Código base en cuerpo de documento para cargar React

<!-- Cargar React -->

        <script crossorigin src="https://unpkg.com/react@16/umd/react.production.min.js"></script>

        <script crossorigin src="https://unpkg.com/react-dom@16/umd/react-dom.production.min.js"></script>

        <script src="https://unpkg.com/babel-standalone@6/babel.min.js"></script>

**Variables y constants**

Var ya no se debe usar en javascript

const -> se utiliza para declarer vriables que no cambian nunca su valor

let-> se utiliza para declarar variables que pueden cambiar su valor

**Concatenacion de string**

Caso 1

const nombre='Yoisy';

 const apellido='Perez';

 const nombreCompleto= nombre + ' ' + apellido;

Caso 2

const nombre='Yoisy';

 const apellido='Perez';

 // const nombreCompleto= nombre + ' ' + apellido;

 const nombreCompleto= `

 ${nombre}

 ${apellido}

 `;

Caso 3

const nombre='Yoisy';

 const apellido='Perez';

 // const nombreCompleto= nombre + ' ' + apellido;

 const nombreCompleto= `

 ${nombre}

 ${apellido}

 `;

 console.log( nombreCompleto );

 function getSaludo (nombre) {

    return 'Hola mundo';

 }

console.log (`Este es un texto: ${ getSaludo() }` );

Caso 4

const nombre='Yoisy';

 const apellido='Perez';

 // const nombreCompleto= nombre + ' ' + apellido;

 const nombreCompleto= ` ${nombre} ${apellido} `;

 console.log( nombreCompleto );

 function getSaludo (nombre) {

    return 'Hola ' + nombre;

 }

console.log (`Este es un texto: ${ getSaludo(nombre) }` );

**Objetos literales o Diccionarios**

{} siempre significan que es un objeto

Ejemplo de declaración de objeto persona

 const persona = {

    nombre: 'Tony',

    apellido: 'Stark',

    edad: 45,

    direccion: {

        ciudad: 'Houston',

        zip: 77084,

        lat: 123.34,

        lng: 14.87

    }

 };

 //console.table (  persona  );

 console.log ( persona );

Los campos de un objeto pueden aparecer en orden alfabético de acuerdo al navegador aunque en el código sea otro orden.

**Clon de objetos:**

Para crear un clon de un objeto se utilizan los … delante del objeto que se desea clonar. Ejemplo:

 const persona = {

    nombre: 'Tony',

    apellido: 'Stark',

    edad: 45,

    direccion: {

        ciudad: 'Houston',

        zip: 77084,

        lat: 123.34,

        lng: 14.87

    }

 };

 //console.table (  persona  );

const persona2 = {...persona};

persona2.nombre = 'Peter';

 console.log ( persona );

 console.log ( persona2 )

Nunca se debe usar esta sintaxis: Persona 2=Persona, porque lo que se crea es una referencia del objeto Persona en Persona 2 y no un clon.

**Arreglos en JS**

Bibliografia: <https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Global_Objects/Array/map>

Los arreglos también se pueden clonar con el operador …

Ejemplo:

const arreglo = [1,2,3,4];

//arreglo.push(1)

//arreglo.push(2)

//arreglo.push(3)

//arreglo.push(4)

let arreglo2=[ ...arreglo, 5];

**Metodo map en arreglos**

El método **map()** crea un nuevo array con los resultados de la llamada a la función indicada aplicados a cada uno de sus elementos.

Ejemplo:

const arreglo = [1,2,3,4];

let arreglo2=[ ...arreglo, 5];

const arreglo3=arreglo2.map ( function (numero) {

return numero\*2;

});

Toda función en JS retorna undefined cuando no tiene un return especificado.

**Funciones en JS**

Para trabajar con funciones se recomienda declarar una const y asignarle la función en lugar de declarar la función únicamente

Funciones de flechas

Se puede cambiar la palabra reservada function por =>

Una función flecha puede devolver también un objeto en lugar de un return, para ello hay que colocarle paréntesis a los campos del objeto. Ejemplo

const getUsuarioActivo1 = (nombre) => ({

        uid:'AB123',

        username: nombre,

});

const usuarioActivo1 = getUsuarioActivo1('Fernando');

console.log(usuarioActivo1);

**Desestructuración de objetos**

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

Estudiar bibliografia

**Desestructuración de arreglos**

Estudiar bibliografi: <https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Global_Objects/Array/filter>

**Promesas**

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

**Declaracion de una promesa**

const promesa= new Promise((resolve, reject) => {

})

**Resolve:** lo que se ejecuta cuando la promesa se ejecuta

**Reject:** lo que se ejecuta cuando la prosa falla

Then

Cach

Finally

**Fetch API**

Es librerías que vienen incluidas en el navegador web o en el propio javascript para hacer peticiones http a endpoints sin utilizar ninguna libreria

Bibliografia: <https://developer.mozilla.org/es/docs/Web/API/Fetch_API>

DUDAS CON CALLBACK

**Crear app React**

Existen dos formas: Create React App y Vite

Vite es mucho mejor

Comando: yarn create vite, permite crear un proyecto react en vite

**Fragmaentos**

<>Fragmento, usado para agrupar elementos HTML. </>

**Impresion de variables en HTML**

const getResult = (a,b)=> {

  return a + b;

}

export const FirstApp = () => {

  return (

      <>

      <h1>{ getResult (1,10) }</h1>

      {/\* <code> {JSON.stringify (newMessage)} </code> \*/}

      <p> Soy un subtitulo</p>

      </>

    )

  }

**Crear estilos globales en React**

Se crear un fichero .css con todos los estilos (por ejemplo styles.css) y luego se importa en el main (import ‘./styles.css’)

**Combinaciones de teclas**

! + tab permite generar el código base en html

Ctrl + Espacio permite autocompletar

F2 encima de una variable permite cambiar su nombre y su referencia, todos los demás lugares donde aparezca.

Ctr+A selecciona todo el código

Ctrl+K+C comenta todo el código

Ctrl+K+U descomenta todo el código

Alt+Shift+f Formatea el c’odigo

**Bibliografia**

Mejor info sobre JS: MDN web docs, url

<https://developer.mozilla.org/es>