Repaso de JavaScript

Vamos a ver dos estructuras de datos que son importantes

1. Arreglos o Arrays
2. Objetos

1. Arreglos o Arrays

Son estructuras de datos que nos permiten almacenar varios datos y agruparlos en una sola variable

Sintaxis: []

```
//     index:      0         1       2        3         4
const frutas = ["Manzana", "Pera", "Piña", "Sandía", "Melón"];

// Conocer cuantos elementos tiene un array
console.log( frutas.length ); // 5

// Acceder a un elemento del array
console.log( frutas[1] );

// Agregar un elemento a un array
frutas.push("Naranja"); // Agrega un elemento al final del array

console.log(frutas);

// Recorrer un array a partir de un método iterador
// forEach
// Cuando recibo una función como parámetro se le conoce como callback
frutas.forEach( function(fruta, index){
    console.log(fruta, index);
});
```

2. Objetos literales

Son un estructura de datos que nos permite agrupar datos declarando una clave y valor

Sintaxis de un objeto literal

```
const nombreObjeto = {
     clave: valor,
     clave: valor,
     clave: valor,
}
```

Ejemplo de un objeto literal

```
const termo = {
    color: 'verde',
    marca: 'Stanley',
    capacidad: 1,
    carga: 500,
    servir: function(){
        this.carga = this.carga - 100;
        console.log('glu glu glu');
    }
}

// Acceder a una propiedad de un objeto
console.log( 'El color del termo es ' + termo.color  )
console.log( 'la marca del termo es ' + termo.marca  )

// Ejecutar un método de un objeto
termo.servir();
termo.servir();

console.log( 'la carga del termo es ' + termo.carga  )
```

Combinación de estructura de datos (ARRAYS de OBJETOS)

Vamos a aplicar lo visto hasta ahora y usaremos los conceptos de array y objetos para crear una estructura de datos más compleja.

```
const products = [

    {
        id: 1,
        title: 'Termo',
        price: 500,
        description: 'Lorem ipsum dolor sit amet consectetur adipisicing
elit. Quisquam, voluptatum.',
        image: 'https://picsum.photos/200/300',
```

```
        category: 'Hogar',
        stock: 10,
    },
    {
        id: 2,
        title: 'Mate',
        price: 300,
        description: 'Lorem ipsum dolor sit amet consectetur adipisicing
elit. Quisquam, voluptatum.',
        image: 'https://picsum.photos/200/300',
        category: 'Hogar',
        stock: 6,
    },
    {
        id: 3,
        title: 'Cafetera',
        price: 1000,
        description: 'Lorem ipsum dolor sit amet consectetur adipisicing
elit. Quisquam, voluptatum.',
        image: 'https://picsum.photos/200/300',
        category: 'Hogar',
        stock: 7,
    },
    {
        id: 4,
        title: 'Televisor',
        price: 10000,
        description: 'Lorem ipsum dolor sit amet consectetur adipisicing
elit. Quisquam, voluptatum.',
        image: 'https://picsum.photos/200/300',
        category: 'Electrodomésticos',
        stock: 5,
    }
];
```

Funciones flecha (arrow functions)

Las funciones flecha son una forma más corta de escribir funciones en JavaScript. Y forman parte de las nuevas características que se agregaron en el 2015 con la versión de JavaScript ES6.

Sintaxis de una función flecha

```
() => {}
```

```
const nombreFuncion = (parametro1, parametro2) => {
    // código a ejecutar
}
```

Ejemplo de una función flecha

Esta es una función normal

```
function sumar(a, b){
    return a + b;
}
```

Esta es una función flecha

```
const sumar = (a, b) => {
    return a + b;
}
```

Si la función flecha tiene una sola línea de código, podemos omitir las llaves y la palabra return

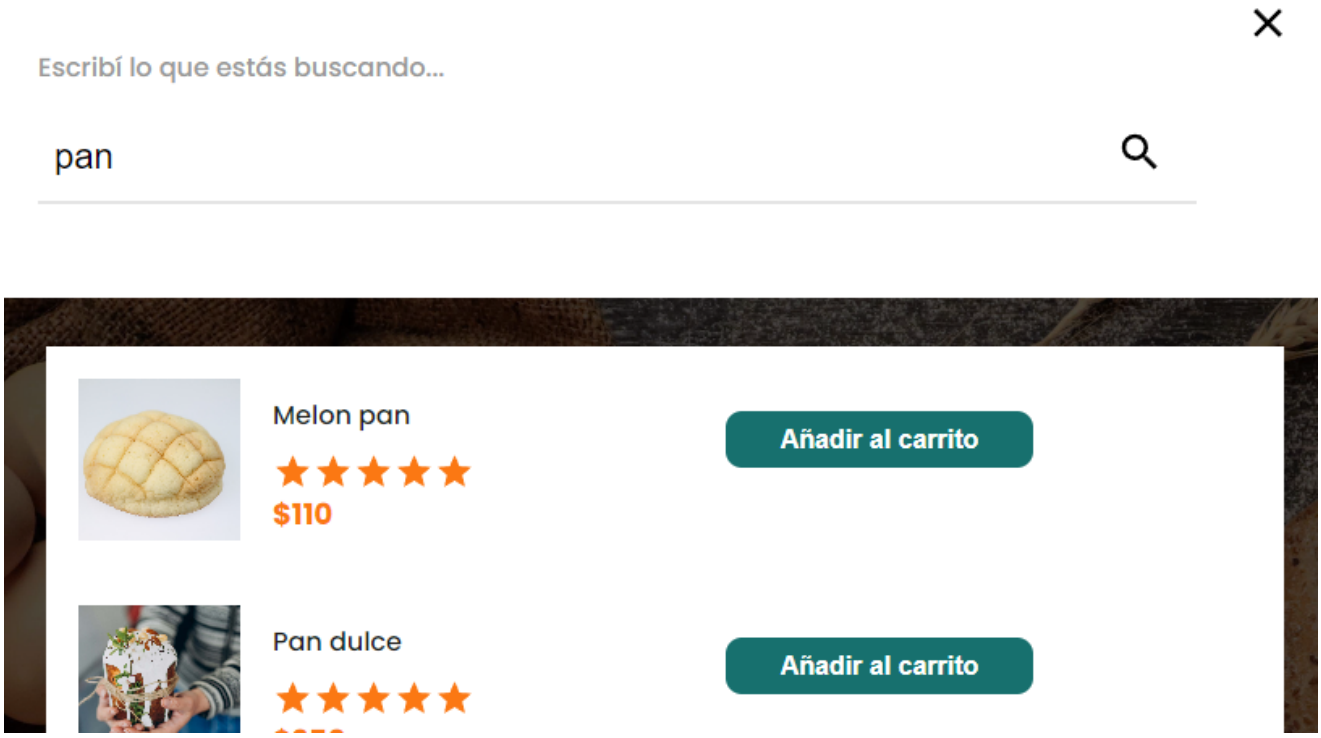
```
const sumar = (a, b) => a + b;
```

Si la función flecha recibe un solo parámetro, podemos omitir los paréntesis

```
const cuadrado = numero => numero * numero;
```

Ejercicio

1. Crear una función que reciba un array de productos y una query de búsqueda y devuelva un array con los productos que coincidan con la query de búsqueda



Para resolver este ejercicio usaremos el método filter de los arrays y las funciones flecha.

Método filter

El método filter es un método iterador que nos permite recorrer un array y filtrar los elementos que cumplan con una condición.

Sintaxis del método filter

```
nombreDeArray.filter( (element) => { return condición } )
```

Método includes

El método includes es un método que nos permite saber si un string incluye otro string.

Sintaxis del método includes

```
string.includes(stringABuscar)
```

Si el string incluye el stringABuscar, el método includes devolverá `true`, de lo contrario devolverá `false`.

```
const products = [

    {
        id: 1,
        title: 'Termo',
        price: 500,
```

```
        description: 'Lorem ipsum dolor sit amet consectetur adipisicing
elit. Quisquam, voluptatum.',
        image: 'https://picsum.photos/200/300',
        category: 'Hogar',
        stock: 10,
    },
    {
        id: 2,
        title: 'Mate',
        price: 300,
        description: 'Lorem ipsum dolor sit amet consectetur adipisicing
elit. Quisquam, voluptatum.',
        image: 'https://picsum.photos/200/300',
        category: 'Hogar',
        stock: 6,
    },
    {
        id: 3,
        title: 'Cafetera',
        price: 1000,
        description: 'Lorem ipsum dolor sit amet consectetur adipisicing
elit. Quisquam, voluptatum.',
        image: 'https://picsum.photos/200/300',
        category: 'Hogar',
        stock: 7,
    },
    {
        id: 4,
        title: 'Televisor',
        price: 10000,
        description: 'Lorem ipsum dolor sit amet consectetur adipisicing
elit. Quisquam, voluptatum.',
        image: 'https://picsum.photos/200/300',
        category: 'Electrodomésticos',
        stock: 5,
    }
];


const filterProductsByName = (arrayProducts, query) => {
    const result = arrayProducts.filter( prod =>
prod.title.toLowerCase().includes(query.trim().toLowerCase()) );
    return result;
}

console.log( filterProductsByName(products, 'M') );
```

Conclusiones

1. El método filter nos permite recorrer un array y filtrar los elementos que cumplan con una condición.
2. El método includes nos permite saber si un string incluye otro string.
3. Las funciones flecha son una forma más corta de escribir funciones en JavaScript.
4. Las funciones flecha forman parte de las nuevas características que se agregaron en el 2015 con la versión de JavaScript ES6.
5. Fusionar arrays y objetos nos permite crear estructuras de datos más complejas.