1. Selección de elementos en el DOM

Para seleccionar elementos en el documento HTML:

document.getElementById(id)

Selecciona un elemento por su id.

```
const elemento = document.getElementById('miId');
console.log(elemento); // Muestra el elemento con id="miId"
```

document.getElementsByClassName(className)

Selecciona todos los elementos con una clase específica.

```
const elementos = document.getElementsByClassName('miClase');
console.log(elementos); // Muestra una colección de elementos
con class="miClase"
```

document.getElementsByTagName(tagName)

Selecciona todos los elementos con un nombre de etiqueta específico.

```
const parrafos = document.getElementsByTagName('p');
console.log(parrafos); // Muestra una colección de todos los
elementos <p>
```

document.querySelector(selector)

Selecciona el primer elemento que coincide con un selector CSS.

```
const primerParrafo = document.querySelector('p');
console.log(primerParrafo); // Muestra el primer elemento <p>
```

document.querySelectorAll(selector)

Selecciona todos los elementos que coinciden con un selector CSS.

```
const todosParrafos = document.querySelectorAll('p');
console.log(todosParrafos); // Muestra una NodeList de todos
los elementos <p>
```

2. Modificación de contenido

Para modificar el contenido de los elementos:

element.innerHTML

Obtiene o establece el contenido HTML de un elemento.

```
const div = document.getElementById('miDiv');
div.innerHTML = '<p>Nuevo contenido</p>'; // Cambia el
contenido del div
```

element.textContent

Obtiene o establece el contenido de texto de un elemento.

```
const div = document.getElementById('miDiv');
div.textContent = 'Texto nuevo'; // Cambia el texto del div
```

element.value

Obtiene o establece el valor de un elemento de formulario.

```
const input = document.getElementById('miInput');
console.log(input.value); // Muestra el valor del input
input.value = 'Nuevo valor'; // Cambia el valor del input
```

element.innerText

Similar a textContent, pero respeta estilos CSS.

```
const parrafo = document.getElementById('miParrafo');
parrafo.innerText = '¡Nuevo texto!'; // Cambia el texto del
párrafo
```

3. Modificación de atributos

Para cambiar o manipular atributos de los elementos:

element.getAttribute(attr)

Obtiene el valor de un atributo.

```
const link = document.getElementById('miLink');
console.log(link.getAttribute('href')); // Muestra el valor
del atributo href
```

element.setAttribute(attr, value)

Establece el valor de un atributo.

```
const link = document.getElementById('miLink');
link.setAttribute('href', 'https://nuevo-enlace.com'); //
Cambia el atributo href
```

element.removeAttribute(attr)

Elimina un atributo.

```
const link = document.getElementById('miLink');
link.removeAttribute('target'); // Elimina el atributo target
```

element.classList

Manipula las clases de un elemento.

```
const div = document.getElementById('miDiv');
div.classList.add('nuevaClase'); // Añade una clase
div.classList.remove('viejaClase'); // Elimina una clase
div.classList.toggle('activo'); // Alterna una clase
console.log(div.classList.contains('activo')); // Verifica si
tiene la clase
```

4. Modificación de estilos CSS

Para modificar estilos de los elementos:

element.style.property

Obtiene o establece una propiedad de estilo en línea.

```
const div = document.getElementById('miDiv');
div.style.color = 'red'; // Cambia el color del texto a rojo
```

window.getComputedStyle(element)

Obtiene los estilos calculados de un elemento.

```
const div = document.getElementById('miDiv');
const estilos = window.getComputedStyle(div);
console.log(estilos.color); // Muestra el color calculado del
div
```

5. Manipulación del DOM (Crear, Insertar y Eliminar elementos)

Para crear y manipular elementos:

document.createElement(tagName)

Crea un nuevo elemento HTML.

```
const nuevoParrafo = document.createElement('p');
nuevoParrafo.textContent = 'Este es un nuevo párrafo';
document.body.appendChild(nuevoParrafo); // Añade el párrafo
al final del body
```

element.removeChild(childElement)

Elimina un hijo específico de un elemento.

```
const padre = document.getElementById('padre');
const hijo = document.getElementById('hijo');
padre.removeChild(hijo); // Elimina el elemento hijo
```

6. Eventos en el DOM

Para manejar eventos en los elementos:

element.addEventListener(event, callback)

Añade un listener de eventos a un elemento.

```
const boton = document.getElementById('miBoton');
boton.addEventListener('click', () => {
    alert('¡Hiciste clic!');
});
```

event.preventDefault()

Evita el comportamiento por defecto de un evento.

```
const form = document.getElementById('miForm');
form.addEventListener('submit', (event) => {
    event.preventDefault(); // Evita que el formulario se
envíe
    console.log('Formulario no enviado');
});
```

7. Navegación en el DOM

Para navegar entre los nodos del DOM:

element.parentNode

Obtiene el nodo padre de un elemento.

```
const hijo = document.getElementById('hijo');
console.log(hijo.parentNode); // Muestra el padre del
elemento
```

element.childNodes

Obtiene una lista de todos los nodos hijos.

```
const padre = document.getElementById('padre');
console.log(padre.childNodes); // Muestra todos los hijos del
elemento
```

8. Manipulación de formularios

Para manejar formularios y sus entradas:

formElement.submit()

Envía un formulario.

```
const form = document.getElementById('miForm');
form.submit(); // Envía el formulario
```

inputElement.focus()

Enfoca un elemento de formulario.

```
const input = document.getElementById('miInput');
input.focus(); // Enfoca el input
```

9. LocalStorage y SessionStorage

Para almacenar datos en el navegador:

localStorage.setItem("clave", "valor")

Guarda un dato de forma permanente.

```
localStorage.setItem('miDato', 'valor');
```

localStorage.getItem("clave")

Obtiene el dato guardado.

```
const dato = localStorage.getItem('miDato');
console.log(dato); // Muestra el valor guardado
```

10. Fetch API (Peticiones HTTP)

Para hacer solicitudes HTTP:

fetch(url)

Realiza una solicitud HTTP.

```
fetch('https://jsonplaceholder.typicode.com/posts')
    .then(response => response.json())
    .then(data => console.log(data))
    .catch(error => console.error('Error:', error));
```

11. Manipulación de ventana y documento

Para manipular la ventana y el documento:

window.alert(message)

Muestra una alerta en el navegador.

```
window.alert('¡Hola, mundo!'); // Muestra una alerta
```

document.title

Obtiene o establece el título del documento.

```
document.title = 'Nuevo título'; // Cambia el título de la
página
```

12. Métodos útiles adicionales

Para realizar operaciones adicionales:

element.matches(selector)

Verifica si un elemento coincide con un selector CSS.

```
const div = document.getElementById('miDiv');
console.log(div.matches('.miClase')); // Verifica si el div
tiene la clase "miClase"
```

element.insertAdjacentHTML(position, text)

Inserta HTML en una posición relativa al elemento.

```
const div = document.getElementById('miDiv');
```

```
div.insertAdjacentHTML('beforeend', '<p>Nuevo párrafo</p>');
// Inserta HTML al final del div
```

13. Manipulación de eventos avanzados

Para manejar eventos de manera más avanzada:

event.stopPropagation()

Detiene la propagación del evento a otros elementos.

```
const padre = document.getElementById('padre');
const hijo = document.getElementById('hijo');
padre.addEventListener('click', () => console.log('Padre
clickeado'));
hijo.addEventListener('click', (event) => {
    event.stopPropagation(); // Evita que el evento se
propague al padre
    console.log('Hijo clickeado');
});
```

event.target

Obtiene el elemento que activó el evento.

```
document.addEventListener('click', (event) => {
    console.log('Elemento clickeado:', event.target);
});
```

14. Manipulación de clases avanzada

Para trabajar con clases de manera más dinámica:

element.classList.replace(oldClass, newClass)

Reemplaza una clase por otra.

```
const div = document.getElementById('miDiv');
div.classList.replace('viejaClase', 'nuevaClase'); //
Reemplaza la clase
```

15. Manipulación de nodos de texto

Para trabajar con nodos de texto:

document.createTextNode(text)

Crea un nodo de texto.

```
const nuevoTexto = document.createTextNode('Este es un nuevo
texto');
document.body.appendChild(nuevoTexto); // Añade el texto al
final del body
```

16. Manipulación de iframes

Para trabajar con iframes:

document.getElementById('miIframe').contentWindow

Accede al contenido de un iframe.

```
const iframe = document.getElementById('miIframe');
const iframeWindow = iframe.contentWindow;
console.log(iframeWindow.document.body.innerHTML); // Accede
al contenido del iframe
```

17. Manipulación de eventos de teclado

Para manejar eventos de teclado:

keydown y keyup

Detecta cuando una tecla es presionada o liberada.

```
document.addEventListener('keydown', (event) => {
    console.log('Tecla presionada:', event.key);
});

document.addEventListener('keyup', (event) => {
    console.log('Tecla liberada:', event.key);
});
```

18. Manipulación de eventos de ratón

Para manejar eventos de ratón:

mouseover y mouseout

Detecta cuando el cursor entra o sale de un elemento.

```
const div = document.getElementById('miDiv');
div.addEventListener('mouseover', () => console.log('Cursor
sobre el div'));
div.addEventListener('mouseout', () => console.log('Cursor
fuera del div'));
```

19. Manipulación de eventos de formularios

Para manejar eventos específicos de formularios:

input y change

Detecta cambios en los elementos de formulario.

```
const input = document.getElementById('miInput');
input.addEventListener('input', () => console.log('Input
cambiado'));
input.addEventListener('change', () => console.log('Input
modificado y confirmado'));
```

20. Manipulación de animaciones

Para trabajar con animaciones:

element.animate(keyframes, options)

Crea una animación en un elemento.

```
const div = document.getElementById('miDiv');
div.animate(
    [{ transform: 'translateX(0px)' }, { transform:
'translateX(100px)' }],
    { duration: 1000, iterations: Infinity }
);
```

21. Manipulación de elementos multimedia

Para trabajar con elementos multimedia como audio y video:

play() y pause()

Controla la reproducción de elementos multimedia.

```
const video = document.getElementById('miVideo');
video.play(); // Reproduce el video
video.pause(); // Pausa el video
```

22. Manipulación de elementos SVG

Para trabajar con gráficos SVG:

document.createElementNS(namespace, tagName)

Crea un elemento SVG.

```
const svgNS = 'http://www.w3.org/2000/svg';
const circle = document.createElementNS(svgNS, 'circle');
circle.setAttribute('cx', '50');
circle.setAttribute('cy', '50');
circle.setAttribute('r', '40');
circle.setAttribute('fill', 'red');
document.getElementById('miSvg').appendChild(circle);
```

23. Manipulación de eventos de arrastrar y soltar (Drag and Drop)

Para manejar eventos de arrastrar y soltar:

dragstart, dragover, y drop

Controla el comportamiento de arrastrar y soltar.

```
const elemento = document.getElementById('miElemento');
const destino = document.getElementById('destino');
```

```
elemento.addEventListener('dragstart', (event) => {
    event.dataTransfer.setData('text/plain', 'Arrastrando');
});

destino.addEventListener('dragover', (event) => {
    event.preventDefault(); // Permite soltar el elemento
});

destino.addEventListener('drop', (event) => {
    event.preventDefault();
    console.log('Elemento soltado:',
event.dataTransfer.getData('text/plain'));
});
```

24. Manipulación de elementos de lista (HTMLCollection y NodeList)

Para trabajar con colecciones de elementos:

Array.from()

Convierte una colección HTML o NodeList en un array.

```
const elementos = document.getElementsByClassName('miClase');
const arrayElementos = Array.from(elementos);
arrayElementos.forEach(elemento => console.log(elemento));
```

25. Manipulación de eventos personalizados

Para crear y manejar eventos personalizados:

CustomEvent

Crea y dispara eventos personalizados.

```
const eventoPersonalizado = new CustomEvent('miEvento', {
detail: { mensaje: 'Hola' } });
document.addEventListener('miEvento', (event) => {
    console.log('Evento personalizado:',
event.detail.mensaje);
});
document.dispatchEvent(eventoPersonalizado); // Dispara el
evento
```

26. Manipulación de elementos de tiempo (Timers)

Para trabajar con temporizadores:

setTimeout y setInterval

Ejecuta código después de un tiempo o de manera repetida.

```
setTimeout(() => console.log('Hola después de 2 segundos'),
2000);
```

```
setInterval(() => console.log('Hola cada segundo'), 1000);
```

27. Manipulación de elementos de almacenamiento (Cookies)

Para trabajar con cookies:

document.cookie

Obtiene o establece cookies.

```
document.cookie = 'nombre=valor; expires=Fri, 31 Dec 2023
23:59:59 GMT; path=/';
console.log(document.cookie); // Muestra las cookies
```

28. Manipulación de elementos de geolocalización

Para trabajar con la geolocalización:

navigator.geolocation

Obtiene la ubicación del usuario.

```
navigator.geolocation.getCurrentPosition((position) => {
    console.log('Latitud:', position.coords.latitude);
    console.log('Longitud:', position.coords.longitude);
});
```

29. Manipulación de elementos de historial (History API)

Para trabajar con el historial del navegador:

history.pushState() y history.replaceState()

Modifica el historial del navegador.

```
history.pushState({ page: 1 }, 'Título 1', '/pagina1');
history.replaceState({ page: 2 }, 'Título 2', '/pagina2');
```

30. Manipulación de elementos de notificaciones

Para mostrar notificaciones al usuario:

Notification

Muestra notificaciones en el navegador.

```
if (Notification.permission === 'granted') {
    new Notification('Hola', { body: 'Esta es una
notificación' });
} else {
```

```
    Notification.requestPermission();
}
```

Diferencias clave entre Cookies, LocalStorage y SessionStorage

Característica	Cookies	LocalStorage	SessionStorage
Persistencia	Pueden tener fecha de expiración o ser de sesión.	Sin fecha de expiración (persistente).	Solo durante la sesión actual.
Capacidad	4 KB por cookie.	5-10 MB.	5-10 MB.
Alcance	Disponible en todas las pestañas y ventanas del mismo dominio.	Disponible en todas las pestañas y ventanas del mismo origen.	Disponible solo en la pestaña actual.
Envío al servidor	Se envían automáticamente en cada petición HTTP.	No se envían al servidor.	No se envían al servidor.
Uso común	Sesiones, autenticación, rastreo.	Preferencias del usuario, cache.	Datos temporales, estado de la aplicación.

¿Cuándo usar cada uno?

Cookies:

- Cuando necesitas que los datos se envíen automáticamente al servidor (por ejemplo, tokens de autenticación).
- Para mantener sesiones de usuario.
- Para rastrear usuarios (aunque esto puede tener implicaciones de privacidad).

LocalStorage:

- Para almacenar preferencias del usuario (como temas o configuraciones).
- Para cachear datos que no cambian con frecuencia.
- Cuando necesitas persistencia entre sesiones.

SessionStorage:

- Para datos temporales que solo son relevantes durante la sesión actual.
- Para mantener el estado de una aplicación (como un carrito de compras).
- Cuando no quieres que los datos persistan después de cerrar la pestaña.

Ejemplo combinado

Supongamos que tienes una aplicación web donde:

- Usas **cookies** para mantener la sesión del usuario.
- Usas **LocalStorage** para guardar las preferencias del tema.
- Usas **SessionStorage** para almacenar datos temporales del carrito de compras.

```
// Cookies: Mantener la sesión del usuario
document.cookie = "sessionId=abc123; expires=Fri, 31 Dec 2025
23:59:59 GMT; path=/";

// LocalStorage: Guardar preferencias del tema
localStorage.setItem('tema', 'oscuro');

// SessionStorage: Guardar datos temporales del carrito
sessionStorage.setItem('carrito', JSON.stringify([{ id: 1,
nombre: 'Producto A' }]));

// Obtener datos
const sessionId = document.cookie.split('; ').find(row =>
row.startsWith('sessionId')).split('=')[1];
const tema = localStorage.getItem('tema');
const carrito =
JSON.parse(sessionStorage.getItem('carrito'));

console.log(sessionId); // "abc123"
console.log(tema);      // "oscuro"
console.log(carrito);   // [{ id: 1, nombre: 'Producto A' }]
```