**React**

**Funciones.**

Documentación: <https://es.reactjs.org/>

<https://es.reactjs.org/docs/getting-started.html>

<https://developer.mozilla.org/es/docs/Web/JavaScript/Guide/Functions>

Una manera muy común de crear una función es asignándola a una variable constante, esto se hace por que en teoría el código sería más seguro. Porque el valor de una constante no podría ser modificada, mientras que la de una función normal sí.

Ejemplo:

// Esto es una función normal que puede ser modificada

function saludo(nombre)

{

    console.log(`Hola ${nombre}!`);

}

saludo = true;

saludo("Pedro"); // Error: saludo no es una función

// Esto es una función segura en una variable constante

const otroSaludo = function(nombre)

{

    console.log(`Hola ${nombre}!`);

}

otroSaludo = true; // Error: En asignación a variable constante.

otroSaludo("Juan");

**Funciones de flecha.**

**En la actualidad las funciones normales están siendo remplazadas por funciones de flecha,** esto no es más que una forma simplificada de trabajar con funciones, pero también tienen características especificas por las cuales fueron creadas. Una de ellas es el ayudarnos a trabajar con retornos implícitos, no cambiar a lo que apunta el this, entre otras cosas.

[**https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Global\_Objects/Array/some**](https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Global_Objects/Array/some)

Para crear una función de flecha:

1. Primeramente la palabra reservada function se tiene que eliminar.
2. Después de los paréntesis y parámetros se estable el **“=>”,** esto básicamente da la impresión de una flecha gorda.
3. Otra de las ventajas de las funciones de flecha, es en el caso en el que nuestro cuerpo solamente se tiene una línea de código y ese es el return, podemos eliminar las llaves de **“{}”** junto con la palabra reservada return.

Ejemplo:

const elSaludo = (nombre) => console.log(`Hola ${nombre}!`);

elSaludo("Mariana");

Ejemplos retornos de objetos:

const datosUsuario = () => {

    return {

        id: 1,

        nombre: "Benjamín",

        apellido: "Ayala",

        usuario: "bayalad7"

    }

};

const otroUsuario = () => ({

    id: 2,

    nombre: "Pedro",

    apellido: "Lee",

    usuario: "pdl1"

});

console.log( datosUsuario() );

console.log( otroUsuario() );

Ejemplos de buscar en un arreglo de objetos:

const heroes = [{

    id: 1,

    nombre: "Batman"

},{

    id: 2,

    nombre: "Superman"

},{

    id: 3,

    nombre: "Spiderman"

},{

    id: 4,

    nombre: "Ironman"

},{

    id: 5,

    nombre: "Dr Strange"

}];

// Existe heroe.id = 1

const encontro = heroes.some( (heroe) => heroe.id === 1 );

console.log({encontro}); // {encontro: true}

// Buscar heroe.id = 1, si find no encuentra nada retornará undefined

const heroe = heroes.find( (heroe) => heroe.id === 1 );

console.log({heroe}); // heroe: {id: 1, nombre: 'Batman'}

Para hacer una desestructuración de un objeto podemos usar las **{}** en la variable constante e indicar las propiedades que queremos obtener del objeto, sin importar el orden del acomodo de las propiedades.Ejemplo:

const {nombre, id} = heroes.find( (heroe) => heroe.id === 1 );

console.log(id); // 1

console.log(nombre); // Batman