0,1]
 	
  do{
  secuence.push(secuence[count] + secuence[count + 1])
  count++
  } while (secuence[secuence.length-1] <= num)
 	
  console.log(secuence)
 	
  for(let n = 0; secuence[n] <= num; n++){
 	
  if(secuence[n] % 2 !== 0){
  acumulator += secuence[n]
	Dado un entero positivo num, devuelve la suma de todos los numeros impares de fibonacci que son menores o iguales a num.

```
    }
  }
  console.log(acumulator)
  return acumulator
}

sumFibs(1000);
```

13.- Suma todos los números primos:

```
function sumPrimes(num) { // Esta función toma un número 'num' como entrada y devuelve la suma de todos los números primos menores o iguales que 'num'
  let count = 0 // Inicializa una variable 'count' en cero para acumular la suma de los números primos
  function isPrime(n) { // Esta función determina si un número 'n' es primo o no
    if (n <= 1) { // Si 'n' es menor o igual que 1, no es un número primo y se devuelve 'false'
      return false;
    }
    if (n <= 3) { // Si 'n' es menor o igual que 3, es un número primo y se devuelve 'true'
      return true;
    }
    if (n % 2 == 0 || n % 3 == 0) { // Si 'n' es divisible por 2 o 3, no es un número primo y se devuelve 'false'
      return false;
    }
    for (let i = 5; i * i <= n; i += 6) { // Se utiliza un bucle 'for' para verificar si 'n' es divisible por algún número en el rango '5' hasta la raíz cuadrada de 'n', en incrementos de '6'
      if (n % i == 0 || n % (i + 2) == 0) { // Si 'n' es divisible por algún número en el rango mencionado, no es un número primo y se devuelve 'false'
        return false;
      }
    }
    return true; // Si 'n' no es divisible por ningún número en el rango mencionado, es un número primo y se devuelve 'true'
  }
  for(let i = 0; i <= num; i++){ // Se utiliza un bucle 'for' para iterar desde '0' hasta 'num'
    if(isPrime(i)){ // En cada iteración, se llama a la función 'isPrime()' para determinar si el número actual es primo o no
      count += i // Si el número actual es primo, se agrega a la variable acumulada 'count'
    }
  }
  console.log(count) // Imprime la suma total de los números primos
  return count // Devuelve la suma total de los números primos
}
sumPrimes(10); // Llama a la función 'sumPrimes()' con el argumento '10'
```

14.- Mínimo común múltiplo:

function smallestCommons(arr) {
  // Función que verifica si un número es múltiplo común de todos los números en un rango dado
  function isValidMultiple(m, min, max) {
  for (var i = min; i < max; i++) {
  if (m % i !== 0) { // si m no es múltiplo de i
  return false; // entonces m no es múltiplo común y retorna false
  }
  }
  return true; // si m es múltiplo común de todos los números en el rango, retorna true
  }
 	
  var max = Math.max(arr[0], arr[1]); // el número mayor del array de entrada
  var min = Math.min(arr[0], arr[1]); // el número menor del array de entrada
  var multiple = max; // inicializa la variable multiple con el valor del número mayor
 	
  // Mientras no se encuentre un múltiplo común, se incrementa el valor de la variable multiple por el valor del número mayor
  // hasta que se encuentre un múltiplo común utilizando la función isValidMultiple
  while (!isValidMultiple(multiple, min, max)) {
  multiple += max;
  }
 	
  return multiple; // retorna el número más pequeño que es múltiplo común de los dos números en el rango de los dos números dados
}
	Encuentra el múltiplo más bajo de los parámetros proporcionados que pueden dividirse equitativamente por ambos, así como por todos los números consecutivos del rango entre esos parámetros.

15.- Dejalo caer:

function dropElements(arr, func) {
  // Esta línea define una función llamada "dropElements" que toma dos parámetros, "arr" y "func".
 	
  while (arr.length > 0 && !func(arr[0])) {
  // Este bucle "while" se ejecutará mientras la longitud del array "arr" sea mayor que 0
  // y el primer elemento del array no cumpla con la función "func".
  // La condición en el bucle utiliza el operador && (y lógico) para evaluar ambas condiciones.
  // La primera condición verifica si la longitud del array es mayor que 0.
	Dado el arreglo arr, itera y elimina cada elemento comenzando desde el primer elemento (indice 0) hasta que la función func() devuelva true cuando el elemento iterado se pasa a través de él. Luego devuelve el resto del arreglo una vez que se cumpla la condición, de lo contrario, arr debe devolverse como un arreglo vacío.

// La segunda condición utiliza el operador ! (negación lógica) para evaluar si el resultado de la función "func"
// para el primer elemento del array es falso (es decir, no cumple con la condición).
 	
arr.shift();
// Esta línea elimina el primer elemento del array "arr" utilizando el método "shift()".
// Esto se repite en cada iteración del bucle hasta que se cumpla la condición del bucle.
 	
}
 	
return arr;
// Esta línea devuelve el array "arr" después de haber eliminado todos los elementos que no cumplen con la condición.
}	

16.- Aplanadora:

function steamrollArray(arr) {
// Define una función llamada "steamrollArray" que toma un parámetro, "arr".
 	
let flattenedArray = []
// Crea una variable "flattenedArray" que se inicializa como un array vacío.
 	
while(arr.length){
// Inicia un bucle "while" que se repetirá mientras "arr" tenga elementos.
 	
const currentElement = arr.shift()
// Extrae el primer elemento de "arr" y lo almacena en la variable "currentElement".
// También elimina el elemento del array "arr" utilizando el método "shift()".
console.log(currentElement)
 	
if(Array.isArray(currentElement)){
// Comprueba si "currentElement" es un array utilizando la función "Array.isArray()".
 	
arr = currentElement.concat(arr)
// Si "currentElement" es un array, utiliza el método "concat()" para unir los elementos de "currentElement"
// con los elementos restantes en "arr" y asigna el resultado a "arr".
console.log('Is Array')
 	
}
	Aplana un arreglo anidado. Debes tener en cuenta los diferentes niveles de anidación.

else {
flattenedArray.push(currentElement)
// Si "currentElement" no es un array, agrega el elemento a "flattenedArray" utilizando el método "push()".
}
console.log(flattenedArray)
// Imprime el valor actual del array "flattenedArray".
}
 	
return flattenedArray
// Devuelve el array "flattenedArray" que contiene todos los elementos aplanados.
}
 	
steamrollArray([1, [2], [3, [[4]]]]);
// Llama a la función "steamrollArray" con un ejemplo de arreglo multinivel para aplanarlo.	

17.- Agentes binarios:

function binaryAgent(str) {
const result = [] // Crea un array vacío llamado "result"
const myArray = str.split(' ') // Divide el string "str" en un array de strings, cada uno separado por un espacio. Asigna este array a una variable llamada "myArray".
for(let i = 0; i < myArray.length; i ++){ // Recorre cada elemento del array "myArray"
myArray[i] = parseInt(myArray[i], 2) // Convierte cada elemento de "myArray" de binario a decimal y lo asigna de nuevo a "myArray".
result.push(String.fromCharCode(myArray[i])) // Convierte cada elemento de "myArray" de decimal a su equivalente carácter ASCII y lo agrega al array "result".
console.log(String.fromCharCode(myArray[i])) // Imprime en la consola el carácter ASCII correspondiente al valor decimal actual.
}
console.log(result.toString().replace(/,/g, '')) // Imprime en la consola el string resultante de unir todos los caracteres del array "result" en un solo string y eliminar todas las comas.
return result.toString().replace(/,/g, '') // Devuelve el mismo string que se imprimió en la consola, sin las comas.
}
	Devuelve una frase traducida al inglés de una cadena binaria pasada.

18.- Todo sea verdad:

function truthCheck(collection, pre) {
  let result = 0; // Inicializar el contador a cero
 	
  collection.forEach((user) => // Iterar sobre cada objeto en la colección
  user[pre] // Si la propiedad pre en el objeto es verdadera
  ? result += 1 // Incrementar el contador en 1
  : result -= 1 // Si no, decrementar el contador en 1
  );
 	
  if(result == collection.length){ // Si el contador es igual al tamaño de la colección
  return true; // Devolver verdadero
  } else {
  return false; // De lo contrario, devolver falso
  }
}
	Comprueba si el predicado (segundo argumento) es truthy en todos los elementos de una colección (primer argumento).

19.- Argumentos opcionales:

// Definición de la función 'addTogether' que acepta un número variable de argumentos
function addTogether(...args) {
 	
  // Desestructuración de los primeros dos argumentos de 'args' en las variables 'first' y 'second'
  const [first, second] = args
 	
  // Si se proporciona un solo argumento y es un número, devuelve una función que espera un segundo número y devuelve la suma de ambos
  if(args.length === 1 && typeof first === 'number'){
  return num => {
  // Si el segundo argumento es un número, devuelve la suma de 'first' y 'num'
  if(typeof num === 'number'){
  return first + num
  }
  }
  }
  // Si se proporcionan dos argumentos y ambos son números, devuelve la suma de ambos
  if(typeof first === 'number' && typeof second === 'number'){
  return first + second
  }
  // En cualquier otro caso, no se puede sumar los argumentos, así que no se devuelve nada
}
	Crea una función que sume dos argumentos. Si solo se proporciona un argumento, entonces devuelve una función que espere un argumento y devuelve la suma.

20.- Crea una persona:

// Define la función constructora 'Person' que acepta un parámetro 'firstAndLast'
const Person = function(firstAndLast) {
// Crea una variable local 'fullName' y asigna el parámetro 'firstAndLast'
let fullName = firstAndLast;
 	
// Define el método 'getFirstName' que devuelve el primer nombre de 'fullName'
this.getFirstName = function() {
return fullName.split(" ")[0];
};
 	
// Define el método 'getLastName' que devuelve el apellido de 'fullName'
this.getLastName = function() {
return fullName.split(" ")[1];
};
 	
// Define el método 'getFullName' que devuelve 'fullName'
this.getFullName = function() {
return fullName;
};
 	
// Define el método 'setFirstName' que actualiza el primer nombre de 'fullName'
this.setFirstName = function(name) {
fullName = name + " " + fullName.split(" ")[1];
};
 	
// Define el método 'setLastName' que actualiza el apellido de 'fullName'
this.setLastName = function(name) {
fullName = fullName.split(" ")[0] + " " + name;
};
 	
// Define el método 'setFullName' que actualiza 'fullName'
this.setFullName = function(name) {
fullName = name;
};
};
 	
const bob = new Person("Bob Ross");
console.log(bob.getFullName());
	Completa el constructor de objetos con los siguientes métodos.

21.- Mapea el Drevis:

// Definición de una función llamada "orbitalPeriod" que toma un arreglo "arr" como parámetro
function orbitalPeriod(arr) {
  // Declaración de constantes "GM", "earthRadius" y "a"
  const GM = 398600.4418;
  const earthRadius = 6367.4447;
  const a = 2 * Math.PI;
  // Declaración de un nuevo arreglo vacío llamado "newArr"
  const newArr = [];
 	
  // Declaración de una función interna llamada "getOrbPeriod" que toma un objeto "obj" como parámetro
  const getOrbPeriod = function(obj) {
  // Cálculo de la constante "c"
  const c = Math.pow(earthRadius + obj.avgAlt, 3);
  // Cálculo de la constante "b"
  const b = Math.sqrt(c / GM);
  // Cálculo del período orbital "orbPeriod"
  const orbPeriod = Math.round(a * b);
  // Creación de un nuevo objeto con el nombre del satélite y el período orbital calculado
  return {name: obj.name, orbitalPeriod: orbPeriod};
  };
 	
  // Bucle "for...in" que itera sobre cada elemento "elem" del arreglo "arr"
  for (let elem in arr) {
  // Llamada a la función "getOrbPeriod" con el elemento "elem" actual del arreglo "arr" y agregando el resultado al nuevo arreglo "newArr"
  newArr.push(getOrbPeriod(arr[elem]));
  }
 	
  // Retorno del nuevo arreglo "newArr" con los objetos que contienen el nombre del satélite y el período orbital calculado
  return newArr;
}
	De acuerdo con la tercera Ley de Kepler, el periodo orbital T de dos puntos se orbitan mutuamente en una órbita circular o elíptica es: (Fórmula 3 Ley Keppler) Devuelve un nuevo arreglo que transforma la altitud media de los elementos en sus periodos orbitales (en segundos).