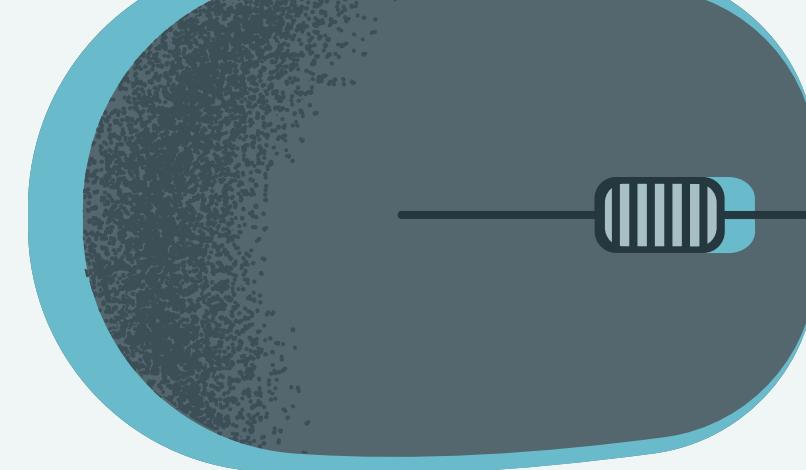
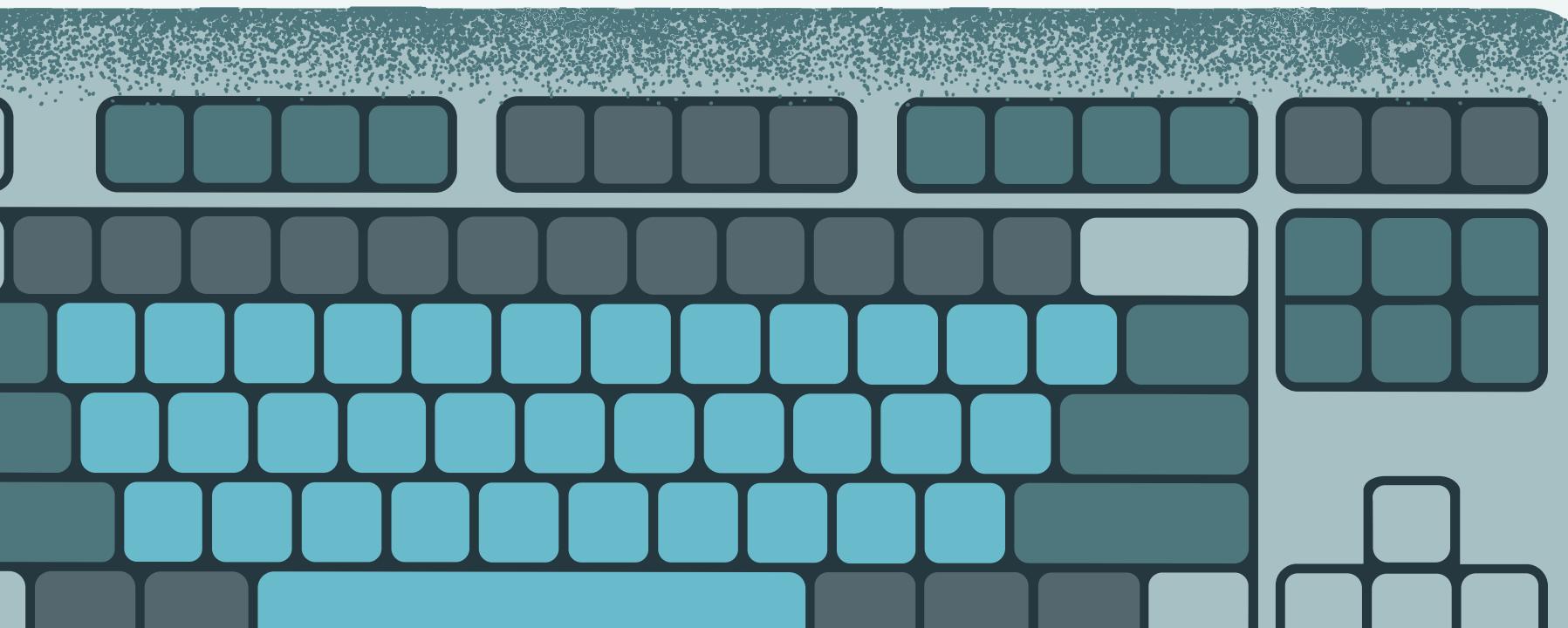
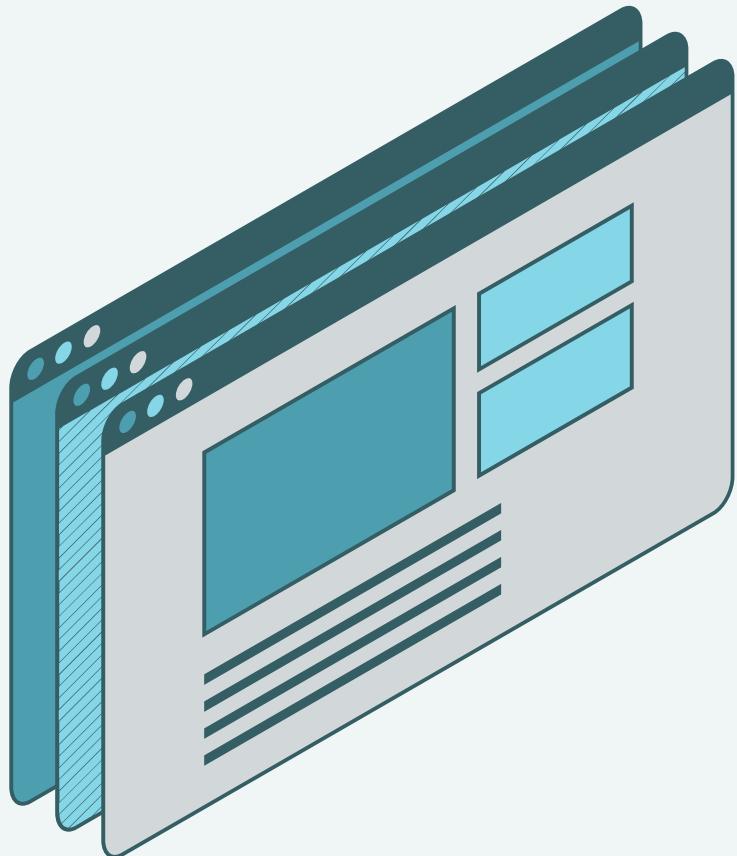


CURSO DÉ JAVASCRIPT

JS



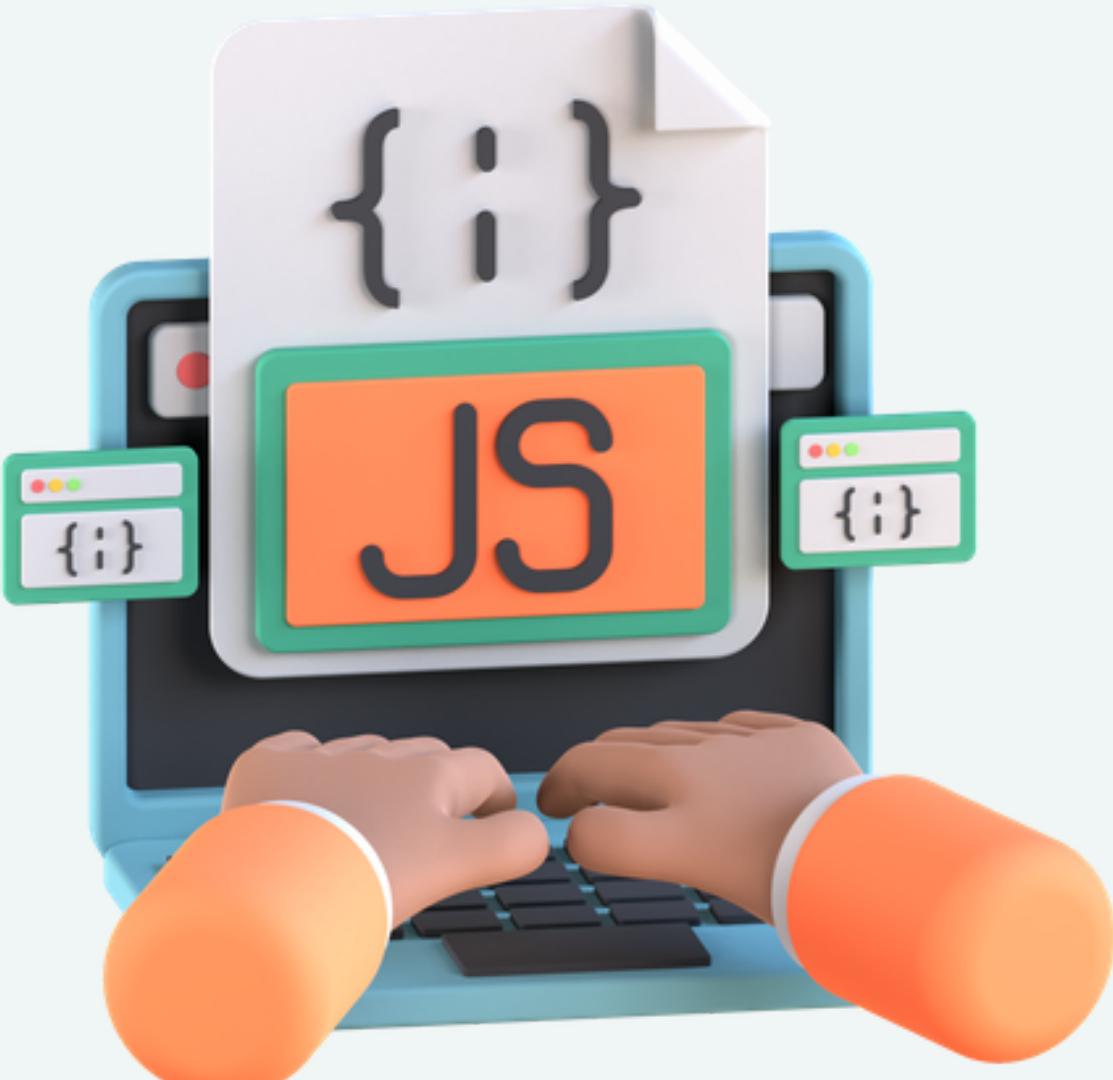
¿QUÉ ES LO QUE CONOCEREMOS EN ESTE CURSO?



En este curso, vamos a explorar los fundamentos de JavaScript, entender su sintaxis, sus estructuras y cómo podemos usarlo para manipular el contenido de una página web en tiempo real.

A lo largo de nuestras sesiones, aprenderán a construir la lógica que dará vida a sus aplicaciones web, y comenzarán a ver cómo JavaScript se convierte en una herramienta poderosa en el desarrollo de proyectos frontend y backend.

¿QUE APRENDEREMOS?



- Los principios básicos del lenguaje: tipos de datos, variables, operadores y control de flujo.
- Cómo definir y utilizar funciones para crear programas reutilizables y bien organizados.
- Entenderemos el modelo de objetos en JavaScript, aprendiendo a trabajar con objetos y arrays.
- Finalmente, daremos nuestros primeros pasos en la manipulación del DOM, donde aprenderemos cómo JavaScript interactúa con las páginas web para cambiar su estructura y contenido dinámicamente.

¿CÓMO SERÁN LAS SESIONES?



- Por cada tema que se trate en cada sesión les daré un contexto teórico del funcionamiento de alguna propiedad enfocadas a posibles casos donde se pueda aplicar.
- Les explicare ejemplos de código donde se muestren los temas de la clase.
- Les propondré algunos ejercicios para que puedan ser resueltos en equipo durante la clase donde en ese momento les podré ayudar con algunas dudas que tengan.

INTRODUCCIÓN A JAVASCRIPT



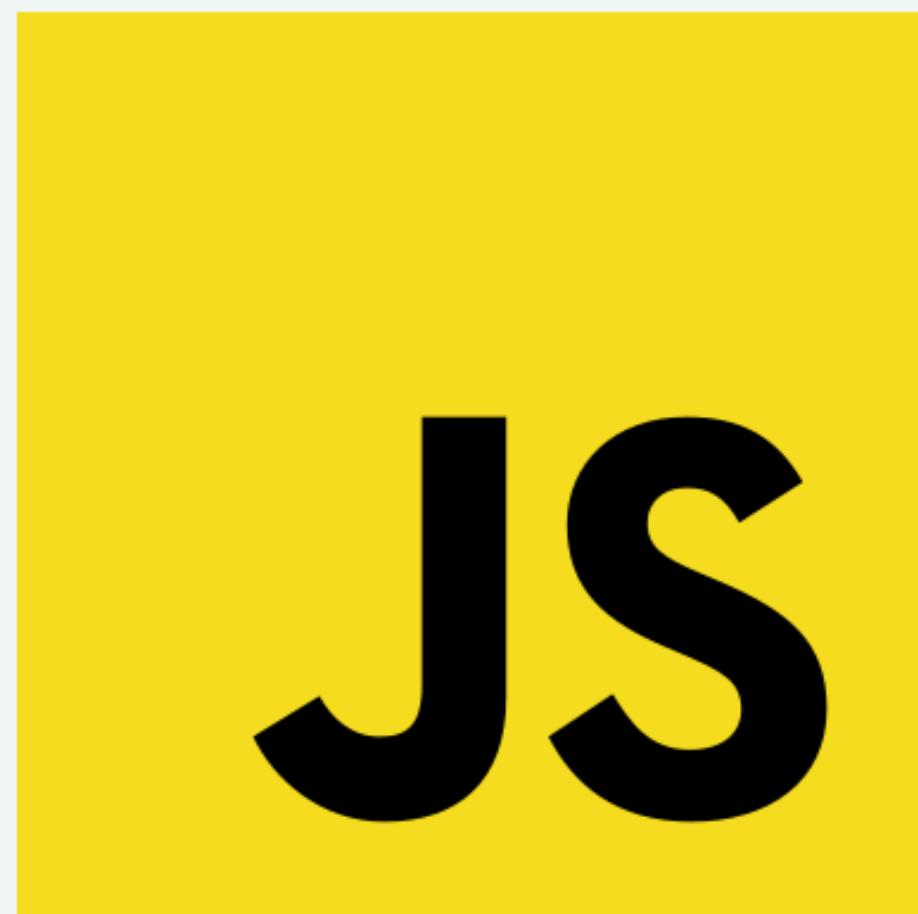
¿QUÉ ES JAVASCRIPT?

DEFINICIÓN

JavaScript es un lenguaje de programación interpretado, que se ejecuta del lado del cliente y del servidor. Principalmente, se usa para crear páginas web dinámicas e interactivas.

BREVE HISTORIA

Fue creado por Brendan Eich en 1995 y ha evolucionado a lo largo de los años, convirtiéndose en una de las tecnologías más importantes del desarrollo web junto con HTML y CSS.



SU IMPORTANCIA

- Es uno de los lenguajes más utilizados en el desarrollo web.
- Compatible con todos los navegadores modernos.
- Versátil: desde la manipulación de DOM hasta la creación de servidores (Node.js).

¿DONDE SE UTILIZA JAVASCRIPT?

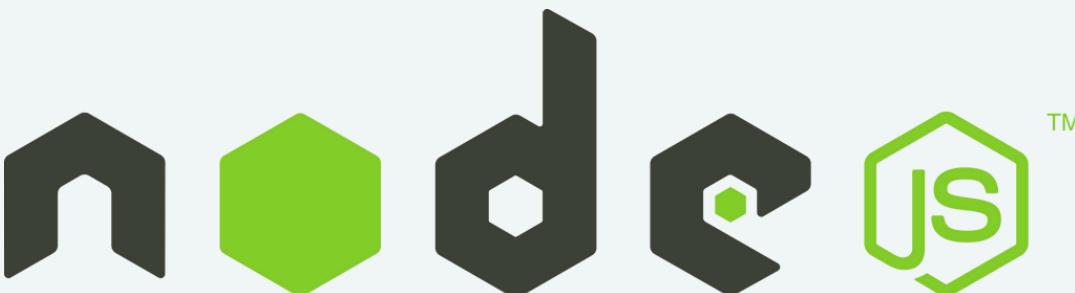
DESEARROLLO WEB (FRONENT)

JavaScript permite añadir interactividad a las páginas web, como validación de formularios, animaciones, interacciones con el usuario, etc

A screenshot of a web form titled "Datos del comprador". It contains fields for Name, Email, Phone, and Country, all with placeholder text. Below this, there's a section for "Datos de tarjeta de crédito o débito" with fields for Card Number and Name, also with placeholder text. At the bottom is a "Confirmar pedido" button.

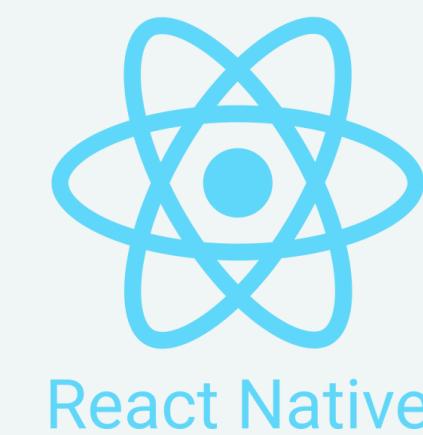
BACKEND (NODE.JS)

JavaScript también se utiliza en el lado del servidor a través de Node.js para construir aplicaciones escalables, APIs y servidores



APLICACIONES MÓVILES Y DE ESCRITORIO

Tecnologías como React Native y Electron, también se pueden desarrollar aplicaciones móviles y de escritorio



CARACTERÍSTICAS CLAVE DE JAVASCRIPT



LENGUAJE INTERPRETADO

Se ejecuta directamente en el navegador, sin necesidad de ser compilado.

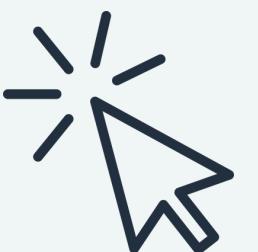
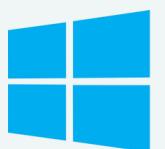
DINÁMICO Y DÉBILMENTE TIPADO

Las variables no necesitan declaración explícita de tipos.

```
let x = 5; // Número  
x = "Hola"; // Ahora es una cadena
```

MULTIPLATAFORMA

Funciona en cualquier dispositivo que soporte un navegador web



BASADO EN EVENTOS

JavaScript puede responder a eventos como clics, teclados, movimientos del ratón, etc

LENGUAJE ORIENTADO A OBJETOS BASADO EN PROTOTIPOS

Esto es debido a la forma en que gestiona la herencia y la creación de objetos

{OOP}

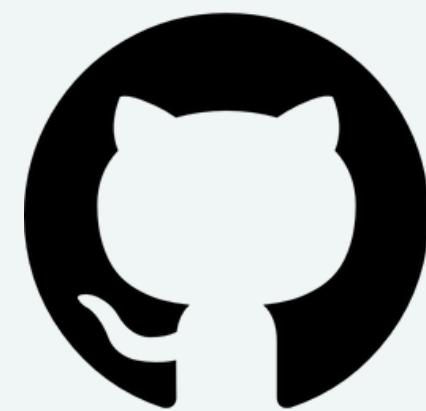
HERRAMIENTAS Y SOFTWARE PARA DESARROLLAR EN JAVASCRIPT



Visual Studio Code es el editor de código, desarrollado por Microsoft, es ligero y rápido y personalizable.

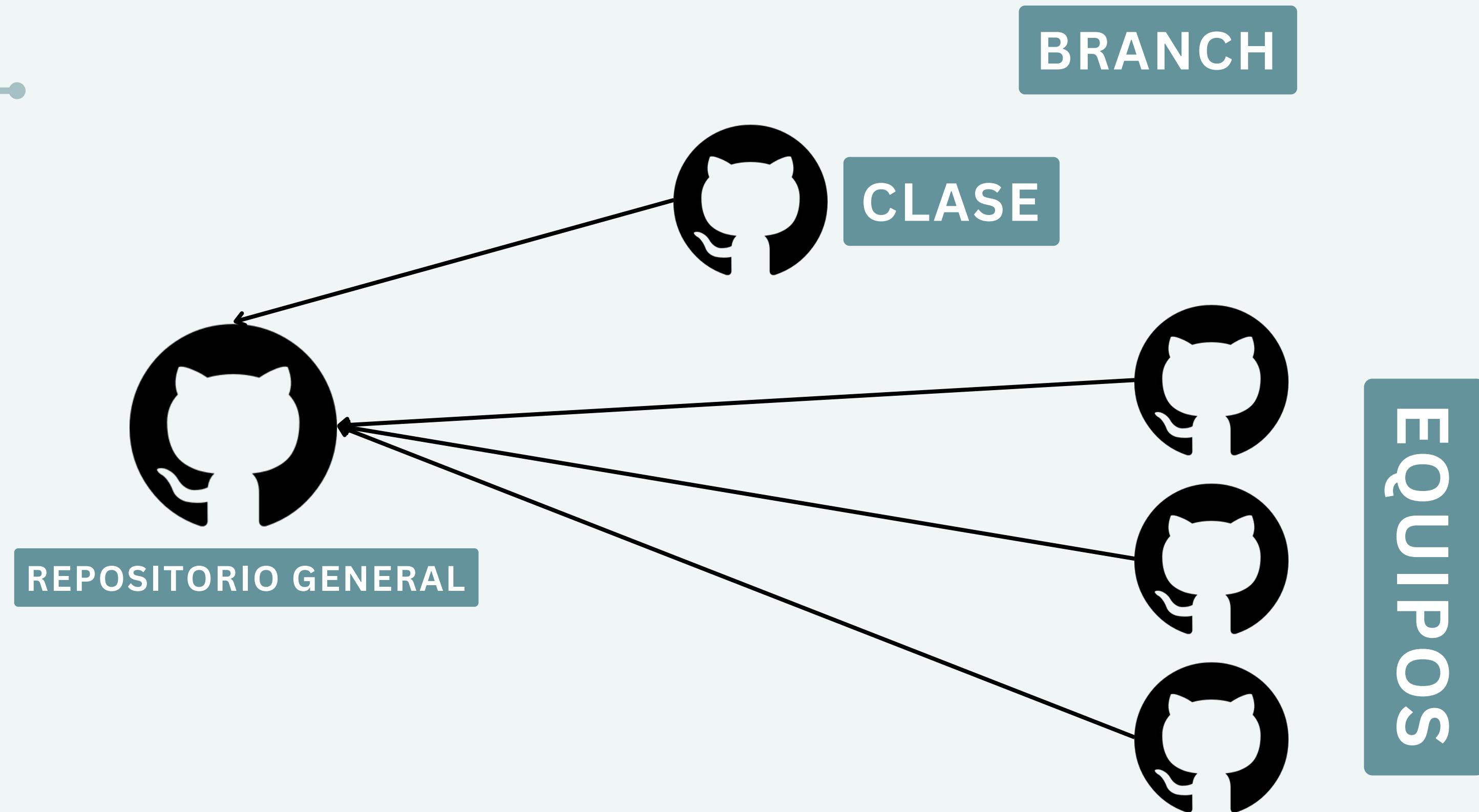


Chrome es el navegador web oficial de Google y está diseñado para ser veloz, seguro y personalizable.



La plataforma está creada para que los desarrolladores suban el código de sus aplicaciones y herramientas.

VERSIONAMIENTO DE PRACTICAS Y PROYECTOS POR EQUIPOS



ESTRUCTURAS CONTROL



SINTAXIS EN JAVASCRIPT

COMENTARIOS

```
3  
4 //Comentario en una sola linea  
5
```

```
3  
4 /*  
5  Este es un comentario  
6  de multiple lineas  
7 */
```

DECLARACIÓN DE VARIABLES

```
1  
2  
3 let nombre = "Juan";  
4 const edad = 20;  
5
```

SINTAXIS EN JAVASCRIPT

TIPOS DE DATOS

STRING: REPRESENTA TEXTO

```
2  
3  let cadena = "Hola";  
4  
5
```

OBJECT: COLECCIÓN DE PARES CLAVE-VALOR

```
3  
4  
5  let persona = { nombre: "Ana", edad: 25 };  
6
```

ARRAY: COLECCIÓN ORDENADA DE DATOS

```
4  
5  let numeros = [1, 2, 3, 4];  
6
```

NUMBER: REPRESENTA NÚMEROS

```
1.  
2  
3  let edad = 34;  
4
```

BOOLEAN: VERDADERO O FALSO

```
2  
3  let esVerdadero = true;  
4
```

SINTAXIS EN JAVASCRIPT

OPERADORES

ARITMÉTICOS

+ , - , * , / , %

```
4  
5  let suma = 5 + 3;  
6
```

LÓGICOS

&& (Y), || (O), ! (Negación)

```
let esMayor = 8 > 4;
```

DE COMPARACIÓN

==, ===, !=, !==, <, >, <=, =>

```
4  
5  let esIgual = 5 === 5;  
6
```

SINTAXIS EN JAVASCRIPT

CONDICIONALES

Los condicionales permiten ejecutar diferentes bloques de código dependiendo de una condición. Los más comunes son if, else if y else.

```
4  
5     let edad = 20;  
6  
7     if (edad >= 18) {  
8         console.log("Es mayor de edad");  
9     } else {  
10        console.log("Es menor de edad");  
11    }
```

SINTAXIS EN JAVASCRIPT

FUNCIONES

Las funciones son bloques de código reutilizables. Puedes definirlas usando la palabra clave `function` o con funciones flecha

```
//Declaración de función
function saludar(nombre) {
    return "Hola " + nombre;
}
console.log(saludar("Ana"));
```

```
//Declaración de Función flecha
const saludar = (nombre) => "Hola " + nombre;
console.log(saludar("Ana"));
```

SINTAXIS EN JAVASCRIPT

ALCANCE DE VARIABLES (SCOPE)

ALCANCE GLOBAL

Una variable declarada fuera de cualquier función o bloque de código es accesible desde cualquier parte del programa

ALCANCE DE FUNCIÓN

Las variables declaradas dentro de una función son accesibles solo dentro de esa función.

```
1  let global = "Estoy en el alcance global";
2
3  function miFuncion() {
4      let local = "Estoy en el alcance local";
5      console.log(global); // Accede a la variable global
6  }
7
8  miFuncion();
9  console.log(local); // Error: local no está definida fuera de la función
```

ALCANCE DE BLOQUE

Con `let` y `const`, las variables tienen un alcance dentro de bloques de código como bucles o condicionales.

SINTAXIS EN JAVASCRIPT

OBJETOS

Los objetos son estructuras de datos que almacenan pares clave-valor

```
let coche = {  
    marca: "Toyota",  
    modelo: "Corolla",  
    año: 2020,  
    conducir: function() {  
        console.log("Conduciendo el coche ");  
    }  
};  
  
console.log(coche.marca); // Toyota  
coche.conducir(); // Conduciendo el coche
```

SINTAXIS EN JAVASCRIPT

ARRAYS

Los arrays son listas ordenadas de valores. Se acceden mediante índices que empiezan en 0

```
1 let frutas = ["manzana", "banana", "naranja"];
2 console.log(frutas[0]); // manzana
3
4 frutas.push("pera");    // Agrega al final del array
5 console.log(frutas.length); // 4
```

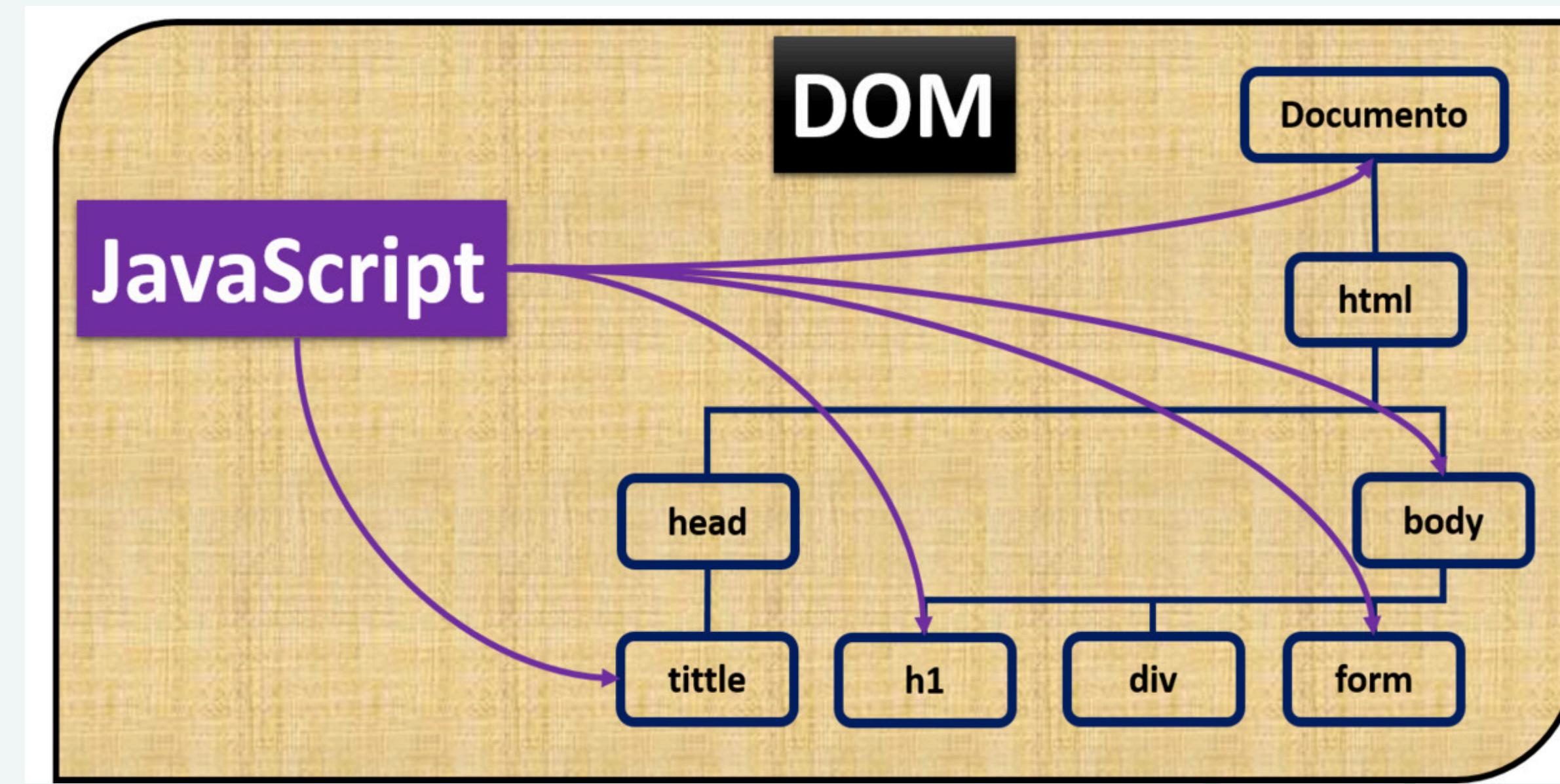
SINTAXIS EN JAVASCRIPT

BUCLES

Los bucles permiten ejecutar un bloque de código repetidamente. Los más comunes son `for`, `while` y `forEach` (en arrays).

```
1 ✓ for (let i = 0; i < 5; i++) {  
2     console.log(i); // Imprime 0, 1, 2, 3, 4  
3 }  
4  
5 let numeros = [1, 2, 3, 4];  
6 numeros.forEach(num => console.log(num)); // Recorre el array|
```

MANEJO DEL DOM



INTRODUCCIÓN AL DOM; ELEMENTOS DEL DOM

DEFINICIÓN

El DOM (Document Object Model) es una representación estructurada de un documento HTML. Es un árbol de nodos que permite a los lenguajes de programación acceder y manipular el contenido, estructura y estilo de un documento web.

INTERACTIVIDAD

JavaScript puede interactuar con el DOM para modificar elementos en tiempo real, lo que permite crear sitios web dinámicos

RELACIÓN CON HTML

Cada etiqueta HTML corresponde a un nodo en el árbol DOM

INTRODUCCIÓN AL DOM; ELEMENTOS DEL DOM

DEFINICIÓN

El DOM (Document Object Model) es una representación estructurada de un documento HTML. Es un árbol de nodos que permite a los lenguajes de programación acceder y manipular el contenido, estructura y estilo de un documento web.

INTERACTIVIDAD

JavaScript puede interactuar con el DOM para modificar elementos en tiempo real, lo que permite crear sitios web dinámicos

RELACIÓN CON HTML

Cada etiqueta HTML corresponde a un nodo en el árbol DOM

ESTRUCTURA DE NODOS EN HTML

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Estructura de árbol del DOM</title>
  </head>
  <body>
    <h1>Estructura de árbol del DOM</h1>
    <h2>Aprende sobre el DOM</h2>
  </body>
</html>
```

- Nuestro documento se llama **nodo raíz** y contiene un **nodo hijo** el cual es el elemento `<html>`.
- El elemento `<html>` contiene dos hijos los cuales son los elementos `<head>` y `<body>`.
- Ambos elementos `<head>` y `<body>` tienen hijos propios.

COMO SELECCIONAR ELEMENTOS EN EL DOCUMENTO

METODOS PARA SELECCIONAR ELEMENTOS

- **getElementById()**

En HTML, los ids se utilizan como identificadores únicos para los elementos HTML. Esto significa que no podemos tener el mismo nombre de id para dos elementos diferentes.

ESTO ES INCORRECTO

```
<p id="para">Este es mi primer parágrafo.</p>
<p id="para">Este es mi segundo parágrafo.</p>
```

TOMANDO EL ELEMENTO

```
document.getElementById("nombre de id va aquí")
```

ESTO ES CORRECTO

```
<p id="para1">Este es mi primer parágrafo.</p>
<p id="para2">Este es mi segundo parágrafo.</p>
```

TOMANDO EL ELEMENTO

```
const paragraph1 = document.getElementById("para1");
console.log(paragraph1);
```

COMO SELECCIONAR ELEMENTOS EN EL DOCUMENTO

METODOS PARA SELECCIONAR ELEMENTOS

- `querySelector()`

Puedes usar este método para encontrar elementos con uno o más selectores de CSS.

```
<h1>Opciones ejemplo</h1>
  <ul class="list">
    <li>Opcion 1</li>
    <li>Opcion 2</li>
    <li>Opcion 3</li>
    <li>Opcion 4</li>
  </ul>
```

```
function getValuebyClass(){
  const h1Element = document.querySelector(".componente");
  console.log(h1Element.textContent);
}

function getListadoOpcionesbyClass(){
  const listElement = document.querySelector(".list");
  console.log(listElement.textContent);
}
```

AGREGANDO ELEMENTOS

METODOS PARA CREAR ELEMENTOS

- `createElement()`

Nos ayuda a crear una nueva etiqueta HTML .

```
<div>
  <ul id="lista">
    <li>Elemento 1</li>
    <li>Elemento 2</li>
  </ul>

  <button onclick="agregarElemento()">Agregar Elemento</button>
</div>
```

```
function agregarElemento(){
  const lista = document.getElementById('lista');
  const nuevoElemento = document.createElement('li');
  nuevoElemento.textContent = 'Nuevo Elemento';
  lista.appendChild(nuevoElemento); // Agrega el nuevo elemento a la lista
}
```

- `createElement('tag')`: Crea un nuevo nodo de tipo elemento (como un ``).
- `appendChild(element)`: Añade el nuevo elemento como hijo del elemento padre.
- `remove()`: Elimina un elemento del DOM.

GITHUB



[HTTPS://GITHUB.COM/SERGIOFINIX/CURSOJAVASCRIPT.GIT](https://github.com/SergioFinix/CursoJavaScript.git)

```
Quarksoft@DESKTOP-S9679MO MINGW64 ~/Documents/CursoJavaScript (main)
$ git clone https://github.com/SergioFinix/CursoJavaScript.git|
```

ACCESO A ELEMENTOS PADRE/HIJO DE UN ELEMENTO DE DOM

ACCESO A ELEMENTOS PADRE

PARENTNODE

El método `parentNode` devuelve el nodo padre inmediato de un elemento

```
<div id="contenedor">
  <p id="parrafo">Este es un párrafo.</p>
</div>

<script>
  const parrafo = document.getElementById("parrafo");
  const padre = parrafo.parentNode;
  console.log(padre); // Muestra el <div id="contenedor">
</script>
```

ACCESO A ELEMENTOS PADRE/HIJO DE UN ELEMENTO DE DOM

ACCESO A ELEMENTOS HIJO

CHILDNODES

Devuelve una lista de todos los nodos hijos, incluyendo texto y comentarios

CHILDREN

Devuelve solo los elementos hijos (excluye texto y comentarios)

FIRSTELEMENTCHILD Y LASTELEMENTCHILD

Accede al primer y último elemento hijo

NEXTELEMENTSIBLING Y PREVIOUSELEMENTSIBLING:

Accede al siguiente o anterior hermano de un elemento.

CREACIÓN DE ELEMENTOS DE DOM, INSERCIÓN DE ELEMENTOS DE DOM.

MÉTODOS COMUNES DE INSERCIÓN

- **appendChild():** Inserta un nuevo nodo al final de otro nodo.
- **prepend():** Inserta un nuevo nodo al principio de otro nodo.
- **insertBefore():** Inserta un nuevo nodo antes de un nodo de referencia.

AGREGAR, EMILIMINAR, MODIFICAR CLASES DE ESTILO DESDE EL DOM

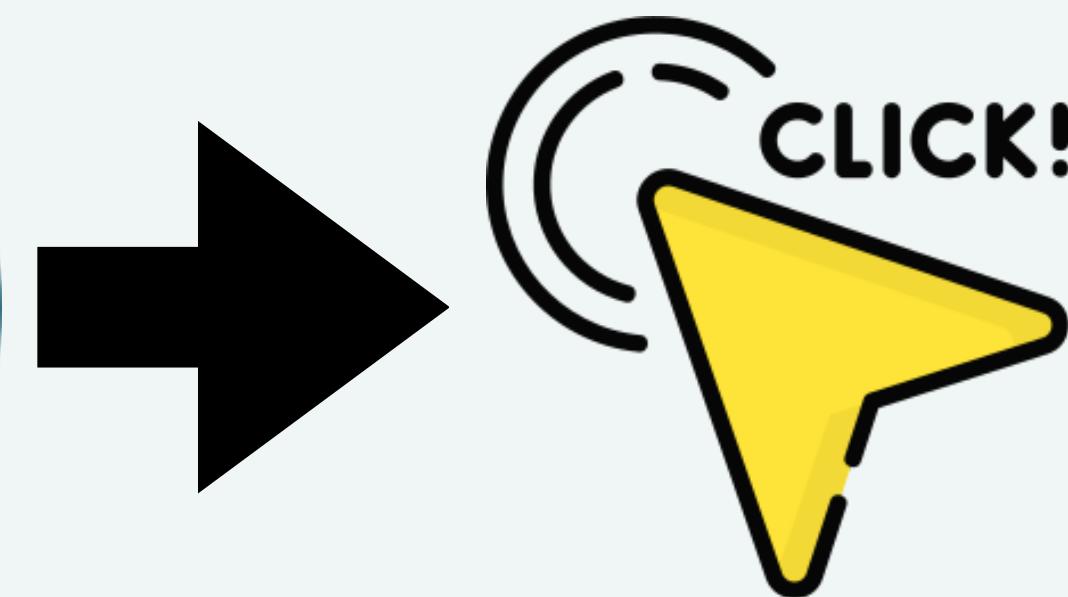
MÉTODOS COMUNES

- **classList.add():** Añade una o varias clases.
- **classList.remove():** Elimina clases.
- **classList.toggle():** Alterna entre añadir y eliminar una clase, útil para elementos interactivos.
- **classList.contains():** Verifica si un elemento tiene una clase

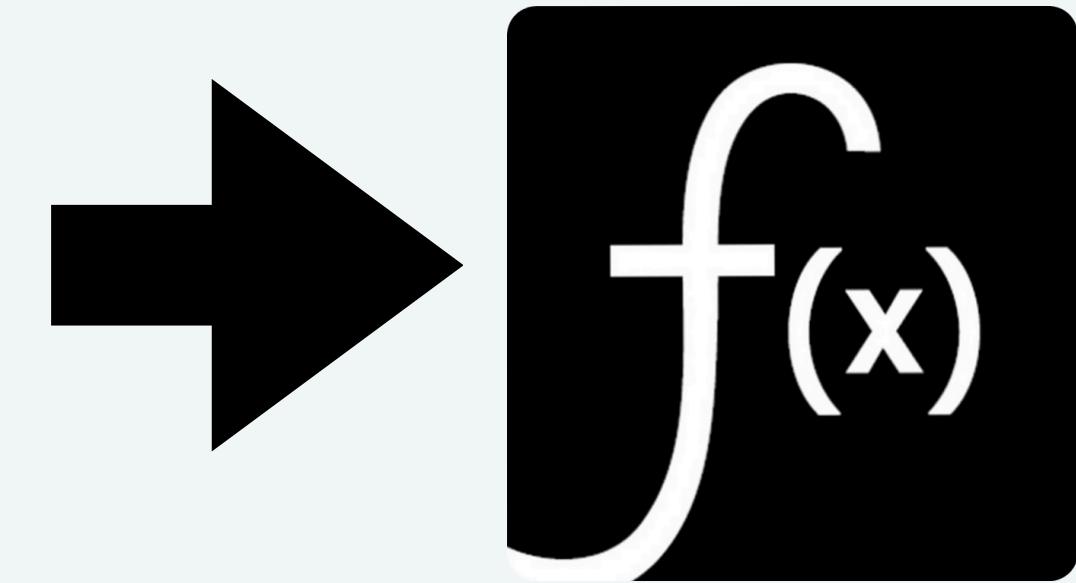
EVENTOS EN JAVASCRIPT



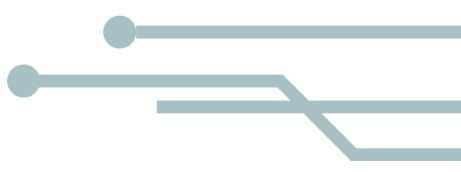
Usuario



Evento JavaScript



Función asociada



TIPOS DE EVENTOS

¿QUÉ ES UN EVENTO?

Un evento Javascript es una característica especial que ha sucedido en nuestra página y a la cual le asociamos una funcionalidad, de modo que se ejecute cada vez que suceda dicho evento.

Por ejemplo, el evento **click** se dispara cuando el usuario hace click en un elemento de nuestra página.

¿QUE ES LO QUE PUEDE DESATAR UN EVENTO?

- **Eventos de ratón:** como **click**, **dblclick**, **mousemove**, **mouseover**, **mouseout**.
- **Eventos de teclado:** como **keydown**, **keyup**, **keypress**.
- **Eventos de formulario:** como **submit**, **change**, **focus**, **blur**.

COMO PODEMOS MANEJAR UN EVENTO

Forma	Ejemplo
Mediante atributos HTML	<button onClick="alert('Hello!')>Saludar</button>
Mediante propiedades Javascript	<pre><button>Saludar</button> <script> const button = document.querySelector("button"); button.onclick = function() {alert("Hello!");} </script></pre>
Mediante addEventListener()	<pre>const button = document.querySelector("button"); function action() { alert("Hello!"); } button.addEventListener("click", action);</pre>

FASES DE UN EVENTO, EVENTOS DE DOM

Los eventos en el DOM siguen un proceso conocido como flujo de eventos, que tiene tres fases:

FASE DE CAPTURA (CAPTURING PHASE):

El evento comienza desde el nodo raíz (generalmente window o document) y viaja hacia abajo a través de los elementos padres hasta llegar al objetivo específico

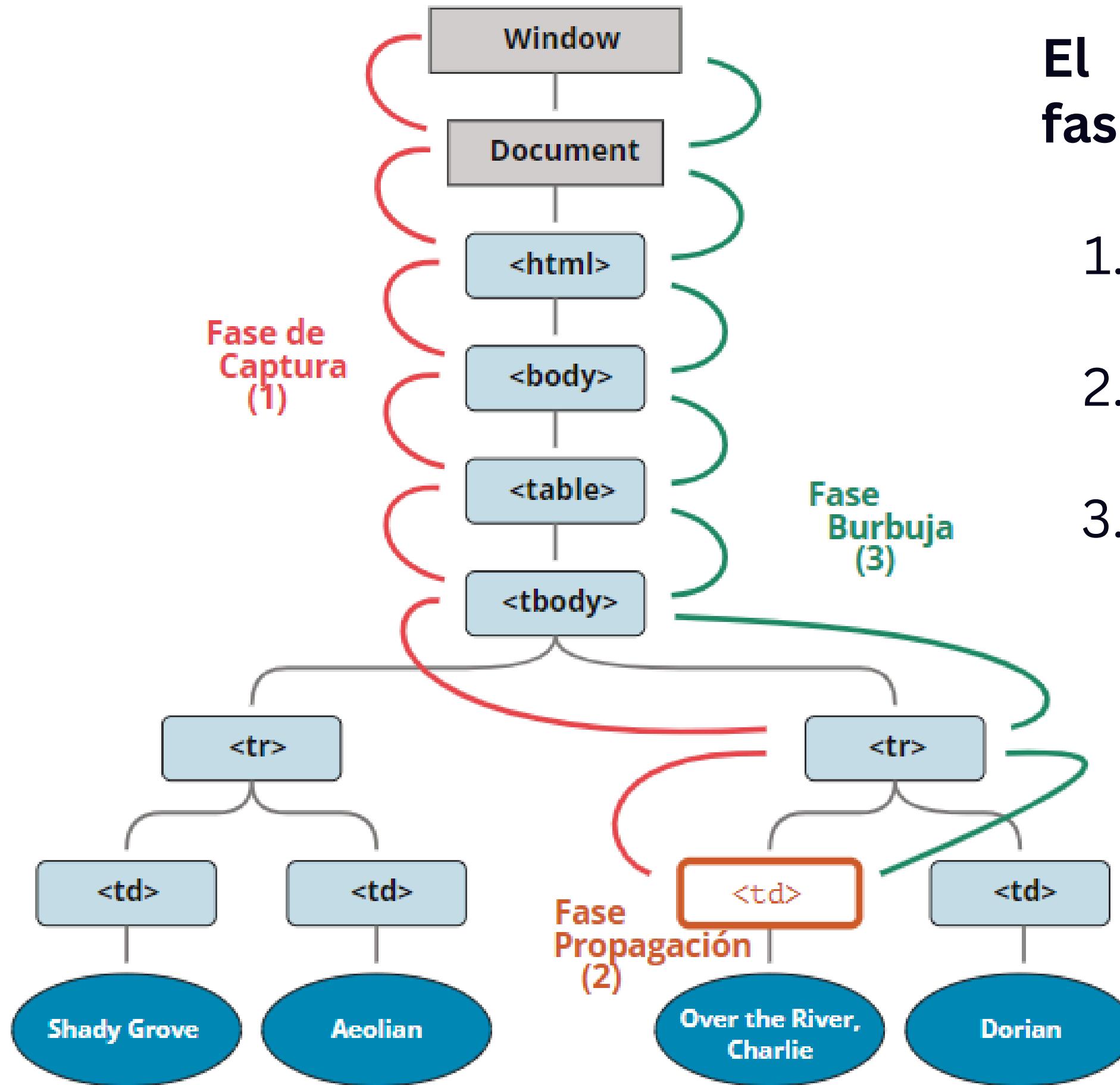
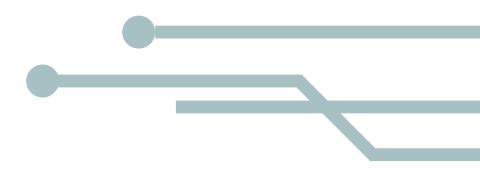
FASE DE OBJETIVO (TARGET PHASE):

El evento realmente se dispara en el elemento objetivo, es decir, el que recibe la interacción directa del usuario

FASE DE BURBUJA (BUBBLING PHASE)

Después de dispararse el evento, el flujo "burbujea" hacia arriba, desde el elemento objetivo hasta los elementos padres nuevamente

FASES DE UN EVENTO, EVENTOS DE DOM



El estándar de eventos del DOM describe 3 fases de la propagación de eventos:

1. Fase de captura – el evento desciende al elemento.
2. Fase de objetivo – el evento alcanza al elemento.
3. Fase de propagación – el evento se propaga hacia arriba del elemento.

OBJETO EVENTO, COLA EVENTO

OBJETO EVENT

El objeto Event es un objeto que se genera automáticamente cuando ocurre un evento en el navegador (como un clic, la carga de una página, el envío de un formulario, etc.). Este objeto contiene toda la información relevante sobre el evento que ha sucedido y nos permite interactuar con él.

EJEMPLOS DE DATOS QUE PODEMOS OBTENER DE UN EVENTO

- Qué elemento disparó el evento.
- La posición del ratón al hacer clic.
- Si se presionó alguna tecla especial.

PROPIEDADES COMUNES DEL OBJETO EVENT

1. `target`: Indica el elemento en el cual ocurrió el evento.
2. `type`: El tipo de evento que ocurrió (por ejemplo, "click", "keydown").
3. `preventDefault()`: Método que evita el comportamiento predeterminado del evento (por ejemplo, evitar que un formulario se envíe).
4. `stopPropagation()`: Método que detiene la propagación del evento hacia los elementos padres.

PROGRAMACIÓN ASINCRÓNICA (CALLBACKS, PROMISES, ASYNC/AWAIT)

INTRODUCCIÓN A JAVASCRIPT ASÍNCRONO

{.js}

ASYNC/AWAIT

¿QUÉ ES JAVASCRIPT ASÍNCRONO?

JavaScript asíncrono permite realizar múltiples tareas simultáneamente sin bloquear la ejecución del código. Mientras se espera una tarea (como la carga de datos desde un servidor), el programa puede continuar ejecutando otras tareas.

SÍNCRONO

Las tareas se ejecutan en orden, una después de la otra



ASÍNCRONO

Permite que una tarea se inicie, pero no detiene la ejecución de otras, lo que mejora la eficiencia



¿POR QUÉ ES IMPORTANTE?

JavaScript asíncrono permite realizar múltiples tareas simultáneamente sin bloquear la ejecución del código. Mientras se espera una tarea (como la carga de datos desde un servidor), el programa puede continuar ejecutando otras tareas.

MEJORA EL RENDIMIENTO

En aplicaciones web, muchas tareas como solicitudes HTTP, acceso a bases de datos o temporizadores pueden tardar en completarse. JavaScript asíncrono ayuda a que la interfaz siga siendo receptiva mientras esas operaciones suceden en segundo plano

EVITA EL "BLOQUEO"

Sin asíncrono, la aplicación puede quedar "congelada" si una tarea tarda mucho en completarse

FORMAS DE MANEJAR LA ASÍNCRONÍA EN JAVASCRIPT

CALLBACKS

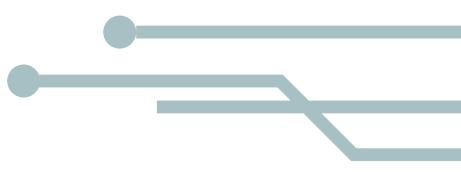
Un callback es una función que se pasa como argumento a otra función, y se ejecuta después de que la tarea asíncrona haya terminado

PROMISES

Una Promise es un objeto que representa la eventual finalización (o fracaso) de una operación asíncrona.

ASYNC/AWAIT

async y await son una sintaxis más moderna y clara para trabajar con código asíncrono. async convierte una función en asíncrona, permitiendo que se use await dentro de ella para esperar que una Promise se resuelva



CASOS COMUNES DE USO

SOLICITUDES HTTP (AJAX)

- Uso de `fetch()` o bibliotecas como `Axios` para obtener datos de servidores.

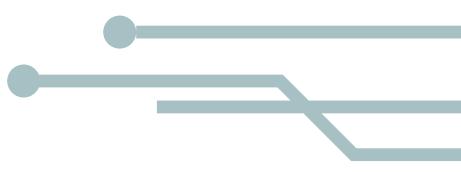
TEMPORIZADORES

Uso de `setTimeout()` o `setInterval()` para ejecutar funciones de manera asíncrona.

LECTURA DE ARCHIVOS O ACCESO A BASES DE DATOS

Estas tareas pueden tardar, por lo que se realizan de manera asíncrona.

PROMISES



¿QUÉ ES UNA PROMISE?

Una promise (promesa) en JavaScript es un objeto que representa la eventual finalización (o fallo) de una operación asíncrona y su resultado

UNA PROMESA TIENE TRES ESTADOS:

- PENDIENTE (PENDING)

El estado inicial ni terminada ni iniciada

- RESUELTA (FULFILLED)

La operación fue completada con éxito

- RECHAZADA (REJECTED)

La operación falló

¿QUÉ ES UNA PROMISE?

SINTAXIS BÁSICA DE UNA PROMISE

Para crear una promesa, deberás crear una nueva instancia del objeto **Promise** llamando al constructor **Promise**

```
// Simula la obtención de datos de usuarios con una Promise
function fetchUsers() {
  return new Promise((resolve, reject) => {
    // Simulamos una operación asíncrona con setTimeout
    setTimeout(() => {
      const success = true; // Cambia a false para simular un error
      if (success) {
        resolve([
          { id: 1, name: 'Juan Pérez' },
          { id: 2, name: 'María López' },
          { id: 3, name: 'Carlos García' }
        ]);
      } else {
        reject('Error al cargar los usuarios');
      }
    }, 2000); // Simulamos un retraso de 2 segundos
  });
}
```

¿QUÉ ES UNA PROMISE?

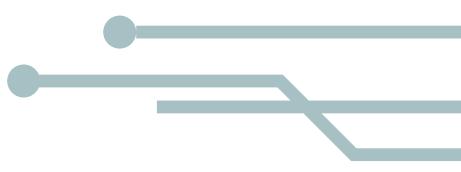
MANEJO DE UNA PROMISE

Para manejar el resultado de los métodos se maneja por medio de dos métodos `.then()` y `.catch()`

```
// Evento del botón para cargar usuarios
document.getElementById('loadButton').addEventListener('click', () => {
  // Limpia los mensajes anteriores
  document.getElementById('userList').innerHTML = 'Cargando usuarios...';
  document.getElementById('errorMessage').textContent = '';

  // Llama a fetchUsers, que devuelve una Promise
  fetchUsers()
    .then(users => {
      displayUsers(users); // Si la Promise se resuelve, muestra los usuarios
    })
    .catch(error => {
      showError(error); // Si la Promise es rechazada, muestra el error
    });
});
```

ASYNC/AWAIT



ASYNC/AWAIT

async/await es una forma más moderna y legible de manejar operaciones asíncronas en JavaScript

ASYNC

Una función marcada como `async` es una función que automáticamente devuelve una promesa. Esto significa que puedes usarla como una promesa y encadenarla con `.then()` o `await` para esperar su resultado.

AWAIT

Se utiliza dentro de funciones `async` y te permite "esperar" a que una promesa se resuelva. Detiene la ejecución de la función hasta que la promesa se resuelve o se rechaza, sin bloquear el hilo principal.

ASYNC/AWAIT

PROMESAS VS ASYNC/AWAIT

```
1 function obtenerDatos() {  
2     return new Promise((resolve) => {  
3         setTimeout(() => {  
4             resolve("Datos obtenidos");  
5         }, 2000); // Simula una operación asíncrona  
6     });  
7 }  
8  
9 obtenerDatos()  
10 .then((resultado) => {  
11     console.log(resultado); // 'Datos obtenidos'  
12 })  
13 .catch((error) => {  
14     console.error("Error:", error);  
15 });
```

```
1 async function obtenerDatos() {  
2     const resultado = await new Promise((resolve) => {  
3         setTimeout(() => {  
4             resolve("Datos obtenidos");  
5         }, 2000); // Simula una operación asíncrona  
6     });  
7     console.log(resultado); // 'Datos obtenidos' después de 2 segundos  
8 }  
9  
10 obtenerDatos();  
11 |
```

ASYNC/AWAIT

async/await es una forma más moderna y legible de manejar operaciones asíncronas en JavaScript

ASYNC TRANSFORMA FUNCIONES EN PROMESAS

Una función declarada como `async` devuelve una promesa automáticamente. Esto significa que puedes usarla con `then()` o esperar su resultado usando `await`

```
1 ↵ async function ejemploAsync() {  
2   |   return "Hola, mundo";  
3   }  
4  
5 ↵ ejemploAsync().then((resultado) => {  
6   |   console.log(resultado); // 'Hola, mundo'  
7   });
```

ASYNC/AWAIT

AWAIT "PAUSA" LA EJECUCIÓN DE LA FUNCIÓN HASTA QUE SE RESUELVE LA PROMESA

- El uso de await dentro de una función async hace que JavaScript espere la resolución de una promesa antes de continuar con la ejecución de la función
- No bloquea el resto de las operaciones en el programa (JavaScript sigue siendo asíncrono), pero "pausa" la función async hasta que la promesa se resuelve

```
async function ejemploAwait() {  
  const resultado = await Promise.resolve("Esperando promesa");  
  console.log(resultado); // 'Esperando promesa'  
}  
  
ejemploAwait();
```

MANEJO DE ERRORES EN ASYNC/AWAIT

AWAIT "PAUSA" LA EJECUCIÓN DE LA FUNCIÓN HASTA QUE SE RESUELVE LA PROMESA

Igual que con las promesas y .catch(), podemos manejar errores en funciones async usando bloques try y catch

```
1  ↴ async function obtenerDatos() {  
2    ↴   try {  
3      const resultado = await Promise.reject("Error en la promesa");  
4      console.log(resultado);  
5    ↴   } catch (error) {  
6      console.error("Error capturado:", error); // 'Error capturado: Error en la promesa'  
7    }  
8  }  
9  
10 obtenerDatos();
```

VENTAJAS DE USAR ASYNC/AWAIT

- Código más legible y estructurado: Comparado con las promesas anidadas o el encadenamiento .then(), async/await hace que el código sea más fácil de entender, sin tantas callback anidadas.
- Simulación de código síncrono: Aunque JavaScript sigue siendo asíncrono, await permite que el flujo de las funciones parezca más secuencial y menos fragmentado.
- Manejo de errores más limpio: En lugar de manejar errores con .catch(), puedes usar bloques try/catch, lo que es más familiar para los desarrolladores.

EN CONCLUSIÓN ASYNC/AWAIT

- **async convierte cualquier función en una promesa. Debes usarla para poder usar await.**
- **await detiene la ejecución de la función hasta que una promesa se resuelva o se rechace. Esto hace que tu código sea más legible y fácil de escribir.**
- **Manejo de errores: En vez de usar .catch(), puedes utilizar try/catch para manejar errores de manera clara y sencilla**

PLANTILLAS LITERALES

INTRODUCCIÓN A PLANTILLAS EN JAVASCRIPT (TEMPLATE LITERALS)

- Las plantillas literales son cadenas literales que habilitan el uso de expresiones incrustadas. Con ellas, es posible utilizar cadenas de caracteres de más de una línea, y funcionalidades de interpolación de cadenas de caracteres.

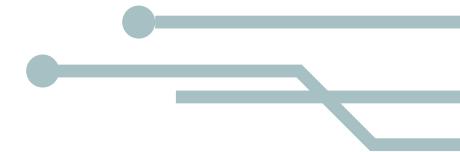
CARACTERÍSTICAS DE LAS PLANTILLAS LITERALES

INTERPOLACIÓN DE VARIABLES

Permite incluir variables y expresiones dentro de una cadena de texto.

INCRUSTACIÓN DE EXPRESIONES

Puedes incluir cualquier expresión de JavaScript dentro de una plantilla



SINTAXIS

- Se utilizan backticks (`) en lugar de comillas simples o dobles para definir una plantilla

```
1
2 const nombre = "Juan";
3 const edad = 30;
4
5 // Usando template literals
6 const mensaje = `Hola, mi nombre es ${nombre} y tengo ${edad} años.`;
7
8 console.log(mensaje);
9 // Salida: Hola, mi nombre es Juan y tengo 30 años.
```

VENTAJAS

INTERPOLACIÓN DE VARIABLES

Como en el ejemplo anterior, puedes insertar variables directamente en las plantillas utilizando \${}

INSERCIÓN DE EXPRESIONES

Puedes ejecutar cualquier expresión dentro de las plantillas

```
1 const x = 10;
2 const y = 20;
3 console.log(`El resultado de sumar x e y es: ${x + y}`);
4 // Salida: El resultado de sumar x e y es: 30
5
6
```

INSERCIÓN DE FUNCIONES DENTRO DE EXPRESIONES

También es posible invocar funciones dentro de una plantilla literal

```
1
2 function saludo() {
3     return "Hola Mundo!";
4 }
5
6 console.log(`El saludo es: ${saludo()}`);
7
```

STRINGS REGULARES O EXPRESIONES REGULARES (REGEX)

STRINGS REGULARES O EXPRESIONES REGULARES (REGEX)

Las expresiones regulares (o RegEx) son patrones utilizados para hacer coincidir combinaciones de caracteres en cadenas de texto. Se pueden utilizar para búsqueda y manipulación de texto.

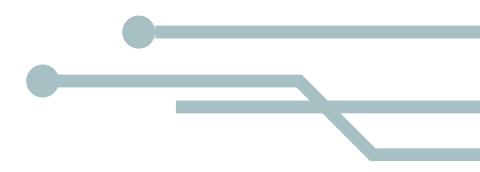
Características de las Expresiones Regulares

- Pueden buscar patrones específicos dentro de strings.
- Se utilizan para validar, buscar, reemplazar o extraer texto.
- Suelen ser parte del manejo avanzado de cadenas y son comunes en casi todos los lenguajes de programación.

SINTAXIS BÁSICA DE EXPRESIONES REGULARES

Una expresión regular se escribe entre barras inclinadas (/patrón/). Las expresiones regulares tienen una sintaxis específica que incluye metacaracteres y cuantificadores.

```
1
2 const regex = /hola/;
3 const texto = "hola, ¿cómo estás?";
4
5 console.log(regex.test(texto));
6 // Salida: true (porque "hola" aparece en el string)
7
```



METACARACTERES COMUNES

METACARACTERES COMUNES

- **.:** Coincide con cualquier carácter, excepto nuevas líneas.
- **\d:** Coincide con cualquier dígito (0-9).
- **\w:** Coincide con cualquier carácter alfanumérico (letras, números o guión bajo).
- **\s:** Coincide con cualquier espacio en blanco.
- **^:** Coincide con el inicio de una cadena.
- **\$:** Coincide con el final de una cadena

CUANTIFICADORES

- *****: Coincide con cero o más repeticiones del carácter o grupo anterior.
- **+**: Coincide con una o más repeticiones del carácter o grupo anterior.
- **?**: Coincide con cero o una repetición del carácter o grupo anterior.
- **{n}**: Coincide con exactamente n repeticiones del carácter o grupo anterior.

EJEMPLO CON METACARACTERES Y CUANTIFICADORES

METACARACTERES COMUNES

```
1
2 const regex = /\d{3}-\d{3}-\d{4}/;
3 const texto = "Mi número de teléfono es 123-456-7890";
4
5 console.log(regex.test(texto));
6 // Salida: true (porque coincide con el formato de un número de teléfono)
7 |
```

EJEMPLO CON METACARACTERES Y CUANTIFICADORES

MÉTODOS COMUNES EN JAVASCRIPT PARA EXPRESIONES REGULARES

test(): Retorna true o false si una cadena coincide con la expresión regular

```
1  
2  
3 const regex = /perro/;  
4 console.log(regex.test("Tengo un perro")); // true  
5  
6 |
```

match(): Busca coincidencias en una cadena y las devuelve en un array.

```
1  
2  
3 const texto = "El perro y el gato son amigos";  
4 const regex = /perro|gato/g;  
5 console.log(texto.match(regex));  
6 // Salida: ["perro", "gato"]  
7 |
```

EJEMPLO CON METACARACTERES Y CUANTIFICADORES

MÉTODOS COMUNES EN JAVASCRIPT PARA EXPRESIONES REGULARES

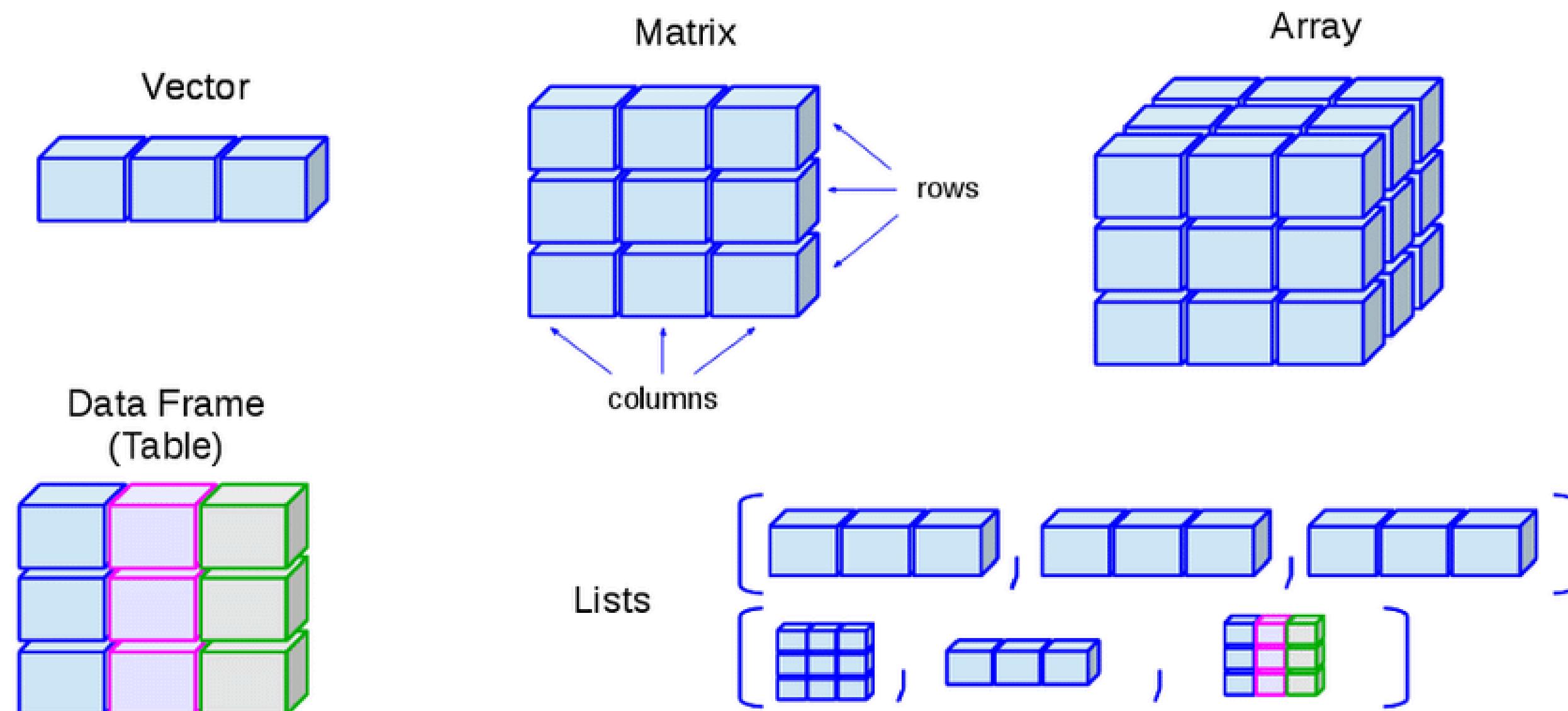
replace(): Busca y reemplaza texto que coincide con la expresión regular.

```
1
2 const texto = "2024 es un buen año";
3 const regex = /\d{4}/;
4 const nuevoTexto = texto.replace(regex, "2023");
5 console.log(nuevoTexto);
6 // Salida: 2023 es un buen año
7
```

ESTRUCTURAS DÉ DATOS, MANIPULACIÓN DÉ OBJETOS Y ARREGLOS.

ESTRUCTURAS DE DATOS

Una estructura de datos en programación es una manera organizada y eficiente de almacenar y gestionar datos para que puedan ser utilizados y procesados de manera efectiva. Cada estructura de datos está diseñada para organizar la información según sus características y el tipo de operaciones que se quieran realizar, como insertar, eliminar, buscar o actualizar datos.

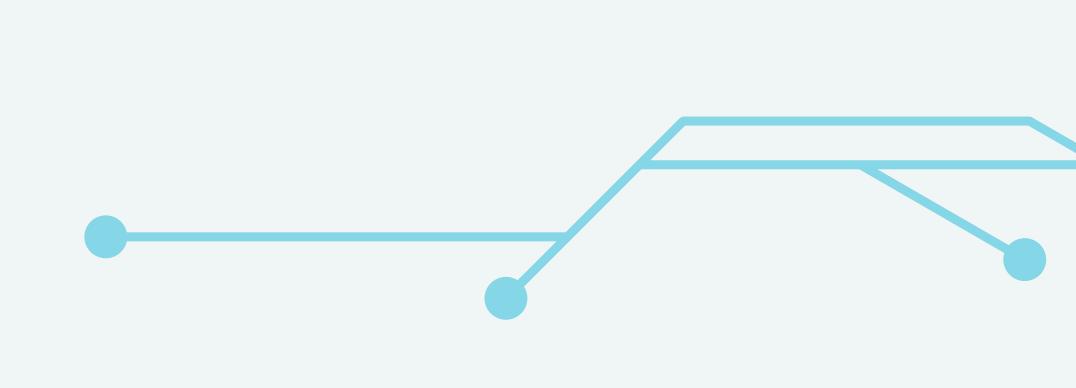
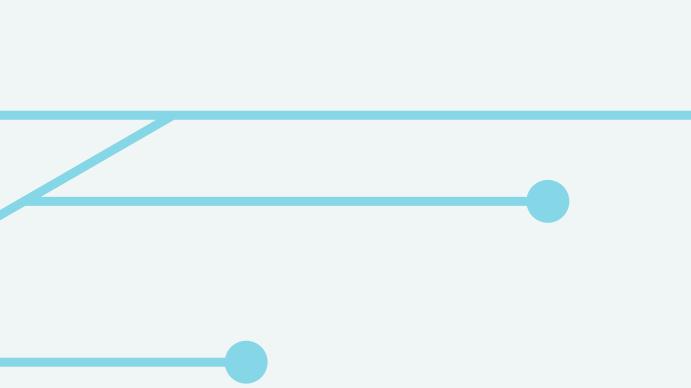


TIPOS DE ESTRUCTURAS DE DATOS EN JAVASCRIPT

ARRAYS

- Los arrays son listas ordenadas de elementos, pueden almacenar diferentes tipos de datos (números, strings, objetos, etc)

```
const frutas = ['Manzana', 'Banana', 'Cereza'];
frutas.push('Durazno'); // Agregar al final
frutas.pop(); // Eliminar el último
frutas.forEach(fruta => console.log(fruta)); // Recorrer el array
```



MUCHAS GRACIAS

