Módulos

Los módulos permiten exportar e importar código y/o datos entre diferentes archivos de JavaScript, promoviendo organización y reutilización.

```
// Exportar un elemento: permite usarlo en otros archivos
export const articulos = [
        {id:1, nombre:"articulo1"},
        {id:2, nombre:"articulo2"},
        {id:3, nombre:"articulo3"}
];

// Importar un elemento desde otro archivo
import {articulos} from "./datos.js";

// Uso en HTML: el atributo "type=module" permite usar módulos en JavaScript
<script src="./js/main.js" type="module"></script>
```

Ámbito

Los módulos tienen un ámbito propio y a veces el HTML no puede acceder a sus variables directamente. El DOM permite interactuar con esos elementos.

```
// Accede al botón en el DOM y agrega un listener que ejecuta una función al hacer clic
document.getElementById("btn").addEventListener("click", verMensaje);
```

Sintaxis

Variables

```
var persona = "pedro"; // variable de alcance global
let persona = "pedro"; // variable de alcance en bloque
const persona = "pedro"; // variable definida como constante, tras sun inicialización no se puede modificar
```

Conversiones de tipo

```
// IMPLÍCITAS -> automáticamente
let numero = 10;
let texto = "5";
let suma = numero + texto; // "105" (convierte número a string)

// EXPLÍCITAS -> forzando
let texto = "10";
let numero = Number(texto); // Convierte a número
console.log(numero); // 10
```

Operadores

Operadores Aritméticos

```
// Realizan operaciones matemáticas básicas.
let suma = 5 + 2; // suma = 7
let resto = 10 % 3; // resto = 1
```

Operadores Relacionales

```
// Comparan valores y devuelven `true` o `false`.
let igualdad = (a == b); // true si a es igual a b
```

Operadores Lógicos

```
// Realizan operaciones lógicas `AND` = &&, `OR` = || y `NOT` = !.
let and = (a && b); // true si ambos son true
```

Operadores de Asignación

```
// Permiten asignar y actualizar valores en una variable.
let a = 5; a += 2; // equivale a: a = a + 2
```

Operador TypeOf

```
// El operador `typeof` permite verificar el tipo de dato de una variable.
console.log(typeof "string"); // "string"
console.log(typeof 10); // "number"
console.log(typeof true); // "boolean"
// object // undefined // symbol // function
```

Condicionales

Sentencia `if-else`:

```
if (condicion) {
 // Código a ejecutar si la condición es verdadera
} else {
 // Código a ejecutar si la condición es falsa
}
```

Sentencia `switch`:

```
switch (expresion) {
        case valor1:
                // Código a ejecutar si la expresión es igual a valor1
                break;
         case valor2:
                // Código a ejecutar si la expresión es igual a valor2
                break;
         default:
                // Código a ejecutar si la expresión no coincide con ningún caso
}
```

Operador condicional `ternario (? :)`:

```
let resultado = (condicion) ? true : false;
// true = si se cumple
// false = si no se cumple
```

Bucles

Bucle `for`

```
// ejecuta un bloque de código un número específico de veces
for (let i = 0; i < 10; i++) {
 // Código a ejecutar 10 veces
}

// ### PROGRAMACIÓN FUNCIONAL ###
Array.from({ length: 10 }).forEach((_, i) => {
  // Código a ejecutar 10 veces
  console.log(`Iteración ${i + 1}`);
});
```

Bucle `for-in`

```
// Recorre las propiedades de un objeto.
let persona = { nombre: "Juan", edad: 30, profesion: "Desarrollador" };

for (let propiedad in persona) {
 console.log(`${propiedad}: ${persona[propiedad]}`);
}

// ### PROGRAMACIÓN FUNCIONAL ###
const persona = { nombre: "Juan", edad: 30, profesion: "Desarrollador" };

Object.entries(persona).forEach(([propiedad, valor]) => {
  console.log(`${propiedad}: ${valor}`);
});
```

Bucle `for-of`

```
// Recorre los valores de un array o cadena de texto (iterable).
let numeros = [10, 20, 30];

for (let numero of numeros) {
 console.log(numero);
}

// ### PROGRAMACIÓN FUNCIONAL ###
const numeros = [10, 20, 30];

numeros.forEach(numero => console.log(numero));
```

Bucle `while`

```
// ejecuta X mientras una condición se cumpla
let i = 0;

while (i < 10) {
 // Código a ejecutar mientras i sea menor que 10
 i++;
}

// ### PROGRAMACIÓN FUNCIONAL ###
const repetirHasta = (condicion, accion, i = 0) => {
  if (!condicion(i)) return;
  accion(i);
  repetirHasta(condicion, accion, i + 1);
};

// Ejemplo de uso:
repetirHasta(i => i < 10, i => console.log(`Iteración ${i + 1}`));
```

Bucle `do-while`

```
// ejecuta X 1 vez, luego comprueba si se cumple y mientras sea así hace X
let i = 0;

do {
 // Código a ejecutar al menos una vez
 i++;
} while (i < 10);

// ### PROGRAMACIÓN FUNCIONAL ###
const repetirAlMenosUnaVez = (condicion, accion, i = 0) => {
  accion(i);
  if (!condicion(i)) return;
  repetirAlMenosUnaVez(condicion, accion, i + 1);
};

// Ejemplo de uso:
repetirAlMenosUnaVez(i => i < 10, i => console.log(`Iteración ${i + 1}`));
```

Objetos Predefinidos

Window

representa la ventana del navegador y proporciona acceso a sus propiedades y métodos

Propiedades

```
// 1. location: Proporciona información sobre la URL actual de la página web y permite redirigir a otras páginas.
console.log(window.location.href); // Imprime la URL actual
window.location.href = "https://www.example.com"; // Redirige a otra página

// 2. document: Representa el documento HTML de la página web, proporcionando acceso a sus elementos y métodos (DOM - Document Object Model).
let titulo = document.getElementById("tituloPagina");
titulo.textContent = "Nuevo título de la página"; // Cambia el texto del título

// 3. history: Proporciona acceso al historial del navegador, permitiendo navegar entre las páginas visitadas.
window.history.back(); // Navega a la página anterior
window.history.go(2);  // Navega dos páginas hacia atrás

// 4. screen: Proporciona información sobre las dimensiones de la pantalla, como la resolución.
let anchoPantalla = window.screen.width;
console.log(`Ancho de la pantalla: ${anchoPantalla} pixeles`); // Imprime el ancho de pantalla

// 5. navigator: Proporciona información sobre el navegador utilizado, como el nombre, la versión y el sistema operativo.
let navegador = window.navigator.userAgent;
console.log(`Navegador: ${navegador}`); // Imprime la información del navegador
```

Métodos

```
// 1. alert(): Muestra un cuadro de diálogo emergente con un mensaje y un botón de "Aceptar".
window.alert("¡Hola, mundo!"); // Muestra una alerta al usuario

// 2. prompt(): Muestra un cuadro de diálogo emergente para obtener información del usuario.
let nombre = window.prompt("Dime un nombre:");
console.log("Nombre:", nombre); // Imprime el nombre introducido o `null` si se canceló

// 3. confirm(): Muestra un cuadro de diálogo emergente para confirmar una acción, con "Aceptar" y "Cancelar".
let confirmar = window.confirm("¿Desea continuar?");
if (confirmar) {
        console.log("Usuario confirmó la acción");
} else {
        console.log("Usuario canceló la acción");
}

// 4. open(): Abre una nueva ventana del navegador con la URL especificada.
window.open("https://www.example.com", "_blank"); // Abre en una nueva pestaña

// 5. close(): Cierra la ventana actual o una ventana abierta con open.
window.close(); // Cierra la ventana actual (si fue abierta mediante JavaScript)
```

String

Creación

```
let nombre = "Juan";
let saludo = `Hola, ${nombre}`;
console.log(saludo); // Imprime "Hola, Juan"

let nombre = new String(‘Juan’); // a través de constructor
```

Propiedades

```
// 1. length: Devuelve la longitud de la cadena
let frase = "JavaScript es un lenguaje de programación";
console.log(`Longitud de la frase: ${frase.length}`);
```

Métodos

```
// 1. charAt(índice): Devuelve el carácter en la posición especificada por el índice.
let texto = "ABCDEFGHIJKLMN";
console.log(texto.charAt(5)); // Imprime "F" (carácter en la posición 5)

// 2. charCodeAt(índice): Devuelve el código Unicode del carácter en la posición especificada por el índice.
let letra = "ñ";
console.log(letra.charCodeAt(0)); // Imprime "165" (código Unicode de la letra "ñ")

// 3. indexOf(subcadena, inicio): Busca la primera aparición de una subcadena, devolviendo la posición de inicio.
let frase = "Esta frase contiene la palabra programación";
let indice = frase.indexOf("programación");
if (indice !== -1) {
  console.log(`programación está en la posición: ${indice}`);
} else {
  console.log("programación no se encuentra en la frase");
}

// 4. lastIndexOf(subcadena, inicio): Busca la última aparición de una subcadena, devolviendo la posición de inicio.
let texto2 = "Repetir la palabra palabra";
console.log(texto2.lastIndexOf("palabra")); // Imprime la posición de la última aparición de "palabra"

// 5. replace(subcadena, reemplazo): Reemplaza todas las apariciones de una subcadena por una cadena de reemplazo.
let texto3 = "Hola a todos, soy un robot";
let textoReemplazado = texto3.replace("robot", "humano");
console.log(textoReemplazado); // Imprime "Hola a todos, soy un humano"

// 6. toUpperCase() y toLowerCase(): Convierten la cadena a mayúsculas y minúsculas.
let saludo = "Hola Mundo";
console.log(saludo.toUpperCase()); // Imprime "HOLA MUNDO"
console.log(saludo.toLowerCase()); // Imprime "hola mundo"

// 7. trim(): Elimina los espacios en blanco al inicio y al final de la cadena.
let texto4 = "   Hola Mundo   ";
console.log(texto4.trim()); // Imprime "Hola Mundo"

// 8. concat(cadena2, ...): Concatena dos o más cadenas.
let nombre = "Juan";
let apellido = "Pérez";
let nombreCompleto = nombre.concat(" ", apellido);
console.log(nombreCompleto); // Imprime "Juan Pérez"

// 9. slice(inicio, fin): Extrae una subcadena desde una posición inicial hasta una posición final especificada.
let frase2 = "JavaScript es un lenguaje de programación";
let subcadena = frase2.slice(10, 25);
console.log(subcadena); // Imprime "es un lenguaje"

// 10. split(): Divide una cadena de texto en una lista de subcadenas, basado en un separador especificado.

        // Ejemplo 1: Dividir una cadena por espacios
        let frase3 = "Esta frase se divide por espacios";
        let palabras = frase3.split(" ");
        console.log(palabras); // Imprime ["Esta", "frase", "se", "divide", "por", "espacios"]

        // Ejemplo 2: Dividir una cadena por comas
        let lista = "rojo,verde,azul,amarillo";
        let colores = lista.split(",");
        console.log(colores); // Imprime ["rojo", "verde", "azul", "amarillo"]

        // Ejemplo 3: Dividir una cadena con un límite especificado
        let cadenaLarga = "Lorem ipsum dolor sit amet, consectetur adipiscing elit";
        let partes = cadenaLarga.split(" ", 5); // Límite de 5 subcadenas
        console.log(partes); // Imprime las primeras 5 palabras de la cadena

// Ejemplo 4: Obtener caracteres individuales
```

```
        let texto5 = "Hola, mundo!";
        let caracteres = texto5.split("");
        console.log(caracteres); // Imprime un array con cada carácter: ["H", "o", "l", "a", ",", " ", "m", "u",
"n", "d", "o", "!"]
```

Date

Creación

```
// Sin argumentos
let fechaActual = new Date();
console.log(fechaActual);

// con argumentos
let fechaEspecifica = new Date(2024, 5, 12, 10, 30, 15, 500);
console.log(fechaEspecifica);

// con cadena de fecha
let fechaCadena = "2024-06-12T10:30:15.500+02:00";
let fechaObjeto = new Date(fechaCadena);
console.log(fechaObjeto);
```

Métodos get

```
// 1. getFullYear(): Obtiene el año de la fecha como un número entero de cuatro dígitos.
let fecha = new Date();
let año = fecha.getFullYear();
console.log(`Año actual: ${año}`); // Ejemplo: 2024

// 2. getMonth(): Obtiene el mes de la fecha como un número entero de 0 a 11, donde 0 representa enero y 11
diciembre.
let mes = fecha.getMonth();
console.log(`Mes actual: ${mes + 1}`); // Ejemplo: 6 (junio)

// 3. getDate(): Obtiene el día del mes de la fecha como un número entero de 1 a 31.
let dia = fecha.getDate();
console.log(`Día actual: ${dia}`); // Ejemplo: 12

// 4. getHours(): Obtiene la hora de la fecha como un número entero de 0 a 23.
let hora = fecha.getHours();
console.log(`Hora actual: ${hora}`); // Ejemplo: 13

// 5. getMinutes(): Obtiene los minutos de la fecha como un número entero de 0 a 59.
let minutos = fecha.getMinutes();
console.log(`Minutos actuales: ${minutos}`); // Ejemplo: 26

// 6. getSeconds(): Obtiene los segundos de la fecha como un número entero de 0 a 59.
let segundos = fecha.getSeconds();
console.log(`Segundos actuales: ${segundos}`); // Ejemplo: 12

// 7. getMilliseconds(): Obtiene los milisegundos de la fecha como un número entero de 0 a 999.
let milisegundos = fecha.getMilliseconds();
console.log(`Milisegundos actuales: ${milisegundos}`);
```

Métodos set

```
// 1. setFullYear(año): Establece el año de la fecha en el valor entero especificado.
let fecha = new Date();
fecha.setFullYear(2023); // Establece el año a 2023
console.log(fecha.getFullYear()); // Imprime: 2023

// 2. setMonth(mes): Establece el mes de la fecha en el valor especificado (de 0 a 11, donde 0 representa enero y 11
diciembre).
fecha.setMonth(4); // Establece el mes a mayo (mes 4)
console.log(fecha.getMonth() + 1); // Imprime: 5 (mayo)

// 3. setDate(día): Establece el día del mes de la fecha en el valor entero especificado (de 1 a 31).
fecha.setDate(10); // Establece el día a 10
```

```
console.log(fecha.getDate()); // Imprime: 10

// 4. setHours(hora): Establece la hora de la fecha en el valor entero especificado (de 0 a 23).
fecha.setHours(18); // Establece la hora a 18
console.log(fecha.getHours()); // Imprime: 18

// 5. setMinutes(minutos): Establece los minutos de la fecha en el valor entero especificado (de 0 a 59).
fecha.setMinutes(30); // Establece los minutos a 30
console.log(fecha.getMinutes()); // Imprime: 30

// 6. setSeconds(segundos): Establece los segundos de la fecha en el valor entero especificado (de 0 a 59).
fecha.setSeconds(15); // Establece los segundos a 15
console.log(fecha.getSeconds()); // Imprime: 15

// 7. setMilliseconds(milisegundos): Establece los milisegundos de la fecha en el valor entero especificado (de 0 a
999).
fecha.setMilliseconds(500); // Establece los milisegundos a 500
console.log(fecha.getMilliseconds()); // Imprime: 500
```

Number

Propiedades

```
console.log(Number.MAX_VALUE); // Valor numérico más grande posible
console.log(Number.MIN_VALUE); // Valor numérico más pequeño posible
console.log(Number.POSITIVE_INFINITY); // Infinito positivo
console.log(Number.NEGATIVE_INFINITY); // Infinito negativo
console.log(Number.NaN); // Valor no numérico ("Not a Number")
```

Métodos

```
const numero = 123.4567;
console.log(numero.toString()); // Convierte a cadena de texto
console.log(numero.toFixed(2)); // "123.46" - Convierte a texto con 2 decimales
console.log(numero.toExponential(2)); // Notación exponencial: "1.23e+2"
console.log(numero.valueOf()); // Valor primitivo del número

// Métodos de comprobación:
console.log(Number.isFinite(numero)); // true - Comprueba si es finito
console.log(Number.isNaN('Hola')); // true - Comprueba si es NaN
console.log(Number.isInteger(numero)); // false - Comprueba si es un entero

// Convertir cadena a número:
const cadena = '100';
const numeroConvertido = Number(cadena);
console.log(numeroConvertido); // 100

// Crear un objeto Number con el constructor
const nuevoNumero = new Number(200);
console.log(nuevoNumero); // Imprime: Number {valueOf: 200}
```

Math

```
// Constantes de Math:
console.log(Math.PI); // Valor de pi (aproximadamente 3.14159)
console.log(Math.E); // Número de Euler (aproximadamente 2.71828)
console.log(Math.LN2); // Logaritmo natural de 2
console.log(Math.LN10); // Logaritmo natural de 10
console.log(Math.SQRT2); // Raíz cuadrada de 2
console.log(Math.SQRT1_2); // Raíz cuadrada de 1/2

// Funciones aritméticas:
console.log(Math.abs(-5)); // Valor absoluto: 5
console.log(Math.pow(2, 3)); // Potencia: 2^3 = 8
console.log(Math.sqrt(16)); // Raíz cuadrada: 4
console.log(Math.floor(4.7)); // Redondea hacia abajo: 4
console.log(Math.ceil(4.3)); // Redondea hacia arriba: 5
console.log(Math.round(4.5)); // Redondea al más cercano: 5
```

```
console.log(Math.random()); // Genera un número aleatorio entre 0 y 1

// Funciones trigonométricas:
console.log(Math.sin(Math.PI / 2)); // Seno de π/2: 1
console.log(Math.cos(Math.PI)); // Coseno de π: -1
console.log(Math.tan(0)); // Tangente de 0: 0
console.log(Math.asin(1)); // Arcoseno de 1: π/2
console.log(Math.acos(0)); // Arcocoseno de 0: π/2
console.log(Math.atan(1)); // Arcotangente de 1: π/4
console.log(Math.atan2(1, 1)); // Arcotangente de 1/1: π/4

// Funciones de logaritmos:
console.log(Math.log(10)); // Logaritmo natural de 10
console.log(Math.log10(100)); // Logaritmo base 10 de 100: 2
console.log(Math.log2(8)); // Logaritmo base 2 de 8: 3
console.log(Math.exp(1)); // Exponencial de 1 (e^1): Math.E

// Funciones de redondeo:
console.log(Math.round(4.5)); // Redondea al más cercano: 5
console.log(Math.floor(4.7)); // Redondea hacia abajo: 4
console.log(Math.ceil(4.3)); // Redondea hacia arriba: 5

// Funciones de mínimo y máximo:
console.log(Math.max(1, 3, 2)); // Máximo: 3
console.log(Math.min(1, 3, 2)); // Mínimo: 1
```

Almacenamiento local

Cookies

```
// Crear cookies
document.cookie = "nombre=Pepe";
document.cookie = "edad=30";
document.cookie = "ciudad=Valencia";
alert("Cookies guardadas correctamente.");
console.log(document.cookie); // Mostrar todas las cookies

// Configuración de propiedades importantes
// 1. path: Accesibilidad de la cookie en todas las páginas
document.cookie = "ciudad=Valencia; path=/";

// 2. expires y max-age: Controlan la duración de la cookie
// Expira cuando el navegador se cierra si no se establece
// Explicación de expires
let date = new Date(Date.now() + 86400e3); // +1 día
document.cookie = "ciudad=Valencia; expires=" + date.toUTCString();

// Eliminar una cookie estableciendo expires en el pasado
document.cookie = "ciudad=Valencia; expires=Thu, 01 Jan 1970 00:00:00 GMT";

// Explicación de max-age
document.cookie = "nombre=Pepe; max-age=3600"; // Expira en 1 hora
document.cookie = "nombre=Pepe; max-age=0"; // Eliminar cookie inmediatamente
```

LocalStorage

```
// Almacenamiento local: persistente y basado en pares clave-valor

// 1. Almacenar datos
localStorage.setItem("nombre", "Juan Pérez");
localStorage.setItem("edad", 30);

// 2. Recuperar datos
let nombre = localStorage.getItem("nombre");
console.log(nombre); // Imprime: "Juan Pérez"
let edad = localStorage.getItem("edad");
console.log(edad); // Imprime: 30

// 3. Eliminar datos
```

```
localStorage.removeItem("nombre");
console.log(localStorage.getItem("nombre")); // Imprime: null
localStorage.removeItem("edad");
console.log(localStorage.getItem("edad")); // Imprime: null

// 4. Limpiar todo el almacenamiento local
localStorage.clear();
console.log(localStorage.getItem("nombre")); // Imprime: null
console.log(localStorage.getItem("edad")); // Imprime: null
```

Eventos

Controladores de eventos

```
let miBoton = document.getElementById("miBoton");

// Evento con DOM
miBoton.onclick=saludar;
function saludar() {alert("Hola")}; // tradicional
miBoton.onclick=function() {alert("Hola")}; // función anónima
miBoton2.onclick=()=>alert ("Hola...") // fatArrow

// Evento con Listener
miBoton.addEventListener("click", saludar); // añadir evento
miBoton.removeEventListener("click", saludar) // eliminar evento
```

Objeto Event

```
// Event se utiliza para manejar eventos
// event.type = indica el evento
// event.currentTarget = elemento que disparó el evento
// event.clientX || event.clientY = coordenadas del cursor en la ventana
let miBoton = document.getElementById("miBoton");
miBoton.onclick = function(event) {
    alert("Evento " + event.type + " en " + event.currentTarget);
    alert("Coordenadas: " + event.clientX + ":" + event.clientY);
};
```

Objeto Onload

```
// cuando se cargue la página lanzamos X evento
// DOM
function saludar() {
    console.log("La página ha sido cargada usando window.onload.");
}
window.onload = saludar;

window.onload = function() { // función anonima
    console.log("La página ha sido cargada con una función anónima.");
};

window.onload = () => { // fatArrow
    console.log("La página ha sido cargada con una función arrow.");
};

// addEventListener
window.addEventListener("load", function() {
    console.log("La página ha sido cargada con addEventListener.");
});
```

Eventos personalizados

```
const myEvent = new CustomEvent('myCustomEvent', { // crear evento
    detail: {
```

```
            message: 'Hola desde el evento personalizado',
            data: 42
        }
});

let miBoton = document.getElementById("miBoton"); // asocia a elemto HTML una variable
miBoton.addEventListener("click", () => window.dispatchEvent(myEvent)); // lanzamos evento
window.addEventListener('myCustomEvent', (event) => {
    console.log(event.detail.message); // "Hola desde el evento personalizado"
});
```

DOM

DOM HTML

```
// ********************************************************************************
// Encontrar elementos HTML

document.getElementById("id");              // Encuentra un elemento por su ID
document.getElementsByTagName("name");      // Encuentra elementos por etiqueta HTML
document.getElementsByClassName("name");    // Encuentra elementos por clase CSS
document.querySelector("css");              // Selecciona el primer elemento que coincida con el selector CSS
document.querySelectorAll("css");           // Selecciona todos los elementos que coincidan con el selector CSS

// ********************************************************************************
// Cambiar elementos HTML

// Propiedades
element.innerHTML = "nuevo contenido";     // Cambia el contenido HTML de un elemento
element.attribute = "nuevo valor";         // Cambia el valor de un atributo de un elemento HTML
element.style.property = "nuevo estilo";   // Cambia el estilo de un elemento HTML

// Métodos
element.setAttribute("attribute", "value"); // Cambia el valor de un atributo de un elemento HTML
// Nota: `setAttribute` es un método que requiere el nombre y valor del atributo como argumentos.

// ********************************************************************************
// Añadir y eliminar elementos

document.createElement("element");         // Crea un nuevo elemento HTML
document.removeChild(element);             // Elimina un elemento HTML
document.appendChild(element);             // Añade un elemento HTML como hijo de otro
document.replaceChild(newElement, oldElement); // Reemplaza un elemento HTML por otro
document.write("texto");                   // Escribe texto o HTML directamente en la página

// ********************************************************************************
// Agregar controladores de eventos

document.getElementById("id").onclick = function() {
  // Código para gestionar el evento `onclick` del elemento con el ID especificado
};

// ********************************************************************************
// Encontrar objetos HTML

// HTML DOM Nivel 1 (1998) define 11 objetos HTML que siguen válidos en HTML5
document.anchors;      // Anclas en el documento
document.body;         // Cuerpo del documento
document.cookie;       // Cookies del documento
document.doctype;      // Tipo de documento
document.forms;        // Formularios en el documento
document.images;       // Imágenes en el documento
document.links;        // Enlaces en el documento
document.readyState;   // Estado de carga del documento
```

Nodos DOM

Propiedades

```
// Propiedad `nodeName`: especifica el nombre de un nodo.
let elemento = document.getElementById("h1");
alert(elemento.nodeName); // Muestra "H1"

// - `nodeName` es de solo lectura.
// - Para un nodo de elemento, `nodeName` es el nombre de la etiqueta en MAYÚSCULAS.
// - Para un nodo de atributo, `nodeName` es el nombre del atributo.
// - Para un nodo de texto, `nodeName` siempre es "#text".
// - Para el nodo de documento, `nodeName` siempre es "#document".

// ****************************************************************************
// Propiedad `nodeValue`: especifica el valor de un nodo.
console.log(elemento.nodeValue);          // `null` para nodos de elemento
console.log(elemento.firstChild.nodeValue); // Texto de un nodo de texto

// - Para elementos de nodo, `nodeValue` es `null`.
// - Para nodos de texto, `nodeValue` es el texto en sí.
// - Para nodos de atributo, `nodeValue` es el valor del atributo.

// ****************************************************************************
// Propiedad `nodeType`: devuelve el tipo de un nodo. Es de solo lectura.

console.log(elemento.nodeType); // Devuelve 1 para ELEMENT_NODE

// Tipos de nodos más comunes:
const nodeTypes = {
  ELEMENT_NODE: 1,       // <h1 class="heading">Titulo</h1>
  ATTRIBUTE_NODE: 2,     // class = "heading" (obsoleto)
  TEXT_NODE: 3,          // Texto de un elemento
  COMMENT_NODE: 8,       // <!-- This is a comment -->
  DOCUMENT_NODE: 9,      // El documento HTML mismo (padre de <html>)
  DOCUMENT_TYPE_NODE: 10 // <!Doctype html>
};
```

Propiedades de relación entre nodos

```
// Propiedades para navegar entre nodos
let nodoEjemplo = document.getElementById("p1");

let nodoPadre = nodoEjemplo.parentNode;             // Nodo padre de un nodo
let nodoHijos = nodoEjemplo.childNodes;             // Colección de hijos de un nodo
let primerHijo = nodoEjemplo.firstChild;            // Primer hijo de un nodo
let ultimoHijo = nodoEjemplo.lastChild;             // Último hijo de un nodo
let siguienteHermano = nodoEjemplo.nextSibling;     // Siguiente nodo hermano
let anteriorHermano = nodoEjemplo.previousSibling;  // Anterior nodo hermano

// ****************************************************************************
// Nodos secundarios y valores de nodo
// Un nodo de elemento no contiene texto directamente, sino un nodo de texto con el valor.

<p id="p1">Tutorial DOM</p>;

let miP1 = document.getElementById("p1").innerHTML;           // "Tutorial DOM"
let miP2 = document.getElementById("p1").firstChild.nodeValue; // "Tutorial DOM"
let miP3 = document.getElementById("p1").childNodes[0].nodeValue; // "Tutorial DOM"

// Ejemplo: copiar el texto de un <h1> a un <p> en tres formas
<h1 id="h1">Mi pagina</h1>
<p id="p1"></p>

<script>

document.getElementById("p1").innerHTML = document.getElementById("h1").innerHTML;
document.getElementById("p1").innerHTML=document.getElementById("h1").firstChild.nodeValue;

document.getElementById("p1").innerHTML=document.getElementById("h1").childNodes[0].nodeValue;

</script>
```

```
// Nota: `innerHTML` es un método rápido para acceder al contenido HTML, pero conocer otros métodos ayuda a entender la estructura de árbol y navegación del DOM.

// Para nodos de texto, también se puede acceder al contenido mediante `textContent`.
let textoNodo = document.getElementById("demo").textContent;
```

Encontrar elementos HTML por colecciones de objetos HTML

```
let misImagenes= document.images // podemos acceder a todas las imagenes del documento
let misLinks= document.links // podemos acceder a todos las links del documento
let misForms=document.forms // podemos acceder a todos las formularios del documento
```

Modificar elementos

```
<!-- ATRIBUTO -->
<img id="miImagen" src="imagen1.gif">

<script>
        document.getElementById("miImagen").src = "imagen2.jpg";
</script>
```

```
<!-- ESTILO -->
<p id="p1">Hola mundo!</p>

<script>
        document.getElementById("p1").style.color = "blue";
</script>
```

Operaciones

crear

```
let imagen = document.createElement("img");  // Crear nodo <img>
imagen.id = "mi_foto";  // Asignar id
imagen.src = "./fotos/logo.jpg";  // Asignar src para definir la imagen

// Añadir imagen al contenedor 'fotos'
let divFotos = document.getElementById("fotos");
divFotos.appendChild(imagen);  // Insertar nodo en el DOM

// Ejemplo con nodo de texto
let miParrafo = document.createElement("p");  // Crear nodo <p>
let miTexto = document.createTextNode("Texto del parrafo.");  // Crear nodo de texto
miParrafo.appendChild(miTexto);  // Agregar texto al <p>

// Añadir párrafo al contenedor 'div1'
let padre = document.getElementById("div1");
padre.appendChild(miParrafo);  // Insertar nodo en el DOM

// textContent para acceder o cambiar texto sin crear nodo de texto
const parrafo = document.getElementById("miParrafo");
console.log(parrafo.textContent);  // Obtener contenido de texto
parrafo.textContent = "Nuevo contenido para el párrafo.";  // Cambiar contenido

// insertBefore para añadir en una posición específica
let miparrafo = document.createElement("p");
let miTexto2 = document.createTextNode("Parrafo nuevo");
miparrafo.appendChild(miTexto2);
let elementoHijo = document.getElementById("p1");  // Nodo de referencia
padre.insertBefore(miparrafo, elementoHijo);  // Insertar antes de elementoHijo
```

eliminar

```
let parrafoEliminar = document.getElementById("p1");
parrafoEliminar.remove();  // Eliminar nodo directamente
```

substituir

```
let nuevoParrafo = document.createElement("p");
let nodoTexto = document.createTextNode("Parrafo nuevo");
nuevoParrafo.appendChild(nodoTexto);
let child = document.getElementById("p1");  // Nodo a reemplazar
padre.replaceChild(nuevoParrafo, child);  // Reemplazar nodo
```

clonar

```
// Método cloneNode(deep): deep = true copia el nodo y todos los descendientes
let clon = document.getElementById("miParrafo").cloneNode(true);
```

Asignar eventos utilizando HTML DOM

```
// Asignación de evento onclick a un botón existente para mostrar la hora actual
document.getElementById("btn1").onclick = displayDate;

function displayDate() {
    document.getElementById("p1").innerHTML = Date();  // Muestra la fecha y hora actuales en el párrafo
}

// Crear un nuevo botón y asignarle un evento onclick usando una función anónima
let miBoton = document.createElement("button");        // Crear nodo <button>
miBoton.innerHTML = "Púlsame...";                      // Asignar texto al botón

miBoton.onclick = function() {                         // Asignar función anónima al evento onclick
    alert(this.innerHTML);                             // Muestra el texto del botón en un alert
};

document.body.appendChild(miBoton);                    // Añadir el botón al body
```

Formularios

Acceder a elementos

```
const nombreInput = document.getElementById('nombre');
const emailInput = document.querySelector('input[name="correo"]');
const formulario = document.forms['registro'];
const botonEnviar = formulario.elements['enviar'];
```

Gestión de eventos

```
// 1. Evento 'change': Se dispara cuando cambia el valor de un campo
// En este caso, comprobamos el valor escrito en el campo 'nombre'
const nombreInput = document.getElementById('nombre');
nombreInput.addEventListener('change', function() {
    const nombre = nombreInput.value;
    console.log('El nombre ha cambiado a:', nombre);
});

// 2. Evento 'focus' y 'blur': Enfoque y desenfoque de un campo
// Podemos mostrar mensajes cuando el campo recibe o pierde el foco
nombreInput.addEventListener('focus', function() {
    console.log('El campo de nombre ha recibido el foco');
});

nombreInput.addEventListener('blur', function() {
    console.log('El campo de nombre ha perdido el foco');
});

// 3. Evento 'submit': Envío del formulario
// Se usa generalmente para validar y enviar datos
const formulario = document.getElementById('miFormulario');
```

```
formulario.addEventListener('submit', function(event) {
    event.preventDefault(); // Evita el envío del formulario por defecto
    // Validar los datos del formulario aquí
    if (nombreInput.value === '') {
        console.log('El campo de nombre está vacío');
        return; // Detener el proceso si el campo está vacío
    }
    // Si los datos son válidos, se podría enviar el formulario aquí
    console.log('Formulario enviado con éxito');
});
```

Submit

```
// Manejador de evento submit para validación previa
formulario.addEventListener('submit', function(event) {
    event.preventDefault(); // Evita el envío por defecto
    // Validación de los datos
    if (nombreInput.value === '') {
        console.log('Por favor, complete el campo de nombre');
    } else {
        console.log('Datos del formulario válidos, listos para enviar');
    }
});
```

Validación

```
// Función para validar un correo electrónico usando una expresión regular
function validarEmail(email) {
    const regex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/; // Patrón de regex básico para correos
    return regex.test(email); // Devuelve true si el email es válido
}

// Evento para validar el correo al perder el foco en el campo de email
const emailInput = document.getElementById('email');
emailInput.addEventListener('blur', function() {
    const email = emailInput.value;
    if (!validarEmail(email)) {
        alert('El correo electrónico no es válido.');
    }
});
```

```
// Validación del campo Nombre y Email
function validarFormulario() {
    let errores = false;

    // Validación del nombre
    const nombreInput = document.getElementById('nombre');
    const nombreError = document.getElementById('nombreError');
    if (nombreInput.value === '' || nombreInput.value.length < 5) {
        nombreError.style.display = 'block';
        nombreError.textContent = 'El nombre debe tener al menos 5 caracteres';
        errores = true;
    } else {
        nombreError.style.display = 'none';
    }

    // Validación del correo electrónico
    const emailInput = document.getElementById('email');
    const emailError = document.getElementById('emailError');
    if (emailInput.value === '' || !validarEmail(emailInput.value)) {
        emailError.style.display = 'block';
        emailError.textContent = 'Ingrese un correo electrónico válido';
        errores = true;
    } else {
        emailError.style.display = 'none';
    }

    // Si no hay errores, mostrar un mensaje de éxito
```

```
    if (!errores) {
        alert('Los datos se han enviado con éxito.');
    }
    return !errores; // Devuelve false si hay errores para evitar el envío
}

// Manejador del evento 'submit' del formulario
let formulario = document.getElementById('miFormulario');
formulario.addEventListener('submit', function(event) {
    event.preventDefault(); // Evita el envío del formulario si hay errores
    validarFormulario();
    //formulario.submit();
});
```

Controles

Cuadro de texto y Textarea

```
// Obtener valores de un input y un textarea
const valorText = document.getElementById("texto").value;
const valorTextArea = document.getElementById("parrafo").value;
```

Radio button

```
// Obtener el estado de selección de cada radio button
let elementos = document.getElementsByName("estadoCivil");
Array.from(elementos).forEach(ec => console.log(`${ec.value} es ${ec.checked ? 'seleccionado' : 'no
seleccionado'}`));
```

Checkbox

```
// Verificar la selección de checkboxes
function verCondicionesPrivacidad() {
    const condiciones = document.getElementById("condiciones").checked ? " se ha aceptado" : " no se ha aceptado";

    const privacidad = document.getElementById("privacidad").checked ? " se ha aceptado" : " no se ha aceptado";

    console.log(`Condiciones: ${condiciones}, Privacidad: ${privacidad}`);
}
```

Select (lista desplegable)

```
// Obtener el valor seleccionado de un select
let lista = document.getElementById("lista");
console.log("Valor seleccionado:", lista.value);

lista.addEventListener("change", () => console.log("Nuevo valor:", lista.value));

// Acceder a una opción específica del select
let indiceSeleccionado = lista.selectedIndex;
let opcionSeleccionada = lista.options[indiceSeleccionado];
console.log("Texto seleccionado:", opcionSeleccionada.text);
console.log("Valor seleccionado:", opcionSeleccionada.value);
```

Añadir y eliminar elementos de select

```
// Añadir una opción a la lista
function anyadirElementoLista() {
    let texto = prompt("Dime Texto");
    let valor = prompt("Dime Valor");
    let mioption = document.createElement("option");
    mioption.value = valor;
    mioption.text = texto;
    lista.appendChild(mioption);
}
```

```
// Eliminar una opción de la lista
function eliminarElementoLista(indice) {
    if (lista.options.length > 0) {
        lista.remove(indice);
    }
}
```

ejemplo

```
// Ejemplo de manipulación de controles en formulario
document.getElementById("submitBtn").addEventListener("click", function(event) {
    event.preventDefault();

    // Cuadro de texto y textarea
    let nombre = document.getElementById("nombre").value;
    let mensaje = document.getElementById("mensaje").value;
    console.log("Nombre:", nombre, "Mensaje:", mensaje);

    // Radio Button
    let estadoCivilSeleccionado = Array.from(document.getElementsByName("estadoCivil"))
        .find(rb => rb.checked);
    console.log("Estado Civil Seleccionado:", estadoCivilSeleccionado ? estadoCivilSeleccionado.value : "Ninguno");

    // Checkbox
    let aceptaCondiciones = document.getElementById("condiciones").checked;
    console.log("Acepta Condiciones:", aceptaCondiciones);

    // Select
    let lista = document.getElementById("lista");
    let valorSeleccionado = lista.value;
    console.log("Valor Seleccionado de Lista:", valorSeleccionado);
});
```

API-REST (JSON Server)

```
npm install -g json-server ## instalar
json-server --watch db.json ## inicializar API en -> http://localhost:3000
```

Comunicación Asíncrona (Promesas)

```
// Básico: GET con .then() y .catch()
fetch('https://api.example.com/entidades')
  .then(response => response.json())
  .then(data => console.log(data))
  .catch(error => console.error('Error:', error));

// DELETE
fetch('https://api.example.com/entidades/1', { method: 'DELETE' })
  .then(response => {
    if (response.ok) console.log('Elemento eliminado');
    else console.error('Error al eliminar');
  })
  .catch(error => console.error('Error:', error));

// POST (Enviar datos al servidor)
fetch('https://api.example.com/entidades', {
  method: 'POST',
  headers: { 'Content-Type': 'application/json' },
  body: JSON.stringify({ nombre: 'Nueva Entidad', activo: true }),
})
  .then(response => response.json())
  .then(data => console.log('Creado:', data))
  .catch(error => console.error('Error:', error));
```

```
// PUT (Actualizar datos completos)
fetch('https://api.example.com/entidades/1', {
  method: 'PUT',
  headers: { 'Content-Type': 'application/json' },
  body: JSON.stringify({ nombre: 'Entidad Actualizada', activo: false }),
})
  .then(response => response.json())
  .then(data => console.log('Actualizado:', data))
  .catch(error => console.error('Error:', error));

// PATCH (Actualizar parcialmente)
fetch('https://api.example.com/entidades/1', {
  method: 'PATCH',
  headers: { 'Content-Type': 'application/json' },
  body: JSON.stringify({ activo: true }),
})
  .then(response => response.json())
  .then(data => console.log('Parcialmente Actualizado:', data))
  .catch(error => console.error('Error:', error));

// Encadenamiento de promesas + Refactorización
function manejarRespuesta(response) {
  if (!response.ok) throw new Error('Error en la respuesta');
  return response.json();
}

fetch('https://api.example.com/entidades')
  .then(manejarRespuesta)
  .then(data => {
    console.log('Primera respuesta:', data);
    return fetch(`https://api.example.com/entidades/${data[0].id}`);
  })
  .then(manejarRespuesta)
  .then(data => console.log('Segunda respuesta:', data))
  .catch(error => console.error('Error:', error));

// Función Genérica para GET
function getEntidad(url) {
  return fetch(url)
    .then(response => {
      if (!response.ok) throw new Error('Error al obtener entidad');
      return response.json();
    });
}

// Uso de getEntidad
getEntidad('https://api.example.com/entidades')
  .then(data => console.log(data))
  .catch(error => console.error('Error:', error));

// Async / Await
async function fetchConAsync() {
  try {
    const response = await fetch('https://api.example.com/entidades');
    if (!response.ok) throw new Error('Error al obtener datos');
    const data = await response.json();
    console.log(data);
  } catch (error) {
    console.error('Error:', error);
  }
}

// Ejemplo: Petición Encadenada para Datos de Artículo de Proveedor
async function obtenerArticuloProveedor(proveedorId) {
  try {
    const proveedor = await getEntidad(`https://api.example.com/proveedores/${proveedorId}`);
    const articulo = await getEntidad(`https://api.example.com/articulos/${proveedor.articuloId}`);
    console.log('Proveedor:', proveedor);
    console.log('Artículo:', articulo);
  } catch (error) {
    console.error('Error:', error);
```

```
    }
}
```
