Estructuras de datos básicas.

1.- Utiliza un arreglo para almacenar una colección de datos:

let yourArray = ["Hello", "Bye", 8, 90, true];	Hemos definido una variable llamada yourArray. Completa la sentencia asignando un arreglo de al menos 5 elementos de longitud. Tu arreglo debe contener al menos un string, un number y un boolean.

2.- Accede a los contenidos de un arreglo utilizando la notación de corchetes:

let myArray = ["a", "b", "c", "d"];
myArray[1] = "not B";
console.log(myArray);
	Para completar este desafío, establece la segunda posición (índice 1) de myArray a cualquier cosa que quieras, además de la letra b.

3.- Agrega elementos a un arreglo con push() y unshift():

function mixedNumbers(arr) {
  arr.unshift('I', 2, 'three');
  arr.push(7, 'VIII', 9);
  return arr;
}
console.log(mixedNumbers(['IV', 5, 'six']));
	Hemos definido una función, mixedNumber, a la cual le estamos pasando un arreglo como argumento. Modifica la función utilizando push() y unshift() para agregar 'I', 2, 'three' al principio y 7, 'VIII', 9 al final.

4.- Elimina elementos de un arreglo con pop() y shift():

function popShift(arr) {
  let popped = arr.pop();
  let shifted = arr.shift();
  return [shifted, popped];
}
console.log(popShift(['challenge', 'is', 'not', 'complete']));
	Hemos definido una función, popShift, la cual toma un arreglo como argumento y devuelve un nuevo arreglo. Modifica la función con pop() y shift(), para eliminar el primer y último elemento del arreglo y asignar los elementos eliminados a sus correspondientes variables.

5.-Elimina elementos usando splice():

Splice() admite hasta 3 parámetros, pero nos centraremos en los dos primeros, que representan el índice desde el que empieza y el número de elementos que . Splice() eliminará

const arr = [2, 4, 5, 1, 7, 5, 2, 1];
arr.splice(1, 4);
 	
console.log(arr);
	Hemos inicializado un arreglo arr. Usa splice() para eliminar elementos de arr, de forma que sólo contenga elementos que sumen el valor de 10.

6.- Agrega elementos usando splice():

El tercer parámetro de splice() está compuesto por uno o varios elementos que se agregaran al arreglo.

function htmlColorNames(arr) {
  arr.splice(0,2,'DarkSalmon', 'BlanchedAlmond');
  return arr;
}
console.log(htmlColorNames(['DarkGolden Rod', 'WhiteSmoke', 'LavenderBlush', 'PaleTurquoise', 'FireBrick']));
	Hemos definido una función, htmlColorNames, que toma un arreglo de colores HTML como argumento. Modifica la función usando splice() para eliminar los dos primeros elementos del arreglo y agrega 'DarkSalmon' y 'BlanchedAlmond' en sus respectivos lugares.

7.- Copia elementos de un arreglo usando slice():

En lugar de modificar un arreglo, slice(), copia o extrae un número determinado de elementos a un nuevo arreglo, dejando intacto el arreglo al que llama. slice() toma dos parámetros: el primero es el índice en que se inicia la extracción, y el segundo el índice en el que se detiene la extracción (sin incluirlo).

function forecast(arr) {
  return arr = arr.slice(2,4);
}
 	
console.log(forecast(['cold', 'rainy', 'warm', 'sunny', 'cool', 'thunderstorms']));
	Hemos definido una función, forecast, que toma un arreglo como argumento. Modifica la función usando slice() para extraer información del arreglo de argumentos y devuelve un nuevo arreglo que contenga los elementos warm y sunny.

8.- Copia un arreglo con el operador de propagación:

El operador de propagación (...) nos permite copiar fácilmente todos los elementos de un arreglo, en orden, con una sintaxis simple y altamente legible.

function copyMachine(arr, num) {
  let newArr = [];
  while (num >= 1) {
  newArr.push([...arr]);
  num--;
  }
  return newArr;
}
 	
console.log(copyMachine([true, false, true], 2));
	Hemos definido una función que toma un array y un número como argumentos. Se supone que la función devuelve un nuevo arreglo compuesto por un número de copias del arreglo. Modifica la función utilizando el operador de propagación para que funcione correctamente.

9.- Combina arreglos con el operador de propagación:

Otra gran ventaja del operador de propagación es la capacidad de combinar arreglos, o insertar todos los elementos de un arreglo en otro, en cualquier índice.

function spreadOut() {
  let fragment = ['to', 'code'];
  let sentence = ['learning',...fragment, 'is', 'fun'];
  return sentence;
}
console.log(spreadOut());
	Hemos definido una función spreadOut que devuelve la variable sentence. Modifica la función usando el operador de propagación para que devuelva el arreglo ['learning', 'to', 'code', 'is', 'fun'].

10.- Comprueba la presencia de un elemento con .indexOf():

.indexOf nos permite comprobar rápida y fácilmente la presencia de un elemento en un arreglo. Toma un elemento como parámetro, y cuando lo llama, devuelve la posición o -1 si no lo encuentra.

function quickCheck(arr, elem) {
 if(arr.indexOf(elem) == -1){
  return false;
 } else {
  return true
  }
}
 	
console.log(quickCheck(['squash', 'onions', 'shallots'], 'mushrooms'));
	Hemos definido una función, quickCheck, que toma un arreglo y un elemento como argumentos. Modifica la función usando indexOf() para que devuelva true si el elemento existe y false si no.

11.- Itera a través de todos los elementos de un arreglo utilizando bucles “for”:

A veces, cuando se trabaja con arreglos, es muy útil poder iterar a través de cada elemento para encontrar uno o más elementos que podamos necesitar, o manipular un arreglo en función de los elementos de datos que cumplen un determinado conjunto de criterios.

JavaScript ofrece varios métodos incorporados que iteran sobre arreglos de formas ligeramente diferentes para conseguir distintos resultados (como .every() .forEach() .map()) Sin embargo la técnica más flexible y que más control nos ofrece es un simple bucle for.

function filteredArray(arr, elem) {
  let newArr = [];
 	
  for (let i = 0; i < arr.length; i++){
  if(arr[i].indexOf(elem) == -1){
  newArr.push(arr[i]);
  }
  }
	Hemos definido una función que toma dos argumentos. El segundo argumento no puede estar presente en el primero (que será un array anidado). Modifica la función usando un bucle for para que devuelva una versión filtrada del arreglo pasado. De forma que cualquier arreglo de arr que contenga elem haya sido eliminado.

return newArr;
}
console.log(filteredArray([[3, 2, 3], [1, 6, 3], [3, 13, 26], [19, 3, 9]], 3));	

12.- Crea arreglos complejos multidimensionales:

let myNestedArray = [['unshift', false, 1, 2, 3, 'complex', 'nested'], ['loop', 'shift', 6, 7, 1000, 'method'], ['concat', false, true, 'spread', 'array'], ['mutate', 1327.98, 'splice', 'slice', 'push'], ['iterate', 1.3849, 7, '8.4876', 'arbitrary', 'depth'], [['deep']], [[['deeper']]],[[[['deepest']]]]]; console.log(myNestedArray);	Modifica los datos del arreglo para que tenga 5 niveles de profundidad. Inserta en el 3º deep, en el 4º deeper y en el 5º deepest.

13.- Agrega pares clave-valor a objetos de JavaScript:

En su aspecto más básico los objetos no son más que colecciones de clave-valor. En otras palabras, son piezas de datos (valores) asignadas a identificadores únicos llamados propiedades (clave).

let foods = {
apples: 25,
oranges: 32,
plums: 28
};
 	
foods.bananas = 13;
foods.grapes = 35;
foods.strawberries = 27;
 	
console.log(foods);
	Agrega más entradas al objeto food: bananas con el valor 13, grapes con el valor 35, y strawberries con el valor de 27.

14.- Modifica un objeto anidado dentro de un objeto:

let userActivity = {
id: 23894201352,
date: 'January 1, 2017',
data: {
totalUsers: 51,
online: 42
}
};
userActivity.data.online = 45;
console.log(userActivity);
	Aquí hemos definido un objeto userActivity, que incluye otro objeto anidado dentro de él. Establece el valor de la clave online en 45.

15.- Accede a los nombres de propiedad con la notación de corchetes:

let foods = {
  apples: 25,
  oranges: 32,
  plums: 28,
  bananas: 13,
  grapes: 35,
  strawberries: 27
};
function checkInventory(scannedItem) {
  return foods[scannedItem];
}
console.log(checkInventory("apples"));
	Hemos definido una función que recibe como argumento un elemento escaneado. Devuelve el valor actual de la clave scannedItem en el objeto foods.

16.- Usa la palabra clave “delete” para eliminar las propiedades de los objetos:

let foods = {
  apples: 25,
  oranges: 32,
  plums: 28,
  bananas: 13,
  grapes: 35,
  strawberries: 27
};
delete foods.oranges;
delete foods.plums;
delete foods.strawberries;
console.log(foods);
	Usa la palabra clave delete para eliminar las claves oranges, plums y strawberries del objeto foods.

17.- Evalúa si un objeto tiene una propiedad:

let users = {
  Alan: {age: 27, online: true},
  Jeff: {age: 32, online: true},
  Sarah: {age: 48, online: true},
  Ryan: {age: 19, online: true}
};
 	
function isEveryoneHere(userObj) {
 return userObj.hasOwnProperty("Alan") &&
  userObj.hasOwnProperty("Jeff") &&
  userObj.hasOwnProperty("Sarah") &&
  userObj.hasOwnProperty("Ryan");
}
console.log(countOnline(users));
	Termina de escribir la función para que devuelva true si el objeto pasado contiene los cuatro nombres y devuelve false en caso contrario.

18.- Itera a través de las claves de un objeto con una sentencia “for … in”;

const users = {
  Alan: {online: false},
  Jeff: {online: true },
  Sarah: {online: false}
}
function countOnline(usersObj) {
  let result = 0;
  for(let user in usersObj){
  if(usersObj[user].online == true){
  result++
  }
  }
  return result;
}
console.log(countOnline(users));
	Utiliza for … in para devolver el número de usuarios que este online.

19.-Genera un arreglo de todas las claves de objetos con Object.keys():

Podemos generar un arreglo que contenga todas las claves almacenadas en un objeto, utilizando el método Object.keys(). Toma un objeto como argumento y devuelve un arreglo de cadenas que representa cada propiedad del objeto. De nuevo, no habrá un orden específico para las entradas del arreglo.

function getArrayOfUsers(obj) {
  return Object.keys(obj);
}
	Termina de escribir la función getArrayOfUsers para que devuelva un arreglo que contenga todas las propiedades del objeto que recibe como argumento.

20.- Modifica un arreglo almacenado en un objeto:

function addFriend(userObj, friend) {
  user.data.friends.push(friend);
  return user.data.friends;
  }
	Termina de escribir la función addFriend para que tome user y agregue el nombre del argumento friend al arreglo almacenado en user.data.friends y devuelvelo.