JavaScript

**Enlazar JavaScript con HTML**

* En línea: **<h1 onclick=“alert(‘hola’)”></h1>**
* Etiqueta script: **<script type=“text/javascript”>alert(‘hola’)</script>**
* Archivo en formato .js: **<script src=“codigo.js”></script>**
* Require: **¿?**
* Si no permite JavaScript: **<noscript>Activa JavaScript ;)</noscript>**

**Variables**

* Tipos de datos (primitivos): (*Son de tipado débil ya que pueden cambiar de tipo*)
* **String**: variable = “papel”; (o ‘papel’)
* **Number**: variable = 1;
* **Boolean**: variable = true; variable = (10 > 9);
* Casos especiales (primitivos):
* **Undefined**: La variable existe, pero no está inicializada, no tiene valor aún.
* **Null**: La variable está vacía. Tiene un valor, pero éste es nulo.
* **NaN (Not A Number)**: Error en una operación entre números y letras. Un número expresado en cadena NO es un NaN porque no lo trata como cadena.
* Otros Datos:
* **Object**: Objeto.
* **Function**: Variable que representa una función.
* Formas de declarar variables (Scope):
* **var**: var numero = 15; (existe en todo el documento y es el tipo por defecto con valor undefined)
* **let**: let numero = 15; (existe solo dentro del bloque que ha sido declarada y debe ser declarada)
* **const**: const numero = 15; (es una constante, no va a cambiar, se debe declarar e inicializar en la misma línea, y solo existe en su bloque)
* Declarar múltiples variables: let numero1 = 23, numero2 = 4, numero3 = 77;

**Operadores**

* Aritméticos: + // - // \* // / // ++ // -- // \*\*2 // x&y (resto).
* Asignación: = // += // %= // ...
* Adicción, sustracción, multiplicación, división: += // -= // \*= // /=
* Resto (introduce el resto de la división): %=
* Exponenciación (elevar al número introducido): \*\*=
* De Comparación:
* Simples: == != > < >= <= (texto(“25”) == numero(25) -> true)
* Comparación estricta: === !== (texto(“25”) === numero(25) -> false)
* Lógicos: && // || // !

**Concatenación**

* Signo +: frase 1 + frase 2;
* Forzar a String: frase = “” + 1 + 2; (Resultado: “12”) (*Coerción dinámica de tipos*)
* Función Concat: numero1.concat(numero2); (Resultado: “12”)
* Backticks: frase = `Soy ${nombre}`; (Resultado: “Soy Mario”)

**Backticks (``)**

* Concatenación: frase = `Soy ${nombre}`; (Resultado: “Soy Mario”)
* Escritura literal: `“Dobles” ‘Simples’` (Resultado: “Dobles” ‘Simples’)
* Código HTML: `<h1>Título</h1>` (Resultado: **Título**)
* Escape: “Soy ‘Mario’, encantado” /// ‘Soy “Mario”, encantado’ /// ‘Soy `Mario`, encantado’

**Condicionales**

* if: if(true){alert(“hola”)}
* else if: if(num>5){alert(“hola”)} else if(num<5){alert(“adios”)} else{alert(“.”)}

**Arrays**

* Arrays Clásicos: let miArray = [“Platano”,“Manzana”,“Pera”,6,2];
* Llamada: alert(miArray[0]);
* Arrays Asociativos: let pc = {nombre:“MSI”, cpu:“intel”, ram:“16GB”};
* Llamada: alert(pc[“nombre”]);

**Bucles e Iteración**

* Bucles while: while(x > y){alert(“mayor”)};
* Bucles do-while: do{alert(“mayor”)} while(x>y);
* Bucles for: for(let i = 0; i < y; i++){alert(“mayor”)}
* Bucles for in: for(fruta in miArray){alert(fruta)} (devuelve la posición)
* Bucles for of: for(fruta of miArray){alert(fruta)} (devuelve el dato)
* Sentencia break: break; (Detiene el bucle)
* Sentencia continue: continue; (Pasa directamente a la siguiente vuelta del bucle)
* Sentencia label: bucle1: (se pone encima de las estructuras para darles nombre)
* Secuancia break específica: break bucle1;
* Secuancia continue específica: continue bucle1;

**Funciones y Procedimientos**

* Funciones y Procedimientos predefinidos más usados:
* Mostrar output: **alert(“hola”)**;
* Mostrar input: **let nombre = prompt(“Escribe tu nombre”)**;
* Escribe contenido en la pantalla: **document.write(frase)**;
* Parseo a tipo number: **let numero =** **parseInt(12.0)**;
* Recuperar un elemento por su Id: **document.getElementById(5)**;
* Imprimir en la consola: **console.log(“hola”)**;
* Funciones clásicas: **function miFuncion(num1,num2){return num1+num2;}**
* Llamada: **alert(miFuncion(33,44))**;
* Parámetros por defecto: **function saludo(msj=”Hola”){console.log(msj)}**
* Número variable de parámetros:

**function f(x,y,...z){let suma=0; for(num for z){media+=num;}return suma;}**

* Funciones anónimas: **let f=function(msj=”Hola”){console.log(msj);}**
* Llamada: **f(“Buenos días”)**;
* Funciones flecha: (*si tiene return no se ponen llaves*)

**const miFuncion = () =>`Hola`;**

**const miFuncion = nom =>`Hola ${nom}`;**

**const miFuncion = (nom, ape) =>`Hola ${nom} ${ape}`;**

* Llamada: **console.log(miFuncion(“Fran”))**;

**Clases y Objetos**

* Objeto Simple: **const car = {type:"Fiat", toString: function(){return “hola”;}};**
* Estructura de Clase: **class animal{}**
* Constructores y Atributos: **constructor(especie){this.especie = especie;}**
* Método: **saludar(){document.write(“Hola”);}**
* Instancia de Objeto: **const perro = new animal("perro")**;
* Llamada de métodos: **perro.saludar()**;
* Getters: **get getRaza(){return this.raza;}**
* Llamada: **let raza1 = perro.getRaza**;
* Setters: **set setRaza(valor){this.raza=valor;}**
* Llamada: **perro.setRaza = “Doberman”**; /// **perro.setRaza(“Doberman”)**;
* Herencia: **class perro extends animal{constructor(especie){super(especie);}}**

**Eventos**

* HTML: **<button onclick="this.innerHTML = Date()">Probar</button>**
* Script: **<button onclick="displayDate()">The time is?</button>**
* Tipos HTML: **onchange;onclick;onmouseover;onmouseout;onkeydown;onload**;
* JavaScript: **document.getElementById("prueba").addEventListener("click", function( event ) {**

**event.target.innerHTML = "Conteo de Clicks: " + event.detail;})**;

**Métodos de Números**

* Convertir a String: **x.toString()**;
* Redondear a x decimales: **x.toFixed(2)**;
* Redondear a x decimales en formato exponencial: **x.toExponential(2)**;
* Delimitar la longitud de cifras: **x.toPrecision(6)**;
* Parseo de variable a numero: **Number(" 10")**; (Resultado: 10)
* Parseo a decimal: **parseFloat("10.33")**; (Resultado: 10.33)
* Parseo a entero: **parseInt("10.33")**; (Resultado: 10)

**Métodos de Cadenas**

* Concatenar: **cadena1.concat(cadena2)**;
* Verificar si empiezan igual desde x posición: **cadena1.startsWith(cadena2, 5)**;
* Verificar si terminan igual hasta x posición: **cadena1.endsWith(cadena2, 16)**;
* Verificar si está incluido a partir de x posición: **cadena1.includes(cadena2, 12)**;
* Primera posición de la cadena: **cadena1.search(“p”)**;
* Primera posición de la cadena desde x numero: **cadena1.indexOf(“p”, 7)**;
* Última posición de la cadena hasta x numero: **cadena1.lastIndexOf(“p”, 15)**;
* Carácter alojado en x posición: **cadena1.charAt(0)**; /// **cadena1[0]**;
* Carácter Unicode alojado en x posición: **cadena1.charCodeAt(0)**;
* Rellenar con algo por delante hasta x longitud: **cadena1.padStart(6,“a”)**;
* Rellenar con algo por detrás hasta x longitud: **cadena1.padEnd(6,“a”)**;
* Longitud de la cadena: **cadena1.length**;
* Repite la cadena x veces: **cadena1.repeat(4)**;
* Divide la cadena según x criterio: **cadena1.split(“,”)**;
* Recorta de una posición a otra hasta con negativos: **cadena1.slice(-12, -6)**;
* Recorta de una posición a otra sin negativos: **cadena1.substring(0, 4)**;
* Recorta desde una posición x caracteres con negativos: **cadena1.substr(-7, 3)**;
* Reemplaza la primera ocurrencia por otra: **cad2 = cad1.replace("mar", "Mar")**; (*Al ser key sensitive se puede usar* ***/****MAR****/i*** *para anularlo*)

(*Para reemplazar todas las ocurrencias debe usarse /mar/g*)

* Convertir a minúscula: **cadena1.toLowerCase()**;
* Convertir a mayúscula: **cadena1.toUpperCase()**;
* Elimina espacios en blanco a ambos lados: **cadena1.trim()**;
* Elimina espacios en blanco del principio: **cadena1.trimStart()**;
* Elimina espacios en blanco del final: **cadena1.trimEnd()**;
* Devuelve el valor primitivo: **cadena1.valueOf()**;
* Devuelve un array con los matcheos: **cadena1.matches(/ain/g)**; (regular expression)

**Métodos de Objtos Arrays**

* Crear un array con x elementos indefinidos: **let array = new Array(8)**;
* Verificar si algo es un array: **elemento.isArray()**; // **elemento instanceof Array**
* Longitud del array: **array.length**;
* Convierte a cadena separada por comas: **array.toString()**;
* Convierte a cadena con separador: **array2 = array.join(“ - ”)**;
* Ordena los elementos incluso mediante criterios: **array.sort(function(a,b))**;
* Invierte el orden de los elementos: **array.reverse()**;
* Extrae y elimina el último elemento: **elemento = array.pop()**;
* Agrega elementos al final: **array.push(“mario”,“sara”)**;
* Extrae y elimina el primer elemento: **elemento =** **array.shift()**;
* Agrega elementos al principio: **array.unshift(“mario”)**;
* Vacía una posición del array (no elimina): **delete array[0]**;
* Concatena dos arrays: **array3 =** **array1.concat(array2)**;
* Agrega/reemplaza una porción del array: **array2 = array.splice(6, 4, “mario”)**;
* Devuelve una porción del array: **array2 = array.slice(1, 4)**; (*4 no incluído*)
* Verificar si está incluido: **array.includes(“mario”)**;
* Primera posición de la cadena desde x posición: **array.indexOf(“mario”, 3)**;
* Última posición de la cadena hasta x posición: **array.lastIndexOf(“mario”, 9)**;
* Bucle con función: **array.forEach(numero => document.write(numero+<br>))**;

(*También se usa dentro una función sin paréntesis con “value”, “index” y “array”*)

* Bucle con función y condición: **array.filter(numero => numero.length > 4)**;

(*También se usa dentro una función sin paréntesis con “value”, “index” y “array”*)

* Primer elemento que cumpla el criterio; **array.find(myFunction)**;

(*Se llama a una función con “value”, “index” y “array”*)

* Primera posición que cumpla el criterio; **array.findIndex(myFunction)**;

(*Se llama a una función con “value”, “index” y “array”*)

* Transforma cada elemento del array; **array.map(myFunction)**;

(*Se llama a una función con “value”, “index” y “array”*)

* Reducir array a un valor por la izquierda: **array.reduce(myFunction)**;

(*Se llama a una función con “total”, “value”, “index” y “array”*)

* Reducir array a un valor por la derecha: **array.reduceRight(myFunction)**;

(*Se llama a una función con “total”, “value”, “index” y “array”*)

* Verifica si todos los elementos cumplen la condición: **array.every(myFunction)**;

(*Se llama a una función con “value”, “index” y “array”*)

* Verifica si algún elemento cumple la condición: **array.some(myFunction)**;

(*Se llama a una función con “value”, “index” y “array”*)

* Transforma a array de caracteres una cadena: **Array.from("ABCDEFG")**;
* Devuelve un array con los números de las posiciones: **array.keys()**;
* Devuelve un array con clave/valor por cada posición: **array.entries()**;
* Número máximo: **Math.max.apply(null, array)**;
* Número mínimo: **Math.min.apply(null, array)**;
* Deja como undefined una posición de un array: **delete array2[1]**;

**Objeto Math**

* Potencia: **Math.pow(25,2)**;
* Raíz cuadrada: **Math.sqrt(25)**;
* Raíz cúbica: **Math.cbrt(25)**;
* Devuelve el número mayor: **Math.max(1,6,34,52,321)**;
* Devuelve el número menor: **Math.min(1,6,34,52,321)**;
* Devuelve un número aleatorio: **Math.floor(Math.random() \* 100) + 1;**
* Redondea un número: **Math.round(numero)**;
* Redondea un número con 15 decimales: **Math.fround(numero)**;
* Redondea un número hacia abajo: **Math.floor(numero)**;
* Redondea un número hacia arriba: **Math.ceil(numero)**;
* Trunca un número: **Math.trunc()**;
* Indica si es negativo, positivo o neutro: **Math.sign(-4)**;
* Devuelve el número absoluto: **Math.abs(-4.7)**;
* Número Pi: **Math.PI**
* Raíz cuadrada de 1,5: **Math.SQRT1\_2**
* Raíz cuadrada de 2: **Math.SQRT2**
* Constante de Euler: **Math.E**
* Logaritmo natural de 2: **Math.LN2**
* Logaritmo natural de 10: **Math.LN10**
* Logaritmo de E en base 2: **Math.LOG2E**
* Logaritmo de E en base 10: **Math.LOG10E**

**Métodos de Fechas**

* DateTime actual: **const fecha = new Date()**;
* Fecha específica: **fecha = new Date(2022, 9, 19, 16, 58, 30, 0)**;
* Fechas procedentes de Strings: **fecha = new Date(“October 13, 2014 11:13:00”)**;
* Convierte a cadena UTC: **objFecha.toUTCString()**;
* Convierte a cadena clásica: **objFecha.toDateString()**;
* Convierte a cadena en ISO: **objFecha.toISOString()**;
* Fomato ISO de fecha: **const fecha = new Date("2015-03-25")**;
* Fomato ISO DateTime: **const fecha = new Date("2015-03-25** **T12:00:00Z")**;
* Formato de fecha corta: **const fecha = new Date("03/25/2015")**;
* Formato de fecha larga: **const fecha = new Date("Mar 25 2015")**;
* Convertir a milisegundos: **milis = Date.parse("March 21, 2012")**;
* Getters: **getFullYear(); getMonth(); getDate(); getHours(); getMinutes(); getSeconds(); getMilliseconds(); getTime(); getDay(); Date.now()**;
* Setters: **setDate(); setFullYear(); setHours(); setMilliseconds(); setMinutes(); setMonth(); setSeconds(); setTime()**;

**Métodos de la Consola de Comandos**

* Comprobar condición: **console.assert(5 < 3)**;
* Limpiar consola: **console.clear()**;
* Mensaje de error personalizado: **console.error(“Error personalizado”)**;
* Mensaje de error personalizado: **console.info(“Debes guardar”)**;
* Mensaje de depuración personalizado: **console.log(“%cBien depurado”,“color:red;background:yellow;padding:20px;border:3px solid blue”)**;
* Mensaje de advertencia personalizado: **console.warm(“Debes hacer clic”)**;
* Muestra un array/objeto: **console.table([1,5,23,56,87])**;
* Muestra las propiedades del objeto: **console.dir([1,5,23,56,87])**;
* Aumenta 1 al contador de la consola: **console.count()**;
* Resetea el contador de la consola: **console.countReset()**;
* Crea un nuevo nivel de grupo abierto con nombre: **console.group(“Frutas”)**;
* Crea un nuevo nivel de grupo cerrado con nombre: **console.groupCollapsed()**;
* Retrocede un nivel de grupo: **console.groupEnd()**;
* Inicia un temporizador: **console.time()**;
* Muestra la cantidad de tiempo desde que se inició: **console.timeLog()**;
* Elimina el temporizador y muestra el tiempo: **console.timeEnd()**;

**DOM (Document Object Model) –** **Métodos de Selección de Elementos**

* Recupera el objeto del elemento por ID: **document.getElementById(“parrafo”)**;
* Recupera una colección por nombre: **document.getElementsByTagName(“p”)**;
* Devuelve el 1er objeto que coincida: **document.querySelector(“.parrafo”)**;
* Devuelve un NodeList de la búsqueda: **document.querySelectorAll(“.parrafo”)**;

**DOM – Métodos para Definir, Obtener y Eliminar Valores de Atributos**

* Modifica y agrega atributos: **miObjeto.setAttribute(“type”,“text”)**;
* Recupera el valor de un atributo: **miObjeto.getAttribute(“type”)**;
* Elimina un atributo: **miObjeto.removeAttribute(“type”)**;

**DOM – Atributos Globales**

* Permite la edición: **miObjeto.setAttribute(“contenteditable”, “true”)**;
* Dirección del elemento: **miObjeto.setAttribute(“dir”, “rtl”)**;
* LeftToRight: ltr (*por defecto*)
* RightToLeft: rtl
* Oculta elementos: **miObjeto.setAttribute(“hidden”, “true”)**; (*false hace lo mismo*)
* Permite seleccionar elementos con tab: **miObjeto.setAttribute(“tabindex”, “0”)**;
* Crea una nota de texto al pasar el crusor: **miObjeto.setAttribute(“title”, “Título”)**;

**DOM – Atributos de Inputs**

* Recupera/Modifica el nombre de clase: **miObjeto.className**; /// **= “Nueva clase”**;
* Recupera/Modifica el valor del elemento: **miObjeto.value**; /// **= “5”**;
* Recupera/Modifica el tipo del elemento: **miObjeto.type**; /// **= “number”**;
* Recupera/Modifica los archivos que acepta: **miObjeto.accept**; /// **= “image/gif”**;
* Recupera/Modifica el id del formulario: **miObjeto.form**; /// **= “form2”**;
* Recupera/Modifica el mínimo de caracteres: **miObjeto.minLength**; /// **= “10”**;
* Recupera/Modifica el texto en gris de los text: **miObjeto.placeholder**; /// **= “Hola”**;
* Recupera/Modifica el requerimiento del campo: **miObjeto.required**; /// **= “ ”**;

**DOM – Atributo Style**

* Modifica el color: **miObjeto.style.color = “red”**;
* **miObjeto. style.backgroundColor = “blue”**; (*siempre CamelCase*)
* [...]

**Clases, ClassList y Métodos de ClassList**

* Añade una clase: **miObjeto.classList.add(“grande”)**;
* Elimina una clase: **miObjeto.classList.remove(“grande”)**;
* Devuelve el nombre de la clase especificada: **miObjeto.classList.item(1)**;
* Verifica si la clase contiene el elemento: **miObjeto.classList.contains(“grande”)**;
* Reemplaza una clase por otra: **miObjeto.classList.replace(“grande”,“mediana”)**;
* Añade una clase si no está/Elimina si está: **miObjeto.classList.toggle(“grande”)**;

**Obtención y Modificación de Elementos**

* Texto sin formato en documento pero con solo texto en alert: **textContent**;
* Texto con formato en documento pero con HTML reducido en alert: **innerHTML**; (*también se usa para introducir texto a un elemento:* ***item.innerHTML = “texto”***)
* Texto con formato en documento pero con HTML extenso en alert: **outerHTML**;

**Creación de Elementos**

* Crea elementos: **item = document.createElements(“LI”)**; (*siempre Mayúsculas*)
* Crea texto dentro de un elemento: **texto =** **document.createTextNode(“Texto”)**;
* Crea fragmentos: **frag = document.createDocumentFragment()**; (*para ahorrar recursos los nuevos elementos se meten en el fragmento y después éste se mete en el documento*)

**Obtención y Modificación de Childs (Hijos)**

* Obtiene el primer hijo: **miObjeto.firstChild**; (*el texto cuenta*)
* Obtiene el último hijo: **miObjeto.lastChild**; (*el texto cuenta*)
* Obtiene el primer elemento hijo: **miObjeto.firstElementChild**;
* Obtiene el último elemento hijo: **miObjeto.lastElementChild**;
* Devuelve todos los nodos hijo (NodeList): **miObjeto.childNodes**; (*forEach*)
* Devuelve todos los elementos hijo (colección): **miObjeto.children**; (*for in/of*)
* Devuelve el elemento ascendente más cercano: **miObjeto.closest(“.div”)**;

**Métodos de Childs (Hijos)**

* Añade un elemento hijo: **objetoElemento.appendChild(objetoElemento)**;
* Reemplaza un elemento: **objetoElemento.replaceChild(objElem2, objElem1)**;
* Elimina un elemento hijo: **objetoElemento.removeChild(objetoElemento)**;
* Verifica si tiene hijos: **objetoElemento.hasChildNodes()**;

**Propiedades de Parents (Padres)**

* Selecciona el elemento padre de un elemento: **parent = elem.parentElement**;
* Selecciona el nodo padre de un elemento: **parent = elem.parentNode**;

**Propiedades de Siblings (Hermanos)**

* Obtiene el siguiente nodo hermano: **objetoElemento.nextSibling**;
* Obtiene el anterior nodo hermano: **objetoElemento.previousSibling**;
* Obtiene el siguiente elemento hermano: **objetoElemento.nextElementSibling**;
* Obtiene el anterior elemento hermano: **objetoElemento.previousElementSibling**;

**Métodos JSON**

* Convierte texto a JSON: **const obj = JSON.parse(text)**;

**Callbacks, Promesas y Async/Await**

* Estructura Callback: **function calcule(n1, n2, oper){return oper(n1, n2);}**;
* Llamada Callback: **calcule(1, 3, (a, b) => a + b)**;
* Función Timeout (Esperar): **setTimeout(myFunction, 3000)**;
* Función setInterval (Cada x tiempo): **id =** **setInterval(myFunction, 1000)**;
* Función clearInterval(detener interval): **clearInterval(idSetInterval);**
* Función setTimeout (Delay): **setTimeOut(myFunction, 1000)**;
* PRODUCCIÓN Promesas: **prm = new Promise(function(resolve, reject){**

**if(some) resolve(datos); else reject(error);})**;

* CONSUMO Promesas: **prm.then(function(datos) {}, function(error) {})**; ///

**prm.then((datos) => console.log(datos))**;

* Estados Promesas: **prm.state = "pending"; "fulfilled"; "rejected"**;
* Resultados Promesas: **prm.result = "undefined"; "value"; "error"**;
* Async/Await: **async function fetch(){const dat=await getDat(); return dat;}**

**React (Framework-Librería) y NodeJS (Servidor Local)**

* Componente JSX funcional o Hooks: **function Saludo(props){return <h1>Hola {props.nombre}</h1>;}** (devuelve JSX)
* Componente JSX de clase: **class Saludo extends React.Component{ render(){return <h1>Hola {this.props.nombre}</h1>;}}**
* Importaciones Específicas: **import React from ‘react’;** // **import ReactDOM from ‘react-dom’;**
* Importación General: **import ‘index.css’**;
* Importación nombrada: **import {Album} from ‘Album.js’**;
* Exportación por defecto (1 por archivo): **export default Saludo**;
* Exportación nombrada (+1 por archivo): **export function Album(){return “hi”}**
* React Icons (cd Proyecto): **npm install react-icons –save**

**import { IconName } from "react-icons/ai"**;

* Generador de IDs UUID: **npm install uuid**

**import { v4 as uuidv4} from 'uuid'**;

**let id = uuidv4()**;

* Elemento raíz o principal en HTML: **<div id=“root”></div>** (React DOM)
* Elemento React JSX: **const elemento =** **<h1>Hola Mundo</h1>;**
* Componente React JSX: **<Boton manejarClic={agregarInput}>1</Boton>**
* Atributo class en JSX: **<h1 className=“title”></h1>**
* Atributo for en JSX: **<label htmlFor=“css”></label>** (*desconocido*)
* Propiedades CSS en JSX: **const estiloDiv = {backgroundImage: red;}**

Referencia: **<div style={estiloDiv}>Hola Mundo</div> (ó style={{color:’red’}})**

* Renderizar/Insertar código: **ReactDOM.render(elemento, document.getElementById(‘root’));**
* Crear proyecto React (dentro de proyectos): **npx create-react-app [NewFile]**
* Iniciar servidor (dentro del proyecto): **cd [NewFile] /// npm start**
* Archivo .css de la aplicación: **App.css**
* Archivo .js de la aplicación: **App.js** (incluye el componente funcional principal)
* Archivo .css de estilo general: **index.css**
* Archivo .js del script principal: **index.js** (incluye el reactDOM.render();)
* Ruta de una imagen: **src={require('../imagenes/testimonio-nevermind.jpg')}**
* Fragmentos de React (dentro de return): **<> ... </>**
* Estado de un Componente: **import { useState } from "react"**;

**const [valor, funcionModific] = useState(0)**;

**function sumar(){funcionModific(valor+1);}**

**function restar(){funcionModific(valor-1);}**

(*Se declara el valor inicial del estado a cero y se incluye la función que permitirá la modificación de dicho estado cuando ésta sea llamada en el mismo documento; Después se crearán las funciones que hará uso de la función de modificación y que serán pasadas como parámetros a los componentes.*)

* useEffect
* useContext
* Componentes de clase: **class MiComponente extends React.Component{**

**constructor(props){super(props); this.state = {completo: true, prioridad: 1};}**

**this.setState({completada: false, prioridad: 2});**

**render(){return <p>Propiedad = {this.state.prioridad}</p>;}}**

Página: React Router (Main Concepts/Defining Routes)

Indicas que una ruta especifica html renderizara un determinado componente.

v5.reactrouter.com/web/example/url-params

useEffect: se ejecuta su código tras imprimirse el contenido.