**React**

**Desestructuración de objetos.**

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

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

<https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Operators/Destructuring_assignment>

La desestructuración de los objetos sirve para obtener cierta información específica que uno requiere o necesita y así evitar traer toda la información de un objeto.

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 persona = {

    id: 1,

    nombres: "Benjamín",

    apellido: "Ayala",

    edad: 30,

}

const {nombres, apellido} = persona;

console.log( nombres, apellido ); // Benjamín Ayala

const heroes = [{

    id: 1,

    nombre: "Batman",

    edad: 40,

},{

    id: 2,

    nombre: "Superman",

    edad: 40,

},{

    id: 3,

    nombre: "Spiderman",

    edad: 40,

},{

    id: 4,

    nombre: "Ironman",

    edad: 40,

},{

    id: 5,

    nombre: "Dr Strange",

    edad: 40,

}];

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

console.log(id); // 1

console.log(nombre); // Batman

console.log(edad); // 40

Otra forma de desestructurar los objetos es ir navegando sobre sus propiedades con un **“.” (punto).** Ejemplo:

const persona = {

    id: 1,

    nombres: "Benjamín",

    apellido: "Ayala",

    edad: 30,

    apodo: "Benyi",

}

console.log(persona.nombres); // Benjamín

console.log(persona.apellido); // Ayala

console.log(persona.edad); // 30

console.log(persona.apodo); // Benyi

Pero la idea de la desestructuración de los objetos es usar solamente las propiedades que necesitamos. Ejemplo:

const persona = {

    id: 1,

    nombres: "Benjamín",

    apellido: "Ayala",

    edad: 30,

    apodo: "Benyi",

}

// Desestructuración de información del objeto

const {nombres, apellido} = persona;

console.log( nombres, apellido ); // Benjamín Ayala

Una de las ventajas que tiene la desestructuración de los objetos, es que si declaramos una variable que no se encuentre dentro del objeto, esta solamente marcara undefined. **La razón de esto es porque básicamente todo lo que no está asignado en JS es undefined, pero una de las funciones interesantes de la desestructuración es que podemos poner un valor por defecto.** Ejemplo:

Ejemplo:

const persona = {

    id: 1,

    nombres: "Benjamín",

    apellido: "Ayala",

    edad: 30,

    apodo: "Benyi",

}

const { nombres, apellido, edad, apodo, altura, peso = 'Sin peso' } = persona;

console.log( nombres ) // Benjamín

console.log( apellido ) // Ayala

console.log( edad ) // 30

console.log( apodo ) // Benyi

console.log( altura ) // undefined

console.log( peso ) // Sin peso

Algo que también es muy usado en React, es realizar la desestructuración de los objetos en los parámetros de las funciones, Ejemplo:

const persona = {

    id: 1,

    nombres: "Benjamín",

    apellido: "Ayala",

    edad: 30,

    apodo: "Benyi",

}

// Desestructuración de información del objeto en los parámetros de la función

const retornaPersona = ({ id, nombres, apellido, edad, apodo}) => {

    console.log( id, nombres, apellido, edad, apodo );

}

retornaPersona( persona );

**Desestructuración de los objetos desde los argumentos de las funciones.**

Un uso bastante común es también la desestructuración desde los argumentos de las funciones. Que quiere decir esto, que en vez de desestructurar un objeto dentro de una función, se puede desestructurar desde el inicio en los argumentos que se reciben. Ejemplo:

1. **Desestructuración del objeto dentro de la función**

const crearHeroe = ( persona ) => {

    const { nombre, apellido, edad, apodo, altura } = persona;

    return {

        id: Math.floor(Math.random() \* 100),

        nombre: nombre,

        apellido: apellido,

        edad: edad,

        apodo: apodo,

        altura: altura

    };

}

1. **Desestructuración del objeto desde los argumentos de la función.**

const crearHeroe = ( { nombre, apellido, edad, apodo, altura } ) => {

    return {

        id: Math.floor(Math.random() \* 100),

        nombre: nombre,

        apellido: apellido,

        edad: edad,

        apodo: apodo,

        altura: altura

    };

}

**Nota importante.**

Cuando tenemos el caso en el que una propiedad tiene el mismo nombre que el nombre de una variable o constante que tengamos en nuestro código, podemos obviar la relación de la propiedad con el nombre de la variable o también podemos renombrar una propiedad con los **nombrePropiedad:nuevoNombre**. Ejemplo:

const crearHeroe = ( { nombre, apellido:apellidoPaterno, edad, apodo, altura } ) => (

    // Cuando tenemos el caso en el que una propiedad tiene el mismo nombre que el nombre de una variable o constante que tengamos en nuestro código, podemos obviar la relación de la propiedad con el nombre de la variable.

    // Relación de las propiedades con las variables constantes, los nombres serán las llaves y los valores serán lo que se tenga asignado en las variables constantes

    {

        id: Math.floor(Math.random() \* 100),

        nombre,

        apellidoPaterno,

        edad,

        apodo,

        altura

    }

);

**Desestructuración de objetos anidados.**

const persona = {

    id: 1,

    nombres: "Benjamín",

    apellido: "Ayala",

    edad: 30,

    apodo: "Benyi",

    domicilio: {

        colonia: "San Pedrito",

        calle: "Sector 7 #570",

        cp: 28219,

        municipio: "Manzanillo",

        estado: "Colima",

        pais: "México",

    }

}

// Desestructuración de un objeto anidado

// Opción 1 (la menos común)

const { domicilio:{colonia, calle, cp, municipio, estado, pais} } = persona;

console.log( colonia, calle, cp, municipio, estado, pais )

// Opción 2 (la más común)

const { domicilio } = persona;

const { colonia, calle, cp, municipio, estado, pais } = domicilio;

console.log( colonia, calle, cp, municipio, estado, pais )