1. DOM

El DOM (Document Object Model) es una representación en forma de árbol de la estructura de un documento HTML o XML. En JavaScript, el DOM es una interfaz que permite interactuar con los elementos y contenido de una página web. Proporciona métodos y propiedades para acceder, modificar y manipular dinámicamente los elementos HTML, estilos, atributos y eventos.

El DOM es una representación estructurada del documento HTML, donde cada elemento HTML es un objeto dentro del árbol del DOM. Los objetos del DOM se llaman nodos y existen diferentes tipos de nodos, como el nodo de elemento, nodo de texto, nodo de atributo, entre otros.

Hay varias formas de acceder a los elementos del DOM en JavaScript.

1.1. Acceso por ID:

Podés acceder a un elemento del DOM utilizando su identificador único (ID). Utilizá el método getElementById() y proporciona el ID del elemento.

```
var elemento = document.getElementById("miElemento");
```

1.2. Acceso por clase:

Podés acceder a elementos del DOM utilizando su clase. Utilizá el método getElementsByClassName() y proporcioná el nombre de la clase.

```
var elementos = document.getElementsByClassName("miClase");
```

Este método devuelve una colección de elementos con la clase especificada. Puedes acceder a elementos individuales utilizando índices, por ejemplo, elementos[0].

1.3. Acceso por etiqueta:

Podés acceder a elementos del DOM utilizando su etiqueta HTML. Utilizá el método getElementsByTagName() y proporcioná el nombre de la etiqueta.

```
var elementos = document.getElementsByTagName("p");
```

Este método devuelve una colección de elementos con la etiqueta especificada. Al igual que con getElementsByClassName(), podés acceder a elementos individuales utilizando índices.

1.4. Acceso por selectores CSS:

Podés acceder a elementos del DOM utilizando selectores CSS. Utilizá el método querySelector() para acceder al primer elemento que coincida con

el selector, o querySelectorAll() para acceder a todos los elementos que coincidan con el selector.

```
var elemento = document.querySelector("#miElemento");

var elementos = document.querySelectorAll(".miClase");
```

En el ejemplo anterior, querySelector() accede al primer elemento con el ID "miElemento", mientras que querySelectorAll() accede a todos los elementos con la clase "miClase".

2. Modificación DOM

Podés modificar el contenido de los elementos del DOM utilizando JavaScript. Hay varias formas de hacerlo, dependiendo de lo que desees lograr. Acá tenés algunos métodos comunes para modificar el contenido del DOM:

2.1. Modificar el texto:

Podés modificar el texto dentro de un elemento utilizando la propiedad textContent o innerText.

```
var elemento = document.getElementById("miElemento");

elemento.textContent = "Nuevo texto";
```

En este ejemplo, el texto dentro del elemento con el ID "miElemento" se cambia a "Nuevo texto".

2.2. Modificar el HTML:

Si deseás modificar el HTML dentro de un elemento, podés utilizar la propiedad innerHTML.

```
var elemento = document.getElementById("miElemento");

elemento.innerHTML = "<strong>Nuevo contenido</strong>";
```

En este caso, el contenido HTML dentro del elemento con el ID "miElemento" se reemplaza con el texto "Nuevo contenido" en negrita ().

2.3. Modificar atributos:

Podés modificar los atributos de un elemento utilizando el método setAttribute().

```
var imagen = document.getElementById("miImagen");

imagen.setAttribute("src", "nueva-imagen.jpg");
```

En este ejemplo, el atributo src de la imagen con el ID "miImagen" se cambia a "nueva-imagen.jpg".

2.4. Modificar estilos:

Podés modificar los estilos de un elemento utilizando la propiedad style. Por ejemplo, para cambiar el color de fondo:

```
var elemento = document.getElementById("miElemento");

elemento.style.backgroundColor = "red";
```

En este caso, el color de fondo del elemento con el ID "miElemento" se cambia a rojo.

3. Creación y Eliminación de elementos DOM

Podés crear y eliminar elementos del DOM utilizando JavaScript.

3.1. Crear elementos:

Para crear un nuevo elemento, utilizá el método createElement() y proporcioná el nombre de la etiqueta del elemento que deseás crear. Luego, podés configurar sus atributos y contenido según sea necesario, y finalmente, agregarlo al DOM.

```
// Crear un nuevo elemento <p>

var nuevoParrafo = document.createElement("p");


// Configurar atributos

nuevoParrafo.setAttribute("id", "miParrafo");

nuevoParrafo.setAttribute("class", "miClase");


// Configurar contenido

nuevoParrafo.textContent = "Este es un nuevo párrafo.";


// Agregar el elemento al DOM

var contenedor = document.getElementById("miContenedor");

contenedor.appendChild(nuevoParrafo);
```

En este ejemplo, se crea un nuevo elemento <p>, se le asignan atributos y contenido, y luego se agrega al contenedor con el ID "miContenedor".

3.2. Eliminar elementos:

Para eliminar un elemento del DOM, primero debés acceder a ese elemento y luego utilizar el método remove() o removeChild().

```
// Eliminar un elemento específico
var elementoEliminar = document.getElementById("elementoEliminar");
elementoEliminar.remove();


// Eliminar un elemento hijo de otro elemento
var contenedor = document.getElementById("miContenedor");
var elementoHijo = contenedor.firstChild;
contenedor.removeChild(elementoHijo);
```

En el primer ejemplo, se elimina directamente el elemento con el ID "elementoEliminar" utilizando el método remove(). En el segundo ejemplo, se elimina el primer elemento hijo del contenedor con el ID "miContenedor" utilizando removeChild().

Recordá que al eliminar un elemento, también se eliminan todos sus hijos y cualquier otro evento o referencia asociada a él.

Utilizando estos métodos, puedes crear nuevos elementos y agregarlos al DOM, así como eliminar elementos existentes según sea necesario para manipular la estructura y el contenido de la página web.

4. Arreglos

En JavaScript, un arreglo (también conocido como array) es una estructura de datos que te permite almacenar y acceder a múltiples valores en una sola variable. Puedes pensar en un arreglo como una lista ordenada de elementos. Cada elemento en un arreglo tiene un índice numérico que indica su posición.

4.1. Creación de un arreglo:

Podés crear un arreglo en JavaScript utilizando corchetes [] y separando los elementos por comas. Por ejemplo:

```
var miArreglo = [1, 2, 3, 4, 5];
```

En este caso, hemos creado un arreglo llamado miArreglo con cinco elementos numéricos.

4.2. Acceso a elementos del arreglo:

Podés acceder a los elementos de un arreglo utilizando su índice numérico entre corchetes []. El primer elemento tiene un índice de 0, el segundo tiene un índice de 1, y así sucesivamente. Por ejemplo:

```
var miArreglo = [1, 2, 3, 4, 5];

console.log(miArreglo[0]); // Accede al primer elemento (1)

console.log(miArreglo[2]); // Accede al tercer elemento (3)
```

En este ejemplo, accedemos al primer y tercer elemento del arreglo miArreglo utilizando sus índices (0 y 2 respectivamente).

4.3. Modificación de elementos del arreglo:

Podés modificar los elementos de un arreglo asignando nuevos valores a sus índices. Por ejemplo:

```
var miArreglo = [1, 2, 3, 4, 5];

miArreglo[2] = 10; // Modifica el tercer elemento y le asigna el valor 10


console.log(miArreglo); // Muestra el arreglo modificado
```

En este caso, hemos modificado el tercer elemento del arreglo miArreglo y le hemos asignado el valor 10.

4.4. Longitud del arreglo:

Podés obtener la longitud de un arreglo utilizando la propiedad length. Por ejemplo:

```
var miArreglo = [1, 2, 3, 4, 5];


console.log(miArreglo.length); // Muestra la longitud del arreglo (5)
```

En este ejemplo, utilizamos la propiedad length para obtener la cantidad de elementos en el arreglo miArreglo.

Recordá que los índices de los elementos de un arreglo comienzan en 0 y van hasta length - 1, donde length es la longitud del arreglo.

4.5. Ordenar

El método sort() se utiliza para ordenar los elementos de un arreglo alfabéticamente o numéricamente. Modifica el arreglo original y devuelve el arreglo ordenado.

```
var frutas = ["naranja", "manzana", "plátano", "uva"];

frutas.sort();

console.log(frutas);  // Muestra el arreglo ordenado: ["manzana", "naranja", "plátano", "uva"]
```

En este ejemplo, el método sort() se utiliza para ordenar los elementos del arreglo frutas en orden alfabético.

4.6. Push

El método push() se utiliza para agregar uno o más elementos al final de un arreglo. Modifica el arreglo original y devuelve la nueva longitud del arreglo.

```
var numeros = [1, 2, 3];

numeros.push(4);


console.log(numeros); // Muestra el arreglo modificado: [1, 2, 3, 4]
```

En este caso, utilizamos el método push() para agregar el número 4 al final del arreglo numeros.

4.7. Pop

El método pop() se utiliza para eliminar el último elemento de un arreglo. Modifica el arreglo original y devuelve el elemento eliminado.

```
var numeros = [1, 2, 3, 4];

var ultimoElemento = numeros.pop();


console.log(numeros); // Muestra el arreglo modificado: [1, 2, 3]

console.log(ultimoElemento);  // Muestra el elemento eliminado: 4
```

En este ejemplo, utilizamos el método pop() para eliminar el último elemento del arreglo números y almacenamos el valor eliminado en la variable ultimoElemento.

4.8. Recorre con for

El bucle for clásico te permite recorrer un arreglo utilizando un contador y accediendo a los elementos utilizando su índice.

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


for (var i = 0; i < numeros.length; i++) {
  console.log(numeros[i]);
}
```

En este ejemplo, utilizamos un bucle for para recorrer el arreglo números. La variable i actúa como contador, y accedemos a los elementos del arreglo utilizando números[i].

4.9. Recorrer con for...of

El bucle for...of es una sintaxis más moderna que te permite recorrer directamente los elementos de un arreglo sin la necesidad de utilizar un contador o índices.

```
var numeros = [1, 2, 3, 4, 5];
for (var numero of numeros) {
  console.log(numero);
}
```

En este ejemplo, utilizamos el bucle for...of para recorrer el arreglo números. En cada iteración, la variable número toma el valor de cada elemento del arreglo.

4.10. Recorrer con forEach:

El método forEach es una función de los arreglos que te permite ejecutar una función para cada elemento del arreglo. Recibe una función de devolución de llamada como argumento.

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


numeros.forEach(function(numero) {
  console.log(numero);
});
```

En este caso, utilizamos el método forEach en el arreglo números. La función de devolución de llamada se ejecutará para cada elemento del arreglo, y en cada iteración, la variable numero contendrá el valor del elemento actual.

5. Objetos

En JavaScript, un objeto es una entidad que agrupa propiedades y métodos relacionados en una estructura de datos. Los objetos son fundamentales en JavaScript, ya que permiten modelar entidades del mundo real y organizar datos de manera estructurada. Puedes pensar en un objeto como una colección de pares clave-valor, donde las claves son las propiedades y los valores son los datos asociados a esas propiedades.

5.1. Declaración de un objeto literal:

La forma más común de crear un objeto en JavaScript es utilizando la notación de objeto literal. Un objeto literal se define utilizando llaves {} y se especifican las propiedades y sus valores dentro del objeto.

```
var persona = {
  nombre: "Juan",
  edad: 30,
  profesion: "Desarrollador",
  saludar: function() {
    console.log("Hola, soy " + this.nombre);
  }
};
```

En este ejemplo, hemos declarado un objeto llamado persona. El objeto tiene propiedades como nombre, edad y profesión, y también incluye un método saludar que muestra un mensaje en la consola.

5.2. Acceso a las propiedades del objeto:

Podés acceder a las propiedades de un objeto utilizando la notación de punto (objeto.propiedad) o la notación de corchetes (objeto["propiedad"]).

```
console.log(persona.nombre); // Accede a la propiedad nombre (Juan)

console.log(persona["edad"]); // Accede a la propiedad edad (30)
```

En este caso, accedemos a las propiedades nombre y edad del objeto persona utilizando ambas notaciones.

5.3. Modificación de propiedades del objeto:

Podés modificar los valores de las propiedades de un objeto asignando nuevos valores.

```
persona.nombre = "María"; // Modifica el valor de la propiedad nombre

persona.edad = 35; // Modifica el valor de la propiedad edad
```

En este ejemplo, hemos modificado los valores de las propiedades nombre y edad del objeto persona.

5.4. Funciones para crear objetos

En JavaScript, puedes utilizar funciones para crear objetos utilizando el concepto de constructor de objetos. Los constructores de objetos son funciones que se utilizan como plantillas para crear nuevos objetos con propiedades y métodos específicos.

```
// Definir una función constructora
function Persona(nombre, edad) {
  this.nombre = nombre;
  this.edad = edad;
  this.saludar = function() {
    console.log("Hola, soy " + this.nombre);
  };
}

// Crear un objeto utilizando la función constructora
var persona1 = new Persona("Juan", 30);
var persona2 = new Persona("María", 35);

console.log(persona1.nombre); // Acceder al atributo nombre del objeto persona1
persona2.saludar(); // Invocar el método saludar del objeto persona2
```

En este ejemplo, la función Persona se utiliza como una función constructora para crear objetos que representan personas. Dentro de la función constructora, se definen las propiedades (nombre y edad) y el método (saludar) utilizando la referencia this. Cuando se crea un nuevo objeto utilizando new Persona(...), se ejecuta la función constructora y se asignan los valores proporcionados a las propiedades del objeto.

Podés crear tantos objetos como desees utilizando la función constructora, y cada objeto creado tendrá sus propias propiedades y métodos. Los métodos definidos dentro de la función constructora comparten la misma lógica, pero tienen acceso a las propiedades específicas de cada objeto utilizando this.

Utilizando funciones constructoras, podés crear objetos con estructuras y comportamientos específicos y reutilizar la misma definición de objeto para crear múltiples instancias con diferentes valores. Esto es un concepto fundamental en la programación orientada a objetos.

6. JSON

JSON (JavaScript Object Notation) es un formato de intercambio de datos ligero y legible por humanos. Se utiliza para transmitir datos estructurados entre un servidor y una aplicación web, o entre diferentes sistemas.

En JavaScript, JSON se utiliza para representar datos estructurados en forma de objetos y arreglos. Los datos en formato JSON están compuestos por pares clave-valor, donde las claves son cadenas de texto y los valores pueden ser de cualquier tipo de dato válido en JSON (objetos, arreglos, cadenas, números, booleanos o null).

JSON es ampliamente utilizado en JavaScript debido a su sencillez y facilidad de uso. Algunos casos comunes de uso de JSON en JavaScript incluyen:

- Intercambio de datos: JSON se utiliza para transmitir datos entre un servidor y una aplicación web, permitiendo una comunicación eficiente y estructurada. Por ejemplo, al realizar solicitudes AJAX a un servidor y recibir respuestas en formato JSON.
- Almacenamiento de datos: JSON se puede utilizar para almacenar datos estructurados en archivos o en bases de datos. Esto permite guardar y recuperar datos de forma persistente.
- Configuración: JSON se utiliza para almacenar configuraciones o preferencias de una aplicación en un formato legible y fácil de editar.
- Serialización y deserialización de objetos: JSON se utiliza para convertir objetos JavaScript en una cadena JSON (serialización) y

viceversa (deserialización). Esto es útil para transmitir o almacenar objetos de forma compacta y luego recuperarlos.

En JavaScript, puedes convertir objetos o arreglos en una cadena JSON utilizando el método JSON.stringify(), y puedes convertir una cadena JSON en un objeto o arreglo utilizando el método JSON.parse().

```
// Convertir un objeto a JSON
var objeto = { "nombre": "Juan", "edad": 30 };
var jsonString = JSON.stringify(objeto);

console.log(jsonString);              //              Muestra
'{"nombre":"Juan","edad":30}'

// Convertir una cadena JSON a un objeto
var json = '{"nombre":"María","edad":35}';
var objetoRecuperado = JSON.parse(json);

console.log(objetoRecuperado.nombre); // Muestra 'María'
console.log(objetoRecuperado.edad); // Muestra 35
```

En resumen, JSON es un formato de intercambio de datos ampliamente utilizado en JavaScript para representar y transmitir datos estructurados de manera eficiente. Proporciona una forma común y legible de organizar y compartir información entre diferentes sistemas.

6.1. LocalStorage

LocalStorage es una API (Application Programming Interface) de JavaScript que proporciona métodos para almacenar y recuperar datos en forma de pares clave-valor. Los datos almacenados en LocalStorage están disponibles incluso cuando se cierra y se vuelve a abrir el navegador, a menos que sean eliminados explícitamente por la aplicación o el usuario.

Algunos casos comunes de uso de LocalStorage incluyen:

- Almacenamiento de preferencias del usuario: Puedes utilizar LocalStorage para guardar las preferencias del usuario, como el tema preferido, el idioma o las configuraciones personalizadas.
- Almacenamiento de datos de sesión: LocalStorage puede ser utilizado para almacenar información de sesión del usuario, como el token de autenticación, información del carrito de compras o detalles del usuario durante una sesión.
- Almacenamiento de datos en caché: Puedes utilizar LocalStorage para almacenar en caché datos importantes para tu aplicación web, como resultados de consultas a la API o datos estáticos que se utilizan con frecuencia.

```
// Almacenar datos en LocalStorage
localStorage.setItem("clave", "valor");

// Recuperar datos de LocalStorage
var valor = localStorage.getItem("clave");

// Eliminar datos de LocalStorage
localStorage.removeItem("clave");

// Borrar todos los datos de LocalStorage
localStorage.clear();
```

En este ejemplo, se utilizan los métodos setItem(), getItem(), removeItem() y clear() de LocalStorage para almacenar, recuperar, eliminar y borrar datos respectivamente. Puedes utilizar una clave única para cada par clave-valor almacenado en LocalStorage.

Es importante tener en cuenta que los datos almacenados en LocalStorage están limitados al dominio específico y al tamaño máximo de almacenamiento establecido por el navegador. Además, los datos en LocalStorage se almacenan como cadenas de texto, por lo que es posible que necesites convertirlos a otros formatos si deseas trabajar con tipos de datos más complejos.

LocalStorage es una herramienta útil para el almacenamiento persistente de datos en el lado del cliente y puede ser utilizada para mejorar la experiencia del usuario y el rendimiento de las aplicaciones web.