

Introducción al Desarrollo web

Curso: Desarrollo Frontend con React

Funciones, Eventos y Manipulación del DOM en JavaScript

Un recorrido por los conceptos fundamentales para interactuar con páginas web desde JavaScript



Agenda



1 Funciones en JavaScript

Declaración, uso, parámetros y retorno



2 Funciones flecha

Sintaxis moderna y ventajas



3 Eventos

Interacción con usuarios



4 DOM

Estructura y manipulación



5 Ejemplos prácticos

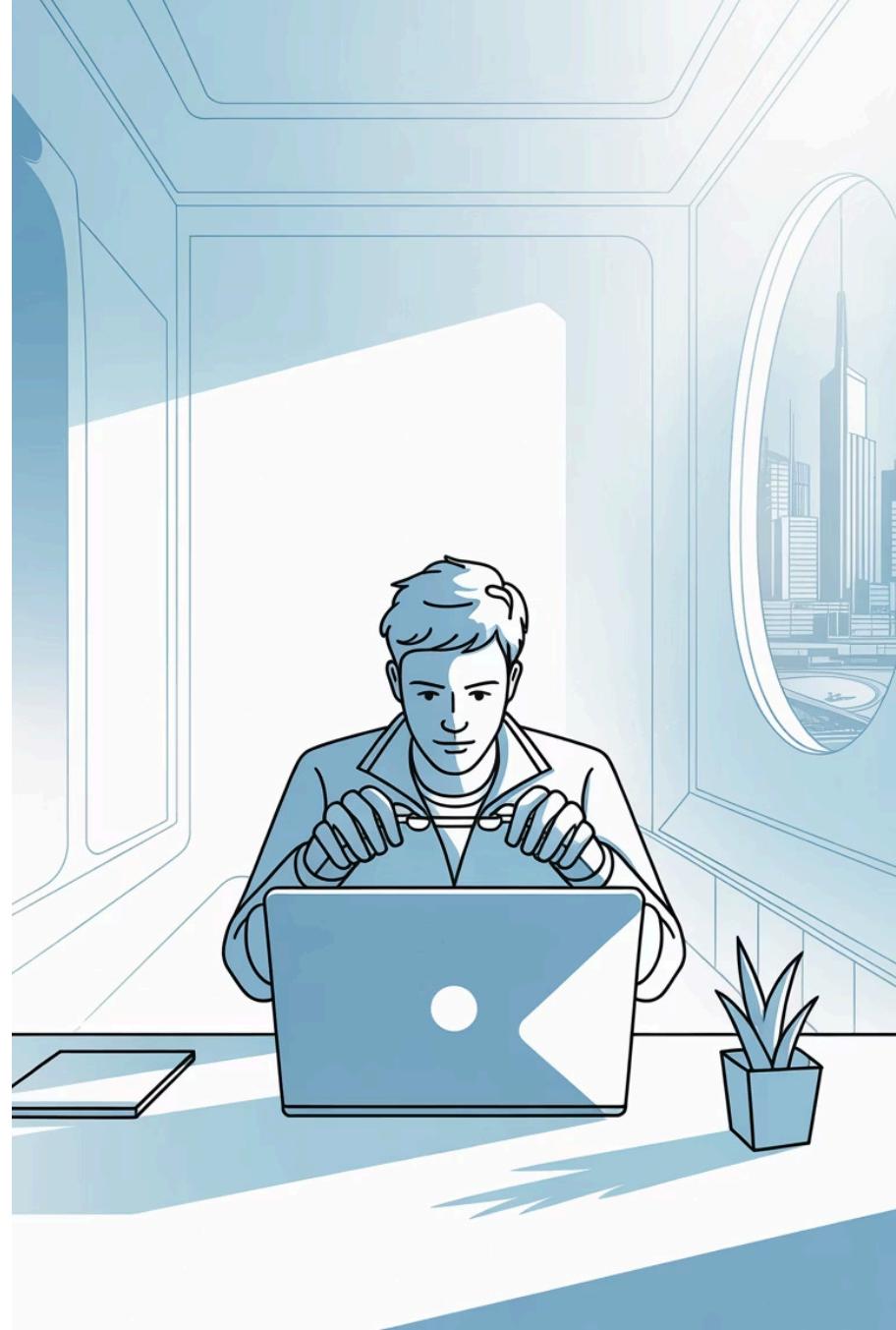
Calculadora básica

Funciones en JavaScript

Las funciones son **bloques de código reutilizables** que realizan una tarea específica.

Ventajas:

- Evitan repetir código
- Mejoran la organización y mantenimiento
- Facilitan la prueba de pequeñas partes del programa



Declaración de Funciones Tradicionales

```
// Declaración de función
function saludar() {
  console.log("¡Hola, estudiante!");
}

// Llamada a la función
saludar();

// Función con parámetros
function sumar(a, b) {
  return a + b; // Valor de retorno
}

let resultado = sumar(5, 3); // resultado = 8
```

Las funciones pueden declararse y luego utilizarse en cualquier parte del código (**hoisting**).

Parámetros y Valores de Retorno

Parámetros

- Valores que recibe la función
- Pueden tener valores por defecto
- Parámetros REST (...args)

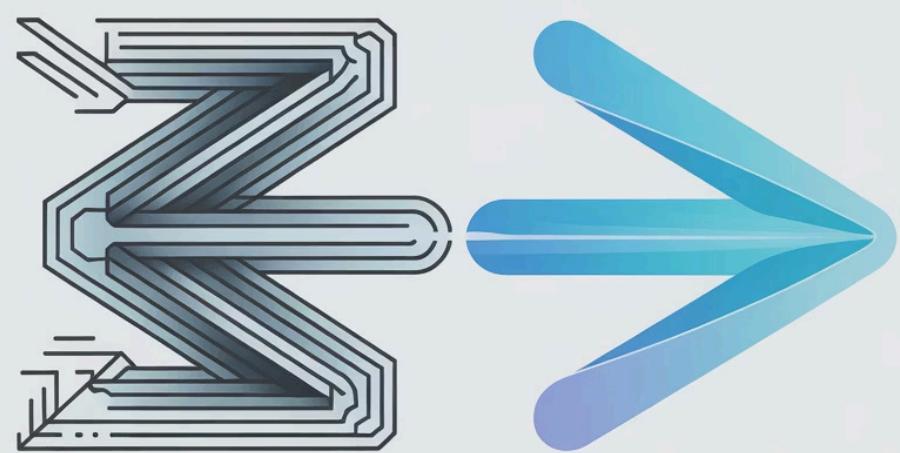
```
function saludar(nombre = "alumno") {  
  console.log(`Hola, ${nombre}`);  
}  
  
saludar(); // "Hola, alumno"  
saludar("María"); // "Hola, María"
```

Valores de retorno

- Resultado que devuelve la función
- Se obtienen con la palabra `return`
- La función termina al ejecutar `return`

```
function calcularIVA(precio) {  
  return precio * 0.21;  
}
```

```
let iva = calcularIVA(100); // 21
```



Funciones Flecha

Las funciones flecha (**arrow functions**) son una forma más concisa de escribir funciones, introducida en ES6.

```
// Función tradicional  
function sumar(a, b) {  
    return a + b;  
}
```

```
// Función flecha equivalente  
const sumar = (a, b) => a + b;
```

Las funciones flecha no tienen su propio `this`, lo heredan del contexto en el que se declaran.

Sintaxis de Funciones Flecha

Con un solo parámetro

```
// Sin paréntesis en el parámetro  
const cuadrado = x => x * x;  
  
console.log(cuadrado(5)); // 25
```

Sin parámetros

```
// Paréntesis vacíos obligatorios  
const saludar = () => "¡Hola!";  
  
console.log(saludar()); // "¡Hola!"
```

Con cuerpo de varias líneas

```
// Requiere llaves y return explícito  
const calcular = (a, b) => {  
    const suma = a + b;  
    const producto = a * b;  
    return { suma, producto };  
};
```

```
console.log(calcular(3, 4));  
// {suma: 7, producto: 12}
```

Comparativa: Funciones tradicionales vs. Flecha



Sintaxis

Las funciones flecha son más concisas, especialmente para funciones simples.

```
// Tradicional  
function suma(a, b) { return a + b; }  
  
// Flecha  
const suma = (a, b) => a + b;
```



this

Las funciones flecha no tienen su propio this, lo heredan del contexto exterior.

Esto evita problemas comunes en callbacks y métodos.



Constructor

Las funciones flecha no pueden usarse como constructores (con new).

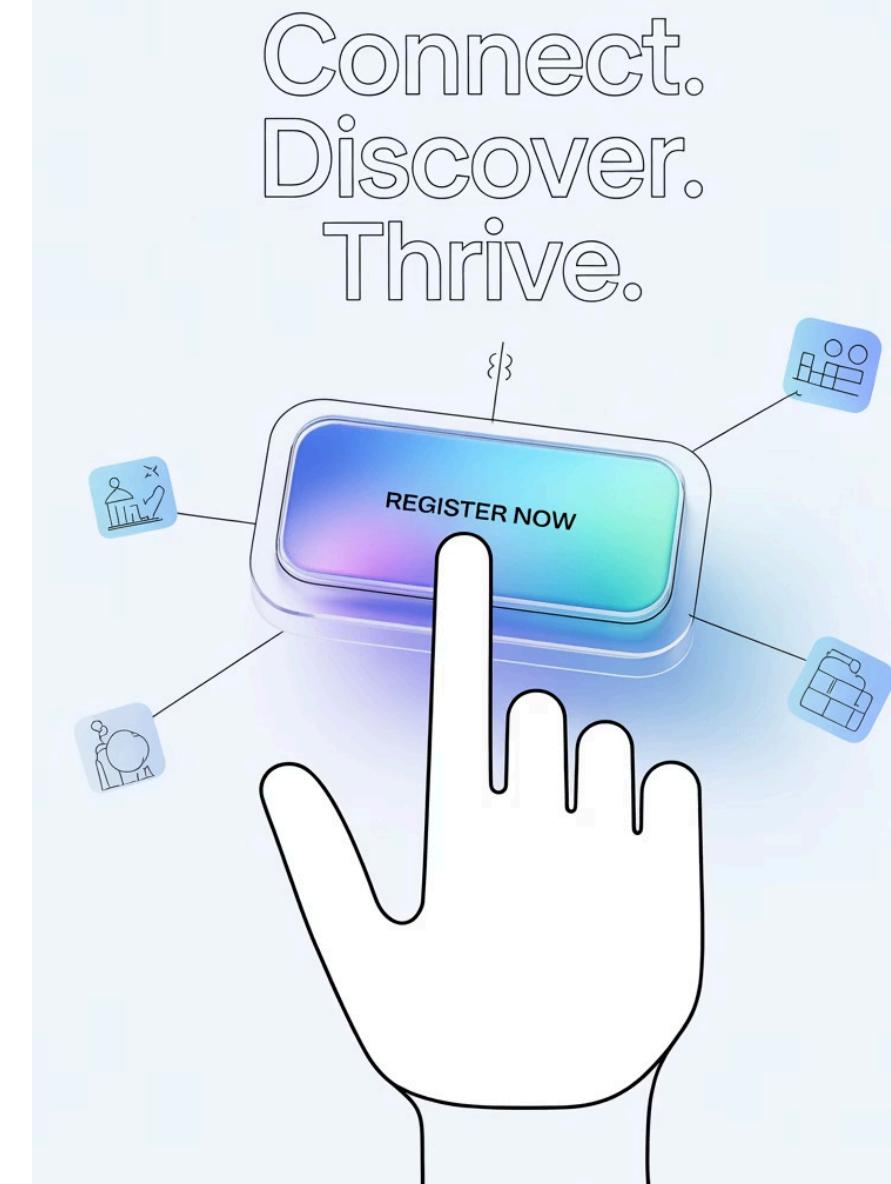
No tienen el objeto prototype.

Eventos en JavaScript

Los [eventos](#) son acciones o sucesos que ocurren en el navegador:

- Clics de ratón
- Pulsaciones de teclas
- Carga de páginas
- Cambios en formularios
- Movimientos del ratón

JavaScript permite **escuchar** estos eventos y **responder** a ellos.



Manejadores de Eventos

Método 1: Atributos HTML

```
<button onclick="alert('¡Hola!')">  
    Saluda  
</button>
```

Simple pero no recomendado para código complejo.

Método 2: Propiedades del DOM

```
const boton = document.getElementById('miBoton');  
boton.onclick = function() {  
    alert('¡Hola desde JavaScript!');  
};
```

Mejor separación del HTML, pero solo permite un manejador por evento.

Método 3: addEventListener (recomendado)

```
boton.addEventListener('click', function() {  
    alert('¡Hola con addEventListener!');  
});
```

Permite múltiples manejadores para un mismo evento y más control.

Eventos Comunes en JavaScript

Eventos de ratón

- click: Al hacer clic
- dblclick: Doble clic
- mouseover: Al pasar por encima
- mouseout: Al salir del elemento

Eventos de teclado

- keydown: Al presionar una tecla
- keyup: Al soltar una tecla
- keypress: Al mantener pulsada

Eventos de formulario

- submit: Al enviar un formulario
- change: Al cambiar valor de input
- focus: Al recibir el foco
- blur: Al perder el foco

Eventos de documento

- load: Al cargar la página
- DOMContentLoaded: Al cargar el DOM
- resize: Al cambiar tamaño ventana

Ejemplo de addEventListener

```
// HTML: <button id="miBoton">Haz clic</button>

const boton = document.getElementById('miBoton');

// Primer manejador
boton.addEventListener('click', function() {
  console.log('Primer manejador de clic');
});

// Segundo manejador (¡ambos funcionarán!)
boton.addEventListener('click', function() {
  console.log('Segundo manejador de clic');
});

// Con función flecha
boton.addEventListener('mouseover', () => {
  boton.style.backgroundColor = 'yellow';
});
```

codC\ "e
"addeveenepe''>
"addevevntlistener > >
addeveentili cntgel >>
eddeveveejt' cype
event yec'\>

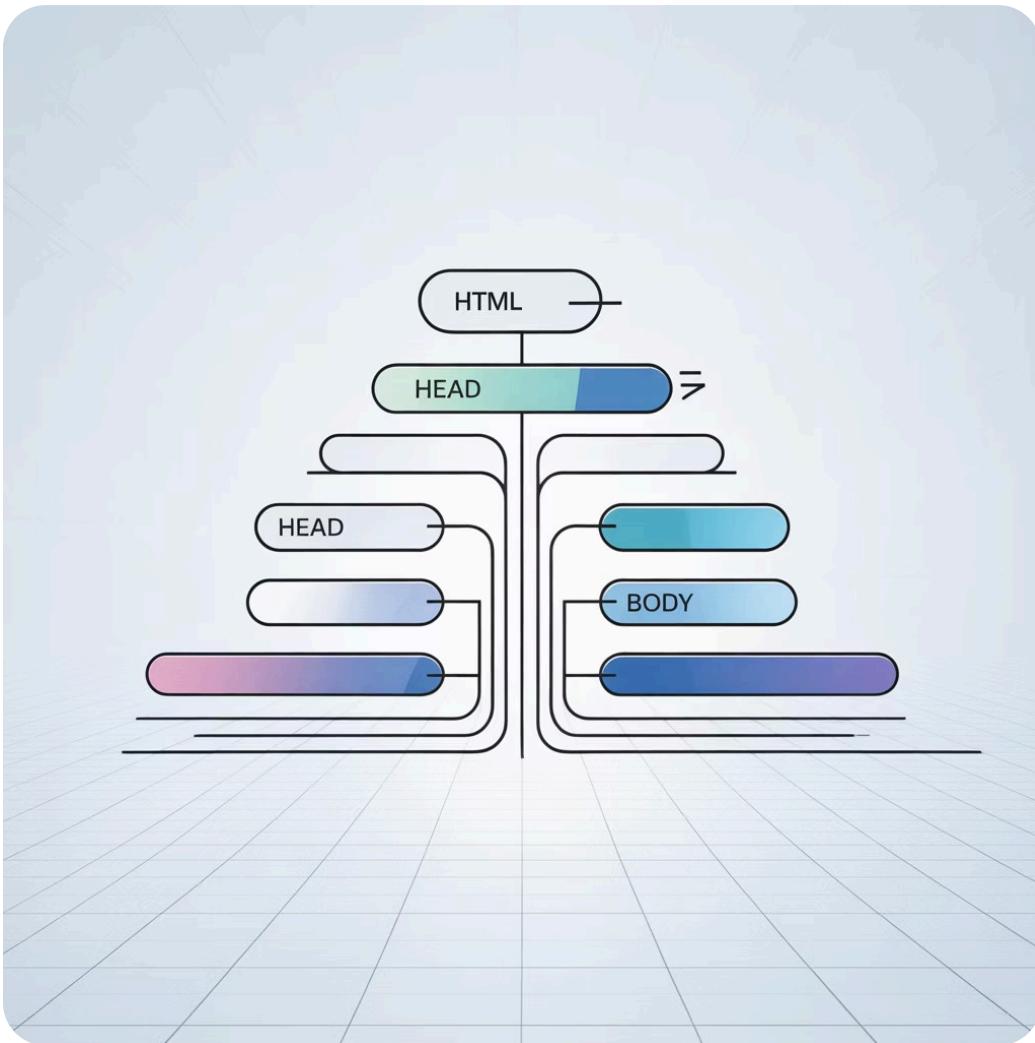
¿Qué es el DOM?

El [Document Object Model](#) (DOM) es una representación estructurada del documento HTML como un **árbol de objetos**.

Permite a JavaScript:

- Acceder a cualquier elemento de la página
- Modificar contenido, estructura y estilos
- Reaccionar a eventos del usuario
- Crear efectos dinámicos e interactividad

Estructura del DOM



El DOM organiza los elementos HTML en una estructura jerárquica de **nodos**:

- Documento: nodo raíz
- Elementos: etiquetas HTML
- Atributos: propiedades de elementos
- Texto: contenido textual
- Comentarios: comentarios HTML

Cada nodo es un **objeto** con propiedades y métodos que permiten manipularlo.

Selección de Elementos del DOM

Métodos tradicionales

```
// Seleccionar por ID  
const titulo = document.getElementById('titulo');
```

```
// Seleccionar por clase (colección)  
const parrafos =  
document.getElementsByClassName('parrafo');
```

```
// Seleccionar por etiqueta (colección)  
const botones =  
document.getElementsByTagName('button');
```

Métodos modernos (recomendados)

```
// Primer elemento que coincide con el selector  
const elemento = document.querySelector('#titulo');
```

```
// Todos los elementos que coincidan (NodeList)  
const items = document.querySelectorAll('.item');
```

```
// Combinando selectores (como en CSS)  
const botonActivo = document.querySelector('button.activo');
```

querySelector y querySelectorAll

Estos métodos usan [selectores CSS](#) para encontrar elementos:

Seleccionar por ID

```
document.querySelector('#miElemento');
```

Seleccionar por clase

```
document.querySelectorAll('.miClase');
```

Selectores combinados

```
// Primer input dentro de un form con clase 'registro'  
document.querySelector('form регистра input');
```

```
// Todos los párrafos con clase 'importante'  
document.querySelectorAll('p.importante');
```

Selectores de atributos

```
// Elementos con atributo data-tipo="usuario"  
document.querySelectorAll('[data-tipo="usuario"]');
```

```
// Links que abren en nueva ventana  
document.querySelectorAll('a[target="_blank"]');
```

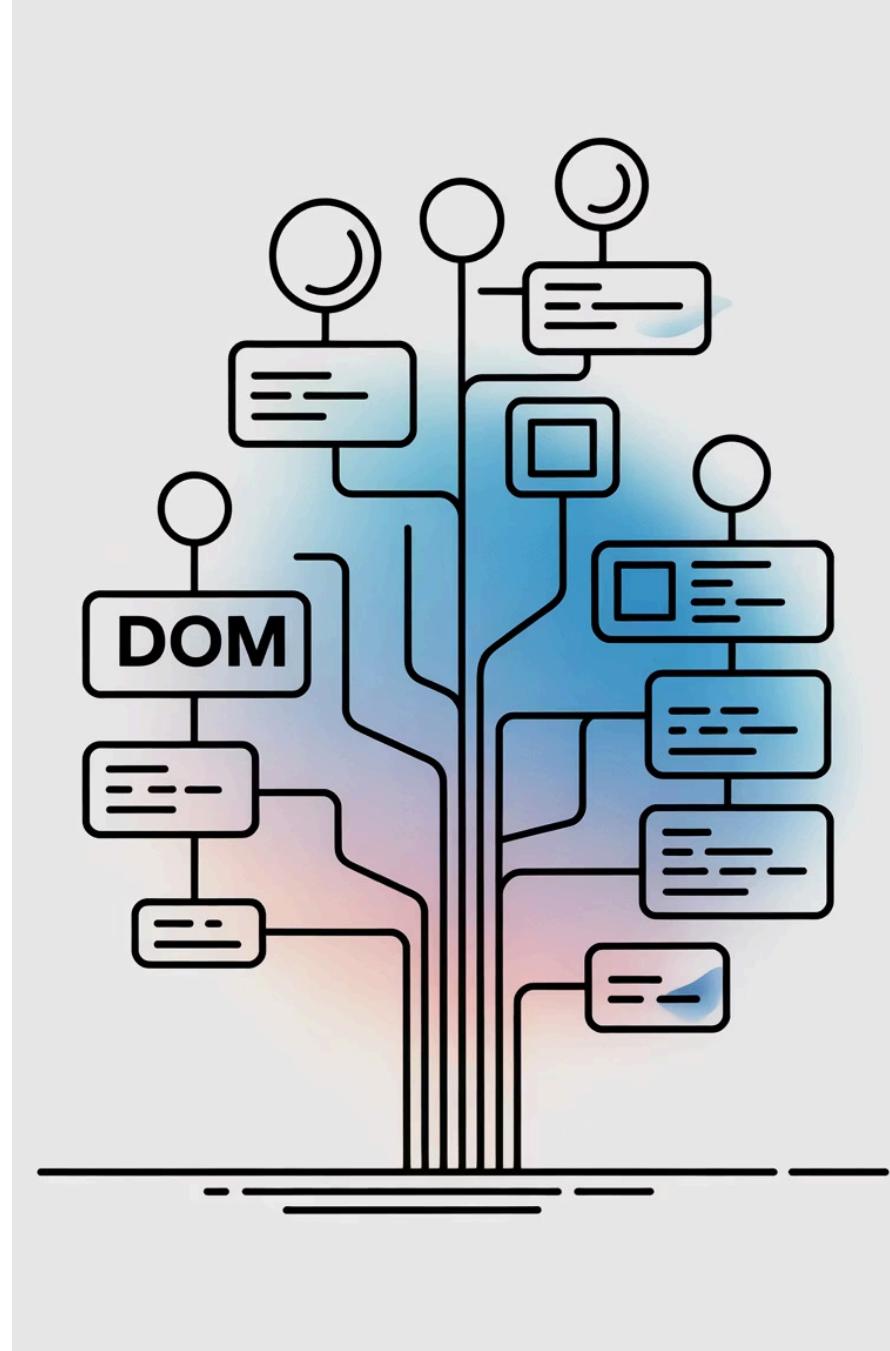
Navegación entre Nodos del DOM

Relaciones de parentesco

- `parentNode`: Nodo padre
- `childNodes`: Colección de nodos hijos
- `firstChild`: Primer nodo hijo
- `lastChild`: Último nodo hijo

Solo elementos HTML (sin texto ni comentarios)

- `parentElement`: Elemento padre
- `children`: Colección de elementos hijos
- `firstElementChild`: Primer elemento hijo
- `lastElementChild`: Último elemento hijo



Modificación del Contenido

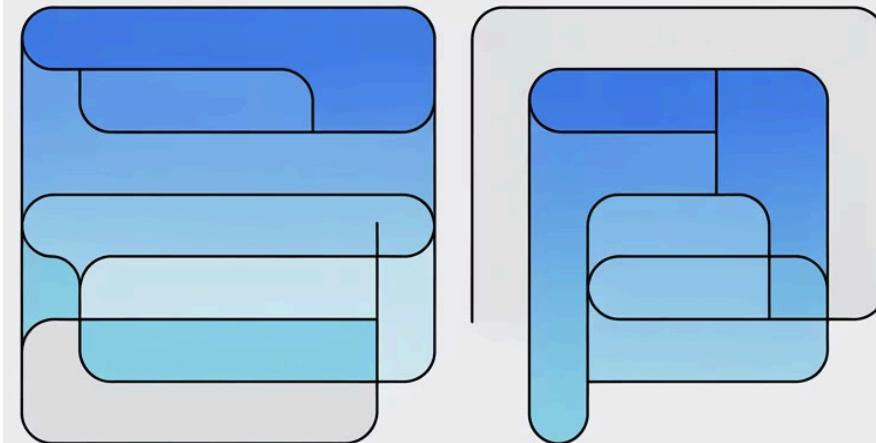
Propiedades para cambiar texto

```
// Cambiar solo el texto  
elemento.textContent = 'Nuevo  
texto';  
  
// Cambiar contenido HTML  
elemento.innerHTML = 'Texto  
con formato';  
  
// Obtener/cambiar valor de  
inputs  
inputElement.value = 'Nuevo  
valor';
```

Diferencias

- `textContent`: Solo texto plano, más seguro
- `innerHTML`: Permite HTML, pero puede ser vulnerable a ataques XSS si no se controla
- `innerText`: Similar a `textContent` pero respeta la visibilidad CSS

Dom Manipulation Explained



TextContent

InnerHTML

textContent vs innerHTML

1

textContent

- Trata el contenido como texto plano
- Más eficiente en rendimiento
- Seguro frente a inyección de código (XSS)
- Muestra exactamente lo que se ve en el editor

```
// HTML: <div id="ejemplo"><b>Hola</b></div>
```

```
// Obtener contenido  
console.log(div.textContent); // "Hola"
```

```
// Asignar contenido  
div.textContent = "Nuevo";  
// Resultado: el texto "Nuevo" aparecerá tal cual
```

2

innerHTML

- Interpreta el contenido como HTML
- Más lento (requiere parsear el HTML)
- Riesgo de seguridad si no se valida
- Útil para insertar contenido con formato

```
// HTML: <div id="ejemplo"><b>Hola</b></div>
```

```
// Obtener contenido  
console.log(div.innerHTML); // "Hola"
```

```
// Asignar contenido  
div.innerHTML = "Nuevo";  
// Resultado: la palabra "Nuevo" aparecerá en negrita
```

Modificación de Atributos

Métodos para atributos

```
// Obtener un atributo  
const src = img.getAttribute('src');  
  
// Establecer un atributo  
link.setAttribute('href', 'https://ejemplo.com');  
  
// Comprobar si existe  
if (boton.hasAttribute('disabled')) {  
    // Hacer algo  
}  
  
// Eliminar un atributo  
input.removeAttribute('readonly');
```

Acceso directo a atributos comunes

```
// Acceso directo (más cómodo)  
img.src = 'nueva-imagen.jpg';  
link.href = 'https://ejemplo.com';  
input.value = 'Nuevo texto';  
div.id = 'nuevold';  
elemento.className = 'clase1 clase2';
```

```
// Trabajando con clases (recomendado)  
elemento.classList.add('nueva');  
elemento.classList.remove('vieja');  
elemento.classList.toggle('activo');  
elemento.classList.contains('importante');
```

Modificación de Estilos CSS

Estilos en línea

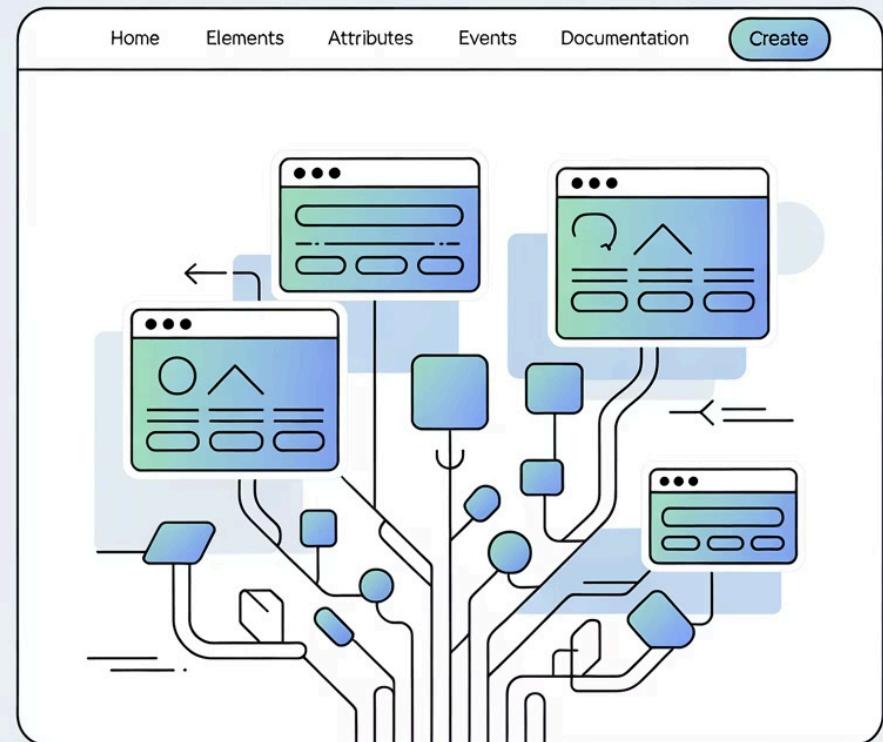
```
// Cambiar una propiedad  
elemento.style.color = 'red';  
elemento.style.fontSize = '16px';  
  
// Propiedades con guion se convierten a camelCase  
elemento.style.backgroundColor = '#f0f0f0';  
elemento.style.borderRadius = '5px';  
  
// Múltiples estilos a la vez  
elemento.style.cssText = 'color: red;  
font-size: 16px; background-color: #f0f0f0';
```

Clases CSS (mejor práctica)

```
// Añadir/quitar clases es más mantenible  
elemento.classList.add('destacado');  
elemento.classList.remove('oculto');  
elemento.classList.toggle('seleccionado');  
  
// Verificar si tiene una clase  
if (elemento.classList.contains('activo')) {  
    // Hacer algo  
}  
  
// Reemplazar todas las clases  
elemento.className = 'nueva-clase';
```

Usar clases CSS es más mantenible que cambiar estilos en línea directamente.

Creación y Eliminación de Elementos



```
// Crear un nuevo elemento
```

```
const nuevoParrafo = document.createElement('p');  
nuevoParrafo.textContent = 'Este es un nuevo párrafo';  
nuevoParrafo.className = 'destacado';
```

```
// Añadir el elemento al DOM
```

```
document.body.appendChild(nuevoParrafo);
```

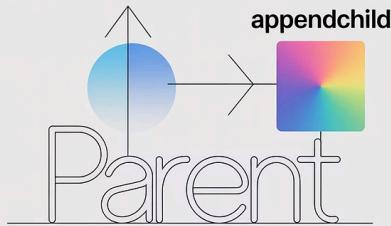
```
// Eliminar un elemento
```

```
const elementoAEliminar = document.getElementById('obsoleto');  
elementoAEliminar.remove();
```

```
// Alternativa para navegadores antiguos
```

```
elementoAEliminar.parentNode.removeChild(elementoAEliminar);
```

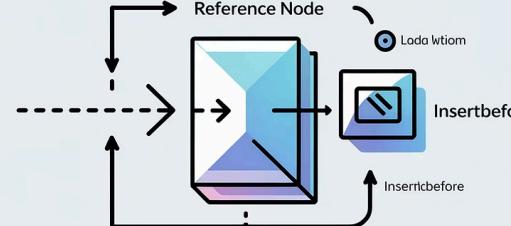
Métodos para Insertar Elementos



appendChild

Añade un elemento como último hijo del elemento padre.

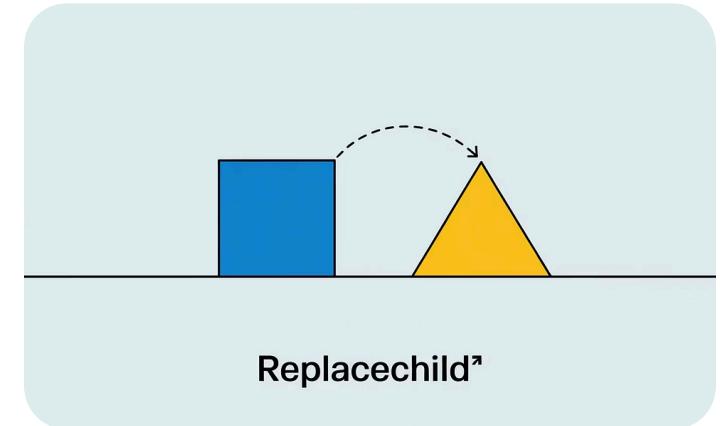
```
padre.appendChild(nuevoElemento);
```



insertBefore

Inserta un elemento antes de un nodo de referencia.

```
padre.insertBefore(nuevo, referencia);
```



replaceChild

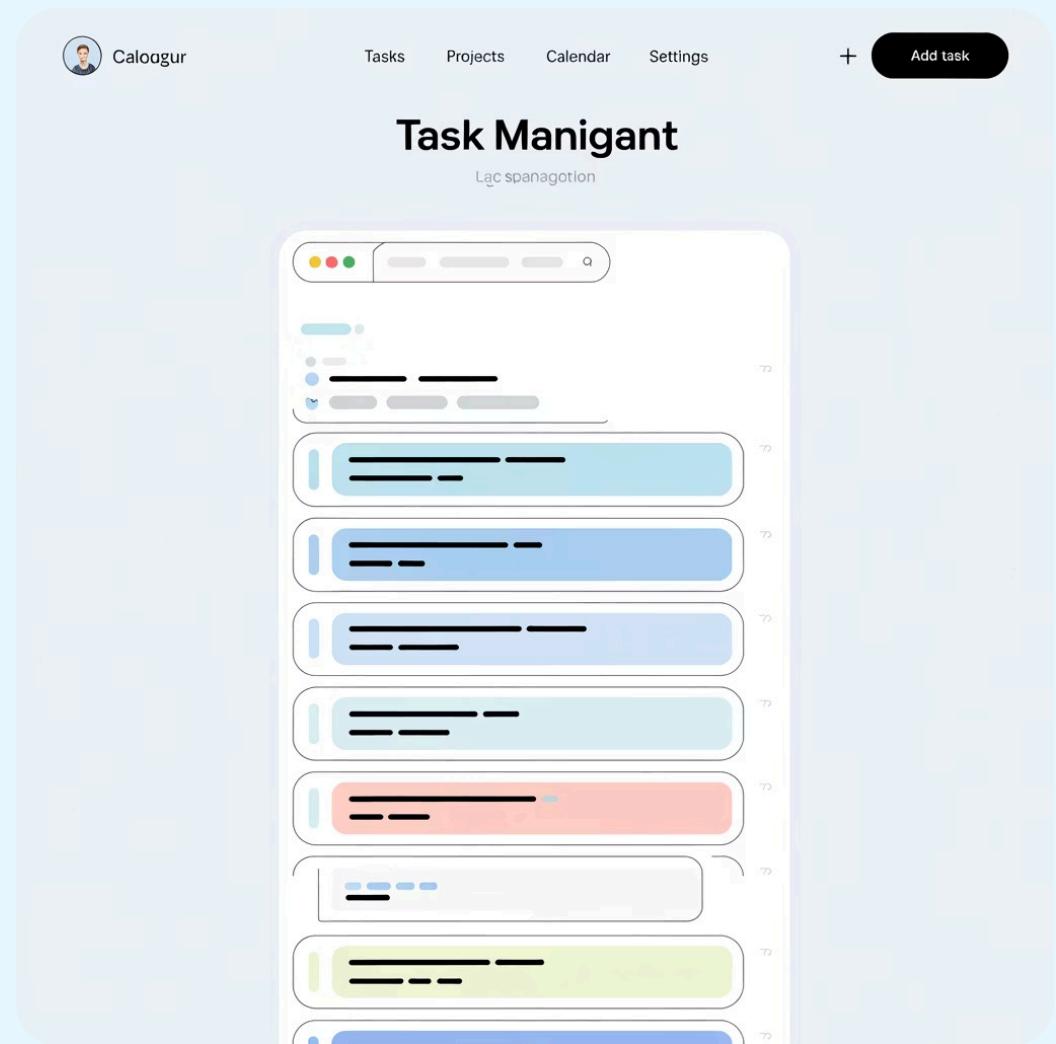
Reemplaza un hijo existente por otro elemento.

```
padre.replaceChild(nuevo, aReemplazar);
```

Existen métodos más modernos como before(), after(), prepend() y append() que ofrecen más flexibilidad.

Ejemplo: Lista Dinámica

```
// HTML:  
  
// <ul id="lista"></ul>  
// <input id="nuevoltem">  
// <button id="agregar">Añadir</button>  
  
const lista = document.getElementById('lista');  
const input = document.getElementById('nuevoltem');  
const boton = document.getElementById('agregar');  
  
boton.addEventListener('click', function() {  
    // Verificar que hay texto  
    if (input.value.trim() === "") return;  
  
    // Crear nuevo elemento de lista  
    const nuevoLi = document.createElement('li');  
    nuevoLi.textContent = input.value;  
  
    // Añadir a la lista  
    lista.appendChild(nuevoLi);  
  
    // Limpiar el input  
    input.value = "";  
});
```



Este ejemplo muestra cómo:

1. Seleccionar elementos existentes
2. Escuchar eventos (clic en botón)
3. Obtener valores de input
4. Crear nuevos elementos
5. Añadirlos al DOM

Delegación de Eventos

La [delegación de eventos](#) es una técnica para manejar eventos en elementos que aún no existen.

- Se coloca un listener en un contenedor padre
- Se aprovecha el **event bubbling** (propagación)
- Se comprueba el origen del evento (`event.target`)

Muy útil para listas dinámicas o elementos que se crean después de cargar la página.

```
// En vez de añadir un listener a cada item
// (que podrían no existir aún)
lista.addEventListener('click', function(event) {
  // Verificar si el clic fue en un LI
  if (event.target.tagName === 'LI') {
    // Marcar como completado
    event.target.classList.toggle('completado');
  }
});

// Ahora cualquier LI que se añada después
// también responderá al clic
```

Ejemplo Práctico: Calculadora

Vamos a construir una calculadora básica que combine todo lo aprendido:

- Funciones para operaciones matemáticas
- Eventos para capturar clics en botones
- Manipulación del DOM para mostrar resultados
- Modificación de estilos para feedback visual

Scientific

Graphing

Currency Converter

123 oowueabdi

Download App



HTML de la Calculadora

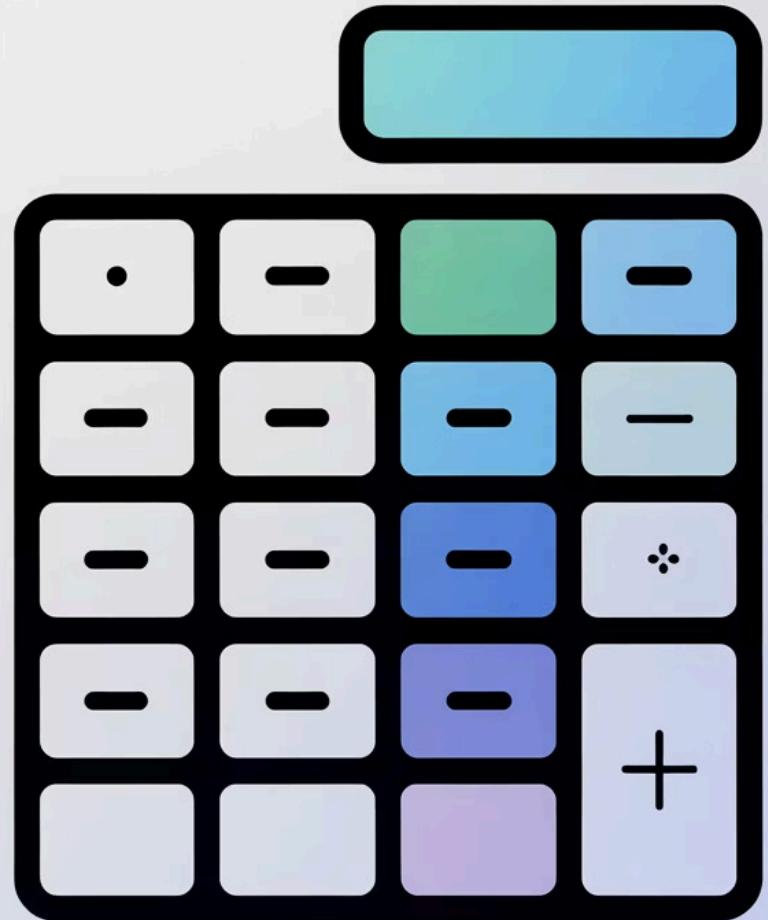
```
<div class="calculadora">  
  <div id="pantalla">0</div>  
  
  <div class="botones">  
    <button class="numero">7</button>  
    <button class="numero">8</button>  
    <button class="numero">9</button>  
    <button class="operador">+</button>  
  
    <button class="numero">4</button>  
    <button class="numero">5</button>  
    <button class="numero">6</button>  
    <button class="operador">-</button>  
  
    <button class="numero">1</button>  
    <button class="numero">2</button>  
    <button class="numero">3</button>  
    <button class="operador">*</button>  
  
    <button class="numero">0</button>  
    <button id="decimal">. </button>  
    <button id="igual">=</button>  
    <button class="operador">/</button>  
  
    <button id="limpiar">C</button>  
  </div>  
</div>
```

CSS de la Calculadora

```
.calculadora {  
    width: 300px;  
    border: 1px solid #ccc;  
    border-radius: 5px;  
    padding: 10px;  
    background-color: #f8f9fa;  
}  
  
#pantalla {  
    background-color: #fff;  
    border: 1px solid #ddd;  
    padding: 10px;  
    margin-bottom: 10px;  
    text-align: right;  
    font-size: 24px;  
    height: 40px;  
}
```

```
.botones {  
    display: grid;  
    grid-template-columns:  
    repeat(4, 1fr);  
    gap: 5px;  
}  
  
button {  
    padding: 15px;  
    font-size: 18px;  
    border: 1px solid #ccc;  
    border-radius: 5px;  
    background-color: #fff;  
    cursor: pointer;  
}  
  
button:hover {  
    background-color: #eaeaea;  
}
```

CSS Grid



JavaScript de la Calculadora

```
// Variables para almacenar los valores y la operación
let valorActual = '0';
let operadorPendiente = null;
let valorAnterior = null;

// Elementos del DOM
const pantalla = document.getElementById('pantalla');
const botones = document.querySelectorAll('button');

// Añadir event listener a todos los botones
botones.forEach(boton => {
  boton.addEventListener('click', function() {
    // Diferentes acciones según el tipo de botón
    if (boton.classList.contains('numero')) {
      ingresarNumero(boton.textContent);
    } else if (boton.classList.contains('operador')) {
      aplicarOperador(boton.textContent);
    } else if (boton.id === 'igual') {
      calcularResultado();
    } else if (boton.id === 'decimal') {
      ingresarDecimal();
    } else if (boton.id === 'limpiar') {
      limpiarCalculadora();
    }

    // Actualizar la pantalla
    actualizarPantalla();
  });
});

});
```

Funciones de la Calculadora

```
function ingresarNumero(numero) {  
    if (valorActual === '0') {  
        valorActual = numero;  
    } else {  
        valorActual += numero;  
    }  
}  
  
function ingresarDecimal() {  
    if (!valorActual.includes('.')) {  
        valorActual += ':';  
    }  
}  
  
function aplicarOperador(operador) {  
    if (operadorPendiente !== null) {  
        calcularResultado();  
    }  
    valorAnterior = valorActual;  
    valorActual = '0';  
    operadorPendiente = operador;  
}
```

```
function calcularResultado() {  
    if (operadorPendiente === null) return;  
  
    const num1 = parseFloat(valorAnterior);  
    const num2 = parseFloat(valorActual);  
  
    switch(operadorPendiente) {  
        case '+': valorActual = (num1 + num2).toString(); break;  
        case '-': valorActual = (num1 - num2).toString(); break;  
        case '*': valorActual = (num1 * num2).toString(); break;  
        case '/': valorActual = (num1 / num2).toString(); break;  
    }  
  
    operadorPendiente = null;  
    valorAnterior = null;  
}  
  
function limpiarCalculadora() {  
    valorActual = '0';  
    operadorPendiente = null;  
    valorAnterior = null;  
}  
  
function actualizarPantalla() {  
    pantalla.textContent = valorActual;  
}
```

#EDCOUNIANDES

<https://educacioncontinua.uniandes.edu.co/>

Contacto: educacion.continua@uniandes.edu.co

© - Derechos Reservados: La presente obra, y en general todos sus contenidos, se encuentran protegidos por las normas internacionales y nacionales vigentes sobre propiedad Intelectual, por lo tanto su utilización parcial o total, reproducción, comunicación pública, transformación, distribución, alquiler, préstamo público e importación, total o parcial, en todo o en parte, en formato impreso o digital y en cualquier formato conocido o por conocer, se encuentran prohibidos, y solo serán lícitos en la medida en que se cuente con la autorización previa y expresa por escrito de la Universidad de los Andes.