

3.- JavaScript : ARRAYS y OBJETOS

RECORDAMOS:

JS es **multi-paradigma** es decir que nos da libertad para elegir el tipo de programación que deseamos realizar, ya sea Orientada a Objetos, Programación estructurada, Programación funcional que es uno de los paradigmas más usados actualmente, o incluso programar como “te dé la gana” que no suele ser lo mejor.

Básicamente en **la FP (Functional programming)** todo o casi todo son funciones a las cuales les pasamos unos parámetros con los cuales trabaja y devuelve un resultado, dicho resultado devuelto solo depende de los valores que les pasemos como parámetros y NUNCA deben depender de variables globales que guarden el estado actual de nuestra aplicación. Por eso entre otras cosas debemos **evitar mutar** las variables es decir añadir un nuevo elemento a un array o una nueva propiedad a un objeto porque entonces estamos modificando variables globales, en vez de hacer eso debemos devolver copias de los Array u Objetos con los nuevos elementos ya añadidos.

RECOMIENDO LEER:

<https://www.freecodecamp.org/espanol/news/programacion-funcional-en-javascript-explicado/>

ARRAYS

Ejemplos de cómo añadir elementos a un array con y sin mutaciones (recordar siempre que es mejor sin mutaciones), es decir creando copias del array sobretodo en react y para respetar el paradigma de la programación funcional.

```
const personas = [  
  {  
    nombre: "Juan",  
    apellidos: "García Pérez",  
    dni: "12345678A",  
    edad: 35,  
  },  
  {  
    nombre: "María",  
    apellidos: "López Martínez",  
    dni: "23456789B",  
    edad: 28,  
  },  
]
```

3.- JavaScript : ARRAYS y OBJETOS

```
{
  nombre: "Carlos",
  apellidos: "Fernández González",
  dni: "34567890C",
  edad: 42,
},
{
  nombre: "Laura",
  apellidos: "Martín Sánchez",
  dni: "45678901D",
  edad: 31,
},
{
  nombre: "Ana",
  apellidos: "Díaz Rodríguez",
  dni: "56789012E",
  edad: 29,
},
{
  nombre: "David",
  apellidos: "Ruiz García",
  dni: "67890123F",
  edad: 39,
},
{
  nombre: "Elena",
  apellidos: "Pérez Martínez",
  dni: "78901234G",
  edad: 25,
},
{
  nombre: "Miguel",
  apellidos: "Sánchez López",
  dni: "89012345H",
  edad: 45,
},
{
  nombre: "Sara",
  apellidos: "González Rodríguez",
```

3.- JavaScript : ARRAYS y OBJETOS

```
dni: "90123456I",
edad: 33,
},
{
  nombre: "Javier",
  apellidos: "Martínez Fernández",
  dni: "01234567J",
  edad: 37,
},
{
  nombre: "Paula",
  apellidos: "Gómez García",
  dni: "12345678K",
  edad: 26,
},
{
  nombre: "Adrián",
  apellidos: "Fernández Martínez",
  dni: "23456789L",
  edad: 40,
},
{
  nombre: "Eva",
  apellidos: "López Sánchez",
  dni: "34567890M",
  edad: 30,
},
{
  nombre: "Alberto",
  apellidos: "García Rodríguez",
  dni: "45678901N",
  edad: 38,
},
{
  nombre: "Cristina",
  apellidos: "Martínez Pérez",
  dni: "56789012O",
  edad: 27,
},
},
```

3.- JavaScript : ARRAYS y OBJETOS

```
{
  nombre: "Luis",
  apellidos: "Rodríguez López",
  dni: "67890123P",
  edad: 36,
},
{
  nombre: "Natalia",
  apellidos: "Sánchez Martín",
  dni: "78901234Q",
  edad: 32,
},
{
  nombre: "Pablo",
  apellidos: "Martín González",
  dni: "89012345R",
  edad: 41,
},
{
  nombre: "Lucía",
  apellidos: "González Pérez",
  dni: "90123456S",
  edad: 34,
},
{
  nombre: "Diego",
  apellidos: "Pérez Rodríguez",
  dni: "01234567T",
  edad: 28,
},
],
```

Ejemplo de array de objetos

```
console.log("array original");
console.log(personas);
```

Lo visualizamos

Añadir una persona mutando

3.- JavaScript : ARRAYS y OBJETOS

Es decir que añadimos un nuevo elemento al array personas no creamos una copia del mismo sino que añadimos un nuevo elemento, mutando o dicho de otro forma cambiando el valor del array original, lo cual puede causar problemas.

```
personas.push({  
  nombre: "Pablo",  
  apellidos: "Martín González",  
  dni: "89012345R",  
  edad: 41,  
});  
console.log("array mutado");  
console.log(personas);
```

MÁS TEORÍA

Los tipos primitivos (números, booleanos, cadenas de texto, null y undefined) son inmutables y no se pueden mutar, puede parecer que estamos mutando un tipo primitivo pero en verdad no lo haces, si te has fijado una cadena de textos tiene muchos métodos como son `.charAt`, `.charCodeAt`, `.substr`, `.substring` pero ninguno muta la cadena (cambia el valor), sino que todos devuelven una cadena nueva. Asignar un nuevo valor a una variable no es mutarla (es difícil de ver la diferencia).

Los objetos (arrays, objetos, funciones, etc.) son mutables y se pueden mutar es decir que podemos cambiar valores sin crear una copia, podemos añadir un elemento nuevo a un array... sin cambiar el mismo array (mutándolo) lo cual no suele ser una buena práctica y es mejor crear una copia del mismo con un elemento más y en el peor de los casos darle a esa copia el mismo nombre de la variable original.

Si no dominas las cadenas de caracteres, repasa el siguiente enlace y realiza los siguientes ejercicios

https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Global_Objects/String

1. **Convertir una cadena a mayúsculas:** Escribe una función que reciba una cadena y la convierta a mayúsculas.
2. **Repetir una cadena tantas veces como el número pasado como parámetro:** Escribe una función que tome una cadena y la repita tantas veces como el número pasado como parámetro.

3.- JavaScript : ARRAYS y OBJETOS

3. **Obtener el carácter en una posición específica:** Escribe una función que reciba una cadena y un índice, y devuelva el carácter en esa posición.
4. **Comprobar si una cadena contiene otra subcadena:** Escribe una función que reciba dos cadenas y verifique si la primera contiene a la segunda. Si eres capaz podrías hacer que en vez de devolver un true o false si se encuentra la cadena, que devuelva el número de veces que la contiene.
5. **Verificar si una cadena es un palíndromo:** Escribe una función que verifique si una cadena es la misma al derecho y al revés. (ya lo hemos hecho, pero vuelve a hacerlo)
6. **Contar la cantidad de palabras:** Escribe una función que reciba una cadena y devuelva la cantidad de palabras que contiene.
7. **Reemplazar un carácter en una cadena:** Escribe una función que reemplace todas las ocurrencias de un carácter específico en una cadena con otro carácter.
8. **Eliminar espacios en blanco:** Escribe una función que elimine todos los espacios en blanco de una cadena.
9. **Convertir la primera letra de cada palabra en mayúscula:** Escribe una función que convierta la primera letra de cada palabra en mayúscula en una cadena.
10. **Extraer una subcadena:** Escribe una función que reciba una cadena y dos índices, y devuelva la subcadena entre esos índices pero sin usar funciones como substr o substring, es decir recorre la cadena carácter a carácter.
11. **Contar mayúsculas, minúsculas, números y otros caracteres que contiene una cadena:** Escribe una función que cuente cuantas mayúsculas, minúsculas... contiene una cadena y devuelva el resultado en un array
12. **Haz dos funciones, una que encripte una cadena y otra que la desencripte.** Para encriptar puedes sumarle una semilla al código ASCII y para desencriptarla restarle la semilla. Usa charCodeAt y fromCharCode
13. **Encontrar la palabra más larga:** Escribe una función que encuentre la palabra más larga en una cadena.
14. **Comprobar si una cadena comienza con un prefijo dado:** Escribe una función que compruebe si una cadena comienza con un prefijo dado.

IMPORTANTE: No uses chatGPT porque lo que suele hacer es usar las funciones o métodos de los ARRAYS y no los de las cadenas. Revisa el enlace y usa los métodos propios de las cadenas que además te aseguran que no estás mutando la cadena.

3.- JavaScript : ARRAYS y OBJETOS

Hay varias razones por las que uno podría querer evitar la mutación en la codificación JavaScript:

- La mutación puede hacer que el código sea más difícil de razonar y depurar, ya que los cambios en el valor de una variable pueden tener efectos secundarios no deseados en otras partes del código.
- La mutación puede generar resultados inesperados en la programación funcional, donde el objetivo es escribir código que sea predecible y tenga efectos secundarios mínimos.
- La mutación puede dificultar la prueba del código, ya que el estado de una variable puede cambiar inesperadamente durante el transcurso de una prueba.
- La mutación puede provocar condiciones de carrera y otros problemas de concurrencia cuando se trabaja con un estado compartido.

Vale la pena señalar que la mutación no siempre es mala y puede usarse en el contexto correcto. Es importante comprender cuándo y dónde usarlo y cuándo evitarlo, según los requisitos de la aplicación, por ejemplo si tuviésemos una array con 100 millones de elementos, es mejor mutarlo para no producir problemas de rendimiento y de memoria.

En REACT y otros Framework la mutación da muchos problemas porque dichas mutaciones producen cambios en los estados de los COMPONENTES que puede que el framework no detecte y no renderize de nuevo el componente. Por ejemplo react renderiza un componente cada vez que se le pasa un props (parametro) nuevo y al mutar el props puede que no lo detecte como nuevo... Ya se verá más adelante.

Añadir una nueva persona mutando el array con PUSH (sigue siendo el mismo array). No es buena práctica

```
personas.push({
  nombre: "Pablo",
  apellidos: "Martín González",
  dni: "89012345R",
  edad: 41,
});
console.log("array mutado");
console.log(personas);
```

3.- JavaScript : ARRAYS y OBJETOS

Ahora lo hacemos sin mutar con CONCAT

```
let personas2 = personas.concat([
  {
    nombre: "Pablo",
    apellidos: "Martín González",
    dni: "89012345R",
    edad: 41,
  },
]);

console.log("array sin mutado personas 2 (concat");
console.log(personas2);
```

Concat devuelve un array nuevo resultante de la concatenación de dos Arrays. Si quisiésemos podríamos haber asignado el nuevo array a la misma variable con

```
personas = personas.concat
```

Añadir una nueva persona mutando el array con **SPREAD**

Es lo más usado, ya que es muy simple, legible y funciona también con objetos

```
const personas3 = [
  ...personas,
  {
    nombre: "Pablo",
    apellidos: "Martín González",
    dni: "89012345R",
    edad: 41,
  },
];

console.log("array sin mutado personas 3 (spread)");
console.log(personas3);
```

3.- JavaScript : ARRAYS y OBJETOS

Lo que hace un SPREAD es como sacar los datos de array, como si los escribieses a mano y se pueden usar en funciones que no admiten arrays como parámetro.

```
const numeros = [5, 2, 8, 1, 7, 4, 9, 6, 3, 10];  
console.log(Math.max(...numeros))
```

FUNCIONES QUE PODEMOS USAR CON ARRAYS

REFERENCIA:

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array

Dado el array:

```
let numeros = [1, 2, 3, 4, 5, 6];
```

Recorrer el array con un for típico y corriente sumando los elementos

```
let suma = 0;  
for (let x = 0; x < numeros.length; x++) suma += numeros[x];  
console.log("la suma es igual " + suma);
```

Ahora una modificación del for, for of (parecido al **for x in lista** de Python)

```
suma = 0;  
for (let x of numeros) suma += x;  
console.log("la suma es igual con for of " + suma);
```

Y ahora con un forEach

```
suma = 0;  
numeros.forEach((elemento) => {  
    suma = suma + elemento;  
});  
console.log("la suma es igual con forEach " + suma);
```

3.- JavaScript : ARRAYS y OBJETOS

MAPEAR UN ARRAY

Es lo mismo que recorrer el array elemento por elemento y devolver un nuevo array transformado, ejemplo doble de un array el return va explícito al no poner las {}

```
let nuevo = numeros.map((elemento) => elemento * 2);  
console.log(nuevo);
```

También puedes coger el índice como segundo parámetro. En la mayoría de los métodos de los arrays que puedas recoger el elemento, también se puede recoger el índice (MAP, FOREACH...)

```
let nuevo2 = numeros.map((elemento, index) => elemento * index);  
console.log(nuevo2);
```

FILTRAR

Devuelve un nuevo array para los elementos que la condición es true

```
let nuevo3 = numeros.filter((elemento) => elemento > 3);  
console.log(nuevo3);
```

EVERY

Comprueba si todos los elementos cumplen una condición, en este caso es false porque no todos los elementos son mayores que 3

```
let verdad = numeros.every((elemento) => elemento > 3);  
console.log(verdad);  
  
//en este caso es true porque todos valen menos de 10  
verdad = numeros.every((elemento) => elemento < 10);  
console.log(verdad);
```

Pasar cadena a array con SPLIT

```
let cad = "123456";  
let cadArray = cad.split("");
```

3.- JavaScript : ARRAYS y OBJETOS

```
console.log(cadArray);
```

El array devuelto en el caso anterior es de cadenas, pero podemos pasarlo a números

```
let numArray = cadArray.map((element) => parseInt(element));  
console.log(numArray);
```

Sacar solo los pares

```
numArray = numArray.filter((e) => e % 2 == 0);  
console.log(numArray);
```

FORMAS DE VOLVER A PASARLO A CADENA

```
cad = numArray.toString();  
console.log(cad);  
//Join te permite elegir el separador de los elementos del array  
cad = numArray.join("-");  
console.log(cad);  
cad = numArray.join("");  
console.log(cad);
```

SUMA CON REDUCE

```
suma = numeros.reduce((acumulador, actual) => acumulador + actual);  
console.log("suma =" + suma);
```

EJERCICIO:

Haz un programa que elimine un elemento de un array (pista: PUEDES USAR SLICE)

OBJETOS

Tres formas de acceder a las propiedades

```
const user = {  
  //propiedades
```

3.- JavaScript : ARRAYS y OBJETOS

```
name: 'Paco',
age: 25,
email: 'paco@examplepetstore.com',
password: '123456',
address: {
  street: '123 Main St',
  city: 'New York',
  state: 'NY',
  zip: '10001'
},
//metodos
sendmail() {
  console.log('Sending mail...');
}
};

//FORMAS DE ACCEDER A LAS PROPIEDADES
console.log(user.age);
console.log(user['age']);
let propiedad = 'age';
console.log(user[propiedad]);

//USAR UN METODO
user.sendmail();
```

DESTRUCTURING (de objetos)

Nos permite escribir menos, solo cogemos como parámetros de la función las propiedades del objeto que nos interesan

```
//Paso el objeto entero
function printUser(user) {
  console.log(user.name);
  console.log(user.age);
}
printUser(user);

//usando destructuring
function printUser({name, age}) {
  console.log(name);
  console.log(age);
}
```

3.- JavaScript : ARRAYS y OBJETOS

```
}  
printUser(user);
```

FUNCIONES QUE DEVUELVEN ARRAYS u OBJETOS (con función flecha)

```
//Con los arrays no hay problemas con el return implicito  
//porque empiezan por []  
const showArray = () => [1,2,3];  
console.log(showArray());  
  
//con los objetos si hay problema porque empiezan por {} que se  
//confunden con el inicio de la función ya que las llaves son  
//el inicio de una función y de un objeto y queda AMBIGUO  
const showObjet = () => ({name: 'Paco', age: 25});  
console.log(showObjet());
```

LO COMPLICAMOS UN POCO...

Tenemos dos Array, el de pokemon donde sale el nombre del pokemon al que le han hecho daño y el array daño donde sale la cantidad de daño que le han hecho al pokemon que está en esa misma posición.

Ejemplo el primero es Pikachu y le han hecho 10 de daño, el segundo es Bulbasur y le han hecho 20 de daño

Datos iniciales

```
let pokemon = [  
  "pikachu",  
  "bulbasur",  
  "chadmander",  
  "pikachu",  
  "pikachu",  
  "chadmander",  
];  
let daño = [10, 20, 5, 10, 10, 5];
```

QUEREMOS SABER CUANTO DAÑO LE HAN HECHO A CADA POKEMON

3.- JavaScript : ARRAYS y OBJETOS

Primera forma con un objeto al que le vamos a ir añadiendo propiedades con el nombre de cada pokemon y su daño

```
let pokeda = {};  
pokemon.forEach((p, indice) => {  
  //si no existe la propiedad con el nombre del pokemon la creo con valor  
  0  
  if (pokeda[p] === undefined) pokeda[p] = 0;  
  // a la propiedad con el nombre del pokemon le sumo el daño  
  pokeda[p] += daño[indice];  
});  
  
console.log(pokeda);
```

LO MISMO CON EL OBJETO CLAVE,VALOR es decir MAP

get(clave) devuelve el valor

set(clave,valor) para fijar o modificar valores

```
let pokeda2 = new Map();  
  
pokemon.forEach((p, indice) => {  
  if (pokeda2.get(p) === undefined) pokeda2.set(p, 0);  
  pokeda2.set(p, pokeda2.get(p) + daño[indice]);  
});  
  
console.log(pokeda2);
```

3.- JavaScript : ARRAYS y OBJETOS

EJERCICIOS:

Dado el siguiente array:

```
const numeros = [3,44,10,1,3,4,56,7,8];
```

- Investiga como lo puedes ordenar con el método .sort
- Soluciona el problema de que ordena como si fueran textos

```
const pokemonArray = [  
  {  
    nombre: 'Pikachu',  
    tipo: 'Eléctrico',  
    fuerza: 50,  
    vida: 100,  
  },  
  {  
    nombre: 'Charizard',  
    tipo: 'Fuego/Volador',  
    fuerza: 80,  
    vida: 120,  
  },  
  // Agrega más Pokémon aquí...  
];
```

1. Añade tres pokemon más debajo de los comentarios (directamente)
2. Crea una función a la cual le pasamos el array y un objeto con un nuevo pokemon y te devuelva una copia del array con el nuevo pokemon añadido al final del array
3. Haz una función que devuelva el array ordenado por el nombre del pokemon
4. Haz una función que devuelva el array ordenado por la propiedad del objeto que el usuario elija

PISTA: Para acceder a una propiedad de un objeto

```
let a = pokemonArray[0]; // primer pokemon
```

3.- JavaScript : ARRAYS y OBJETOS

```
let propiedad = "fuerza";  
console.log(a[propiedad]); //accedo a la fuerza del primer pokemonArray
```

Yo usaría .sort(función de comparación)