

Unidad 4

Manejo del DOM y BOM

Contenido	
Unidad 4	1
Introducción:	4
Objetivos	4
4. Document Object Model (DOM)	5
4.1. ¿Qué es el DOM?.....	5
4.1.1. Las API de navegador	6
4.1.2. Árbol de nodos	6
4.1.3. Veamos el siguiente ejemplo	7
4.2. Acceso a los nodos	7
4.3. Seleccionar elementos del DOM.....	8
4.3.1. Acceso por ID: getElementById()	8
4.3.2. Acceso por etiqueta: getElementsByTagName()	9
4.3.3. Acceso por clase: getElementsByClassName()	10
4.3.4. Acceso por selector CSS o ID: querySelector() / querySelectorAll()	10
4.3.5. Acceso por selector CSS o ID: querySelectorAll()	12
4.4. Modificar textos en el dom	14
4.4.1. La propiedad .textContent	14
4.4.2. La propiedad innerHTML.....	15
4.4.3. La propiedad .outerHTML	16
4.5. Creación de Elementos en el DOM	17
4.5.1. Creación de un Nuevo Elemento:.....	17
4.5.2. Asignación de Contenido al Nuevo Elemento:.....	17
4.5.3. Añadir Atributos al Nuevo Elemento:	17
4.5.4. Adjuntar el Nuevo Elemento al DOM:.....	18
4.6. Eventos.....	19
4.6.1. Tipos de eventos	19
4.6.2. Eventos de Ratón (Mouse Events)	21
4.6.3. Eventos de Foco (Focus Events)	21
4.6.4. Eventos de Teclado (Keyboard Events)	21
4.6.5. Eventos de Formulario (Form Events)	22
4.6.6. Eventos de Documento (Document Events)	22
4.7. Manejadores de eventos.....	22
4.7.1. Apretar una tecla.....	24
4.7.2. Scroll.....	25

4.7.3.	objeto de evento (event)	25
4.7.4.	Ejemplo de una propiedad del Objeto Evento (event) en mousemove: clientX y clientY: 26	
4.7.5.	Practicamos con eventos.....	26
4.8.	Formularios en JS	1
4.8.1.	Experiencia de usuario	1
4.8.2.	Atributos Comunes de los Formularios.....	2
4.8.3.	Atributos de los Elementos de Entrada.....	2
4.8.4.	Estructura Básica de un Formulario	3
4.8.5.	Capturar Eventos de Envío de Formulario	5
4.8.6.	Validación de Formularios.....	6
4.9.	Expresiones regulares	8
4.9.1.	Comprendiendo los Patrones de Expresiones Regulares.....	8
4.9.2.	Cuándo Evitar el Uso de Expresiones Regulares	8
4.9.3.	Partes Fundamentales de una Expresión Regular:.....	8
4.9.4.	Expresiones regulares básicas.....	8
4.9.5.	¿Cómo Crear una Expresión Regular en JavaScript y HTML?.....	10
4.9.6.	Practica Guiada utilización de expresiones regulares en formularios con html.	10
4.9.7.	Practica Guiada, expresión regular para modelo de coche	11
4.9.8.	Herramientas para RegExp.....	12
4.10.	Métodos para Expresiones Regulares:.....	12
4.10.1.	test(cadena):	12
4.10.2.	exec(cadena):	12
4.10.3.	match(expresion):	12
4.10.4.	replace(expresion, reemplazo):	13
4.10.5.	search(expresion):.....	13
4.11.	Ejercicios.....	14
4.12.	Resumen de la Unidad 4.....	16
4.13.	Bibliografía	17

INTRODUCCIÓN:

En esta unidad, exploraremos en profundidad el manejo del DOM y BOM. Aprenderás a seleccionar y manipular elementos del DOM, crear nuevos elementos, y trabajar con eventos para responder a las interacciones del usuario. Además, veremos cómo gestionar formularios y utilizar expresiones regulares para la validación y manipulación de datos.

Esta unidad está diseñada para proporcionar una comprensión integral de cómo JavaScript interactúa con la estructura y el contenido de las páginas web, permitiéndote crear experiencias de usuario más dinámicas y personalizadas.

OBJETIVOS

- Comprender el DOM (Document Object Model):
- Manipular el DOM:
- Crear y Manipular Elementos en el DOM:
- Identificar los diferentes tipos de eventos en JavaScript (mouse, teclado, foco, formulario, etc.).
- Asignar manejadores de eventos y responder a eventos del usuario.
- Utilizar el objeto de evento (event) para obtener información detallada sobre los eventos.
- Gestionar Formularios:
- Utilizar Expresiones Regulares:
- Aplicar expresiones regulares para la validación y manipulación de cadenas en JavaScript.
- Utilizar métodos específicos para trabajar con expresiones regulares (test, exec, match, replace, search).

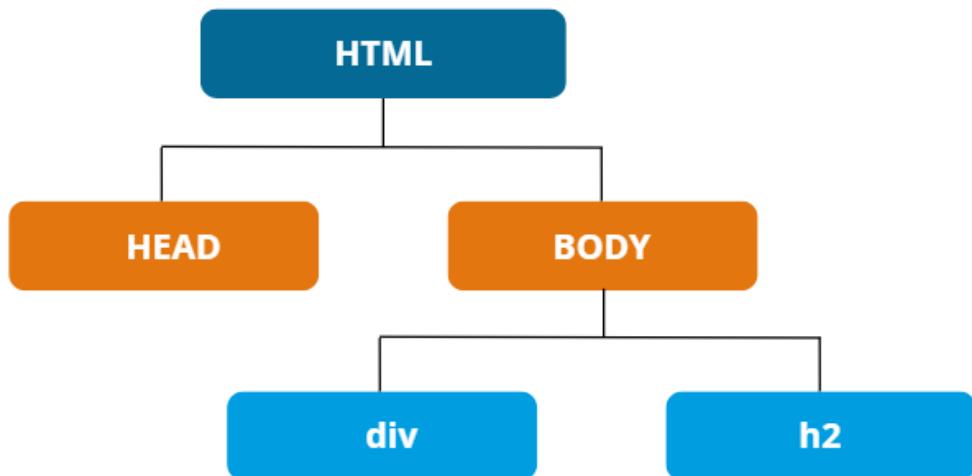
4. Document OBJECT MODEL (DOM)

El Document Object Model (DOM) es una parte esencial de JavaScript que te permite interactuar y manipular la estructura, contenido y presentación de las páginas web. En las siguientes secciones, exploraremos en detalle qué es el DOM, cómo está representado, cómo acceder a él y cómo realizar modificaciones en tiempo real. A través de ejemplos prácticos, aprenderás a usar el DOM para crear interactividad dinámica en tus aplicaciones web.

4.1. ¿QUÉ ES EL DOM?

El DOM es una representación en memoria de la estructura de un documento HTML o XML. Es una interfaz de programación que proporciona un conjunto de objetos y métodos para acceder y manipular los elementos de un documento. Cada elemento HTML se convierte en un objeto en el DOM, lo que te permite acceder a sus propiedades y manipular su contenido y atributos.

Nodo Raíz



En el esquema anterior, cada rectángulo representa un nodo DOM y las flechas indican las relaciones entre nodos. Dentro de cada nodo, se ha incluido su tipo (que se verá más adelante) y su contenido.

4.1.1. LAS API DE NAVEGADOR

Los navegadores te ofrecen una serie de funcionalidades para interactuar con el código de las páginas web a través de JavaScript. Estas funcionalidades se conocen como API de navegador (browser APIs) o API de cliente (web APIs). No debes confundir las API de cliente con las API de servidor.

Existen muchas API de navegador, y están en constante evolución. Algunas de las más útiles son:

- **DOM.** Proporciona acceso a la estructura del documento (normalmente HTML), de manera que puedas leer y modificar desde JavaScript.
- **Web Storage.** Te proporciona herramientas para almacenar información en el navegador.
- **Geolocation.** Permite el acceso a la localización del usuario. Es muy útil, sobre todo cuando se carga la página desde un dispositivo móvil.
- **MediaStream.** Permite enviar y recibir audio y vídeo y trabajar con dispositivos como cámaras web o micrófonos.
- **Drag and Drop.** Proporciona la funcionalidad de arrastrar y soltar para interactuar con elementos y archivos en las páginas web.

4.1.2. ÁRBOL DE NODOS

En cualquier página HTML, el punto de partida del árbol de nodos es siempre un nodo especial llamado "Documento". Este nodo raíz actúa como el contenedor principal de todos los demás nodos en el documento.

Desde el nodo "Documento", la estructura se despliega de manera jerárquica. Cada etiqueta HTML se convierte en un nodo de tipo "Elemento". Los primeros nodos que se derivan directamente del nodo raíz son los nodos HEAD y BODY. Estos nodos representan las secciones principales de cualquier documento HTML.

Cada nodo "Elemento" puede contener otros nodos, que a su vez pueden ser nodos "Elemento" o nodos de tipo "Texto". Este proceso de conversión y anidamiento de etiquetas se realiza siguiendo una jerarquía clara y definida. Por ejemplo, dentro del nodo BODY, podrías tener nodos como div, p, h1, entre otros. Cada una de estas etiquetas HTML se transforma en su respectivo nodo "Elemento" en el árbol DOM.

La transformación de las etiquetas HTML no se detiene ahí. Además de generar un nodo "Elemento" para cada etiqueta, también se crea un nodo de tipo "Texto" que contiene el contenido textual dentro de esa etiqueta. Por ejemplo, la etiqueta HTML <p>Hola, mundo!</p> se convierte en un nodo "Elemento" para <p> y un nodo de tipo "Texto" que contiene "Hola, mundo!".

Este sistema de nodos permite una manipulación detallada y precisa del documento HTML. Puedes agregar, eliminar o modificar nodos "Elemento" y "Texto" para cambiar dinámicamente

la estructura y el contenido de la página web. Además, la relación jerárquica entre los nodos asegura que cualquier cambio en un nodo padre se refleje adecuadamente en sus nodos hijo, manteniendo la integridad del documento.

4.1.3. VEAMOS EL SIGUIENTE EJEMPLO

En este caso, el árbol de nodos se vería algo así:

- Nodo "Documento"
 - Nodo "Elemento" (BODY)
 - Nodo "Elemento" (div)
 - Nodo "Elemento" (p)
 - Nodo "Texto" ("Este es un párrafo con ")
 - Nodo "Elemento" (strong)
 - Nodo "Texto" ("texto en negrita")

```
<body>
  <div>
    <p>Este es un párrafo con <strong>texto en negrita</strong>.</p>
  </div>
</body>
```

Este ejemplo muestra cómo cada parte de la estructura HTML se convierte en nodos en el DOM, permitiéndote acceder y modificar cada elemento de manera granular. Esta capacidad de manipulación es fundamental para crear aplicaciones web interactivas y dinámicas.

4.2. ACCESO A LOS NODOS

Estos son los tipos de acceso más comunes para acceder a los nodos del DOM en JavaScript. Cada uno tiene su propósito y se elige según las necesidades del desarrollo. Es importante tener en cuenta que algunos métodos devuelven una lista de nodos, mientras que otros devuelven un solo nodo. Además, ten en cuenta que los métodos de acceso pueden variar dependiendo del tipo de nodo que estés seleccionando.

Recuerda que una vez que hayas accedido a los nodos, puedes utilizar otras propiedades y métodos del DOM para manipular su contenido, atributos y estructura. La comprensión de estos métodos de acceso es esencial para realizar operaciones dinámicas y efectivas en el DOM.

4.3. SELECCIONAR ELEMENTOS DEL DOM

4.3.1. ACCESO POR ID: GETELEMENTBYID()

El acceso por ID implica seleccionar un nodo específico a través de su atributo "id". Este método es rápido y eficiente, ya que los ID deben ser únicos en un documento. Puedes utilizar el método getElementById() para acceder a un nodo por su ID.

Ejemplo: Selección de un Elemento por ID:

```
<body>
  <div id="miElemento">
    <p>Contenido original</p>
  </div>

  <script>
    const miElemento = document.getElementById('miElemento');
    console.log('Elemento seleccionado:', miElemento);
  </script>
</body>
```

Ejemplo: Modificación de Contenido de un Elemento por ID:

```
<body>
  <div id="miElemento">
    <p>Contenido original</p>
  </div>

  <script>
    const miElemento = document.getElementById('miElemento');
    miElemento.textContent = 'Nuevo contenido';
  </script>
</body>
```

4.3.2. ACESO POR ETIQUETA: GETELEMENTSBYTAGNAME()

El acceso por etiqueta implica seleccionar todos los nodos que coincidan con una etiqueta HTML o XML específica. Puedes utilizar el método `getElementsByName()` para acceder a una lista de nodos que coincidan con la etiqueta especificada.

Ejemplo: Selección de Todos los Elementos de una Etiqueta:

```
<body>
<p>Primer párrafo</p>
<p>Segundo párrafo</p>
<p>Tercer párrafo</p>

<script>
  const parrafos = document.getElementsByTagName('p');
  console.log('Elementos <p> seleccionados:', parrafos);
</script>
</body>
```

Ejemplo: Modificación de Contenido de Todos los Elementos de una Etiqueta:

```
<body>
<p>Primer párrafo</p>
<p>Segundo párrafo</p>
<p>Tercer párrafo</p>

<script>
  const parrafos = document.getElementsByTagName('p');

  for (let i = 0; i < parrafos.length; i++) {
    parrafos[i].textContent = 'Nuevo contenido';
  }
</script>
```

4.3.3. ACESO POR CLASE: GETELEMENTSBYCLASNAME()

El acceso por clase implica seleccionar todos los nodos que tengan una determinada clase CSS. Puedes utilizar el método `getElementsByClassName()` para acceder a una lista de nodos que tengan la clase especificada.

Ejemplo : Selección de Todos los Elementos con una Clase:

```
<body>
  <p class="miClase">Primer párrafo</p>
  <p class="miClase">Segundo párrafo</p>
  <p class="miClase">Tercer párrafo</p>

  <script>
    const elementosConClase = document.getElementsByClassName('miClase');
    console.log('Elementos con la clase "miClase":', elementosConClase);
  </script>
</body>
```

Ejemplo: Modificación de Estilos de Todos los Elementos con una Clase:

```
<body>
  <p class="miClase">Primer párrafo</p>
  <p class="miClase">Segundo párrafo</p>
  <p class="miClase">Tercer párrafo</p>

  <script>
    const elementosConClase = document.getElementsByClassName('miClase');

    for (let i = 0; i < elementosConClase.length; i++) {
      elementosConClase[i].style.color = 'blue';
    }
  </script>
</body>
```

4.3.4. ACCESO POR SELECTOR CSS O ID: QUERYSELECTOR() / QUERYSELECTORALL()

El acceso por selector CSS permite seleccionar nodos utilizando selectores CSS, lo que brinda una gran flexibilidad. Puedes utilizar el método `querySelector()` para seleccionar el primer

nodo que coincida con el selector especificado, y querySelectorAll() para seleccionar todos los nodos que coincidan.

Selección por ID:

También puedes seleccionar un elemento por su ID utilizando #.

```
const elementoConId = document.querySelector('#miElemento');
console.log('Elemento con ID "miElemento" seleccionado:', elementoConId);
```

Selección de Elementos Anidados: querySelector() puede ser utilizado para seleccionar elementos anidados combinando selectores.

```
const enlaceEnLista = document.querySelector('ul li a');
console.log('Enlace dentro de una lista seleccionado:', enlaceEnLista);
```

Ejemplo : Selección del Primer Elemento de una Etiqueta:

```
<body>
  <p>Primer párrafo</p>
  <p>Segundo párrafo</p>
  <p>Tercer párrafo</p>

  <script>
    const primerParrafo = document.querySelector('p');
    console.log('Primer párrafo seleccionado:', primerParrafo);
  </script>
</body>
```

Ejemplo : Selección del Primer Elemento con una Clase:

```
<style>
  .miClase {
    color: red;
  }
</style>
</head>
<body>
  <p class="miClase">Primer párrafo</p>
  <p class="miClase">Segundo párrafo</p>
  <p class="miClase">Tercer párrafo</p>
```

```

<script>
  const primerElementoConClase = document.querySelector('.miClase');
  console.log('Primer elemento con la clase "miClase":', primerElementoConClase);
</script>
</body>

```

4.3.5. ACESO POR SELECTOR CSS O ID: QUERYSELECTORALL()

Es un método que se utiliza para seleccionar todos los elementos en el documento que coinciden con un selector CSS específico. Devuelve una NodeList, que es una colección de nodos (elementos) que cumplen con el selector proporcionado. (parecido a un array pero con diferentes propiedades).

Selector CSS Avanzado: Puedes utilizar selectores CSS avanzados para seleccionar elementos específicos, ya sea por clase, etiqueta, ID, atributo, etc.

Ejemplo de Uso

Supongamos que tienes el siguiente HTML:

```

<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <title>Ejemplo de querySelectorAll</title>
  <style>
    .destacado {
      color: blue;
    }
    .importante {
      font-weight: bold;
    }
  </style>
</head>
<body>
  <div class="contenedor">
    <p class="destacado">Este es un párrafo destacado.</p>
    <p class="importante">Este es un párrafo importante.</p>
    <p class="destacado importante">Este es un párrafo destacado e
importante.</p>
    <p>Este es un párrafo normal.</p>
  </div>
</body>

```

```
</div>
<script src="script.js"></script>
</body>
</html>
```

```
// script.js

// Selecciona todos los párrafos con la clase 'destacado'
var destacados = document.querySelectorAll('.destacado');
console.log(destacados); // Muestra la NodeList de elementos con la clase
'destacado'

// Cambia el color de texto de todos los elementos con la clase
'destacado'
destacados.forEach(function(elemento) {
    elemento.style.color = 'red';
});

// Selecciona todos los párrafos que tienen ambas clases 'destacado' e
'importante'
var destacadoImportante =
document.querySelectorAll('.destacado.importante');
console.log(destacadoImportante); // Muestra la NodeList de elementos con
ambas clases

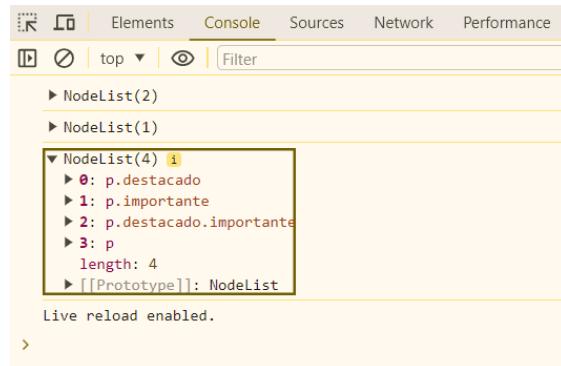
// Cambia el estilo de los elementos que tienen ambas clases
destacadoImportante.forEach(function(elemento) {
    elemento.style.backgroundColor = 'yellow';
});

// Selecciona todos los párrafos
var todosLosParrafos = document.querySelectorAll('p');
console.log(todosLosParrafos); // Muestra la NodeList de todos los
párrafos

// Agrega un borde a todos los párrafos
todosLosParrafos.forEach(function(parrafo) {
    parrafo.style.border = '1px solid black';
});
```

```
Este es un párrafo destacado.  
Este es un párrafo importante.  
Este es un párrafo destacado e importante.  
Este es un párrafo normal.
```

(resultado esperado en el body)



(resultado esperado por consola)

4.4. MODIFICAR TEXTOS EN EL DOM

4.4.1. LA PROPIEDAD .TEXTCONTENT

La propiedad `.textContent` nos devuelve el contenido de texto de un elemento HTML. Es útil para obtener (o modificar) sólo el texto dentro de un elemento, obviando el marcado o etiquetado HTML:

HTML

```
<div class="container">  
  <div class="parent">  
    <p>Hola a todos.</p>  
    <p class="message">Mi nombre es <strong>Alumno</strong>.</p>  
  </div>  
</div>
```

JS

```
const element = document.querySelector(".message");  
  
element.textContent;          // "Mi nombre es Alumno."  
element.textContent = "Hola a todos";
```

```
element.textContent;           // "Hola a todos"
```

4.4.2. LA PROPIEDAD INNERHTML

La propiedad innerHTML es una forma de acceder y manipular el contenido HTML de un elemento desde JavaScript. Permite tanto obtener el contenido HTML actual como establecer un nuevo contenido.

Advertencia de Seguridad:

Al utilizar **innerHTML** para establecer contenido, es importante tener en cuenta la seguridad. Si el contenido proviene del usuario, podría haber riesgos de inyección de código. En estos casos, es preferible usar métodos más seguros como **textContent** o **createElement**.

Ejemplos de Uso:

Ejemplo 1: Obtener y Mostrar Contenido HTML:

```
<body>
  <div id="miElemento">
    <p>Contenido original</p>
  </div>

  <script>
    const miElemento = document.getElementById('miElemento');
    const contenidoHTML = miElemento.innerHTML;
    console.log('Contenido HTML actual:', contenidoHTML);
  </script>
</body>
```

Ejemplo 2: Establecer nuevo contenido HTML:

```
<body>
  <div id="miElemento">
```

```

<p>Contenido original</p>
</div>

<script>

const miElemento = document.getElementById('miElemento');
miElemento.innerHTML = '<strong>Nuevo contenido HTML</strong>';
</script>
</body>

```

4.4.3. LA PROPIEDAD .OUTERHTML

La propiedad **outerHTML** en JavaScript es similar a **innerHTML**, pero a diferencia de esta última, **outerHTML** incluye el propio elemento en el que se encuentra, no solo su contenido. **outerHTML** devuelve una cadena de texto que representa el elemento y todo su contenido HTML.

La propiedad **outerHTML** es útil cuando necesitas manipular un elemento en su totalidad, incluyendo su propia etiqueta. Al igual que con **innerHTML**, es importante considerar la seguridad y asegurarse de que el contenido que se asigna sea confiable y esté libre de posibles vulnerabilidades.

Ejemplo 1: Obtener y Mostrar Contenido HTML con el Propio Elemento:

```

<body>
  <div id="miElemento">
    <p>Contenido original</p>
  </div>

  <script>
    const miElemento = document.getElementById('miElemento');
    const contenidoConElemento = miElemento.outerHTML;
    console.log('Contenido HTML con el propio elemento:', contenidoConElemento);
  </script>
</body>

```

Ejemplo 2: Reemplazar el Contenido Completo de un Elemento:

```

<body>
  <div id="miElemento">
    <p>Contenido original</p>
  </div>

  <script>

```

```
const miElemento = document.getElementById('miElemento');
miElemento.outerHTML = '<div id="nuevoElemento"><p>Nuevo contenido
HTML</p></div>';
</script>
</body>
```

4.5. CREACIÓN DE ELEMENTOS EN EL DOM

4.5.1. CREACIÓN DE UN NUEVO ELEMENTO:

- Utilizamos el método **createElement()** para crear un nuevo elemento en el DOM.
- Ejemplo:

```
const nuevoParrafo = document.createElement('p');
```

4.5.2. ASIGNACIÓN DE CONTENIDO AL NUEVO ELEMENTO:

- Puedes asignar contenido al nuevo elemento utilizando **textContent**, **innerHTML**, u otros métodos según tus necesidades.
- Ejemplo:

```
nuevoParrafo.textContent = 'Este es un nuevo párrafo';
```

4.5.3. AÑADIR ATRIBUTOS AL NUEVO ELEMENTO:

- Puedes añadir atributos al nuevo elemento utilizando el método **setAttribute()**.
- Ejemplo:

```
nuevoParrafo.setAttribute('class', 'miClase');
```

4.5.4. ADJUNTAR EL NUEVO ELEMENTO AL DOM:

- Utilizamos métodos como `appendChild()` o `insertBefore()` para adjuntar el nuevo elemento a un elemento existente en el DOM.
- Ejemplo:

```
const contenedor = document.getElementById('contenedor');
contenedor.appendChild(nuevoParrafo);
```

En el siguiente ejemplo podemos ver un código donde se agrega un nuevo párrafo

```
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <title>Ejemplo de querySelectorAll</title>
    <style>
        .destacado {
            color: blue;
        }
        .importante {
            font-weight: bold;
        }
    </style>
</head>
<body>
    <div id="contenedor">
        <!-- Aquí se añadirán nuevos elementos -->
    </div>

    <script>
        // Crear un nuevo párrafo
        const nuevoParrafo = document.createElement('p');

        // Asignar contenido al nuevo párrafo
        nuevoParrafo.textContent = 'Este es un nuevo párrafo';

        // Añadir atributo de clase al nuevo párrafo
        nuevoParrafo.setAttribute('class', 'miClase');

        // Obtener el contenedor en el que se añadirá el nuevo párrafo
        const contenedor = document.getElementById('contenedor');
    </script>

```

```
// Adjuntar el nuevo párrafo al contenedor  
contenedor.appendChild(nuevoParrafo);  
</script>  
</body>  
</html>
```

4.6. EVENTOS

El DOM no solo te ofrece un mecanismo para leer y modificar el documento HTML, sino que también te permite reaccionar a las acciones que ocurren en la página, como las interacciones de usuario. Esta funcionalidad es conocida como **gestión de eventos**, y es una de las características fundamentales en la programación del lado del cliente.

Con JavaScript, puedes escribir código que se ejecute en respuesta a eventos específicos. Hay un conjunto de eventos estándar que puedes detectar, como los eventos de ratón o de teclado. Para responder a estos eventos, debes asociarles una serie de funciones y objetos encargados de su gestión y tratamiento, conocidos como **manejadores de eventos**.

El mecanismo de gestión de eventos también define cómo se propagan estos eventos a través de la jerarquía de elementos del DOM y establece ciertas acciones por defecto que se ejecutan automáticamente cuando se producen determinados eventos. Es crucial que te familiarices con estas características para diseñar correctamente tus aplicaciones.

4.6.1. TIPOS DE EVENTOS

Los eventos de JavaScript son acciones que ocurren sobre determinados elementos. Algunas de estas acciones son causadas por el usuario, como tú, mientras que otras pueden ser desencadenadas por distintos motivos, como la carga de una página, la recepción completa de un archivo o la finalización de un temporizador. Estos eventos están detallados en la especificación del DOM.

IMPORTANTE

El código JavaScript se ejecuta en la página HTML una vez que se carga la etiqueta `<script>` correspondiente. Si necesitas ejecutar un programa que haga referencia a algún elemento del DOM, es vital esperar a que este se haya cargado. Por lo tanto, los scripts definidos dentro de `<head>` no pueden hacer referencia a ningún nodo que dependa de `document.body`, ya que `<body>` todavía no se habrá cargado. Para evitar este problema, tienes dos opciones:

- Colocar la etiqueta `<script>` con el código del programa al final del elemento `<body>`.

- Incluir el programa principal dentro de la función que maneja el evento DOMContentLoaded. De esta manera, el programa se ejecutará cuando el documento esté completamente cargado.

4.6.2. EVENTOS DE RATÓN (MOUSE EVENTS)

Los eventos de ratón son aquellos que se producen cuando interactúas con el ratón sobre los elementos de la página. Algunos de los eventos de ratón más comunes son:

- **click**: Se produce cuando haces clic en un elemento.
- **mousedown**: Se activa cuando presionas un botón del ratón sobre un elemento.
- **mouseup**: Ocurre cuando sueltas el botón del ratón sobre un elemento.
- **dblclick**: Se produce cuando haces doble clic en un elemento.
- **mouseenter**: Se activa cuando el puntero del ratón entra en el área de un elemento.
- **mouseleave**: Ocurre cuando el puntero del ratón sale del área de un elemento.
- **mousemove**: Se activa cada vez que mueves el ratón sobre un elemento.

4.6.3. EVENTOS DE FOCO (FOCUS EVENTS)

Los eventos de foco se producen cuando un elemento recibe o pierde el foco. Estos eventos son especialmente útiles en formularios y en la gestión de la interacción del usuario con los elementos de entrada de datos. Algunos de los eventos de foco más comunes son:

- **focus**: Se activa cuando un elemento recibe el foco.
- **blur**: Ocurre cuando un elemento pierde el foco.

4.6.4. EVENTOS DE TECLADO (KEYBOARD EVENTS)

Los eventos de teclado se producen cuando interactúas con el teclado. Estos eventos son esenciales para capturar la entrada del usuario a través del teclado. Algunos de los eventos de teclado más comunes son:

- **keydown**: Se activa cuando se presiona una tecla.
- **keyup**: Ocurre cuando se suelta una tecla.
- **keypress**: Se activa cuando se pulsa una tecla que produce un carácter.

4.6.5. EVENTOS DE FORMULARIO (FORM EVENTS)

Los eventos de formulario se producen en los elementos de formulario, como entradas, botones, y el propio formulario. Estos eventos son cruciales para manejar la validación y el envío de formularios. Algunos de los eventos de formulario más comunes son:

- **submit**: Se activa cuando se envía un formulario.
- **reset**: Ocurre cuando se restablece un formulario.
- **change**: Se produce cuando cambias el valor de un elemento de formulario.
- **input**: Se activa cada vez que el valor de un elemento de entrada cambia.

4.6.6. EVENTOS DE DOCUMENTO (DOCUMENT EVENTS)

Los eventos de documento se producen a nivel del documento HTML completo. Estos eventos son útiles para manejar el estado de carga de la página y otras interacciones globales. Algunos de los eventos de documento más comunes son:

- **DOMContentLoaded**: Se activa cuando el documento HTML ha sido completamente cargado y analizado, sin esperar a que se carguen las hojas de estilo, imágenes y subframes.
- **load**: Ocurre cuando la página completa, incluidas todas las dependencias como estilos e imágenes, ha sido completamente cargada.
- **unload**: Se produce cuando la página se está descargando, por ejemplo, cuando navegas fuera de la página.
- **resize**: Se activa cuando cambias el tamaño de la ventana del navegador.

4.7. MANEJADORES DE EVENTOS

Mediante JavaScript, es posible detectar dichos eventos y ejecutar funciones en respuesta a ellos. A estas funciones se las conoce como manejadores de eventos (event handlers, en inglés).

La primera opción para escribir un manejador de eventos es escribirlo directamente sobre el elemento **HTML**. Para ello, se utilizan los aPágina onEVENTO donde EVENTO es el evento que se quiera capturar. Por ejemplo:

```
<body>
  <button id="boton" onclick="saludar ()">Pulsa el botón</button>
```

```
<script>
function saludar () {
    alert("Hola Mundo!");
}
</script>

</body>
```

La segunda opción consiste en hacerlo desde JavaScript:

```
<body>

    <button id="boton">Pulsa el botón</button>
    <script>
        function saludar () {
            alert("Hola Mundo!");
        }
        document.getElementById("boton").onclick= saludar;
    </script>

</body>
```

IMPORTANTE

Como puede verse en los anteriores ejemplos:

- En HTML hay que indicar el código que queramos ejecutar realizando la llamada a la función (esto es, incluyendo los paréntesis).
- En JavaScript hay que indicar el nombre de la función (esto es, **sin incluir los paréntesis**).

La última opción es la más recomendable, ya que, a diferencia de las anteriores, permite asociar más de un manejador de eventos a un elemento. Consiste en utilizar la función addEventListener. A continuación, se muestra un ejemplo de uso:

```
<body>
    <button id="boton">Pulsa el botón</button>
    <script>
        function saludar1() {
            alert("Hola Mundo!");
        }
    </script>
```

```

        }
        function saludar2() {
            alert("Hola otra vez!");
        }
        let boton = document.getElementById("boton");
        boton.addEventListener("click", saludar1);
        // Se puede añadir un segundo manejador de eventos, que se
        // ejecutará tras el anterior
        boton.addEventListener("click", saludar2);
    </script>

</body>

```

Esta opción permite también utilizar la función removeEventListener para eliminar un manejador de eventos:

```

<body>

<button id="boton">Pulsa el botón</button>
<script>
    function saludar1() {
        alert("Hola Mundo!");
    }
    let boton = document.getElementById("boton");
    // Añadir manejador de eventos
    boton.addEventListener("click", saludar1);
    // Código...
    // Eliminar manejador de eventos
    // Hay que indicar de nuevo el evento y la función asociada
    boton.removeEventListener("click", saludar1);
</script>

</body>

```

4.7.1. APRETAR UNA TECLA

En este ejemplo, cuando se pulse la tecla Enter, se mostrará una alerta con el texto Me has pulsado enter.

Es importante destacar la condicional `evento.code === "Enter"`. En caso de querer actuar sobre otra tecla, deberás buscar cual es el código de la tecla. Puedes jugar imprimiendo `evento.code` o buscar la tecla en https://developer.mozilla.org/en-US/docs/Web/API/UI_Events/Keyboard_event_code_values

```
<script>

// Funciones
function instrucciones(evento) {
    // Filtra por la tecla Enter
    if(evento.code === "Enter") {
        alert("Me has pulsado enter");
    }
}

// Eventos
document.body.addEventListener("keydown", instrucciones);

</script>
```

4.7.2. SCROLL

Ejemplo donde se imprime por consola la posición del `scroll` cada vez que se haga scroll.

```
document.addEventListener("scroll", function (evento) {
    const ultimaPosicion = window.scrollY;
    console.log(ultimaPosicion)
});
```

4.7.3. OBJETO DE EVENTO (EVENT)

Este objeto en JavaScript proporciona información sobre un evento que ha ocurrido en la interfaz de usuario. En el caso específico por ejemplo del evento **mousemove**, este objeto tiene diversas propiedades que te permiten obtener información sobre el movimiento del ratón.

4.7.4. EJEMPLO DE UNA PROPIEDAD DEL OBJETO EVENTO (EVENT) EN MOUSEMOVE: CLIENTX Y CLIENTY:

- **Descripción:** Representan las coordenadas horizontales (X) y verticales (Y) del puntero del ratón en relación con la esquina superior izquierda del área de contenido del navegador (ventana gráfica).

```
var mouseX = event.clientX;
var mouseY = event.clientY;
console.log('Coordenadas del ratón: X=' + mouseX + ', Y=' +
mouseY);
```

Ejemplo de utilización para capturar las coordenadas X e Y del ratón

```
let guardo_X;
let guardo_Y;
document.addEventListener('mousemove', function(event) {
    // Obtener las coordenadas X e Y del evento del ratón
    guardo_X = event.clientX;
    guardo_Y = event.clientY;
});
```

4.7.5. PRACTICAMOS CON EVENTOS

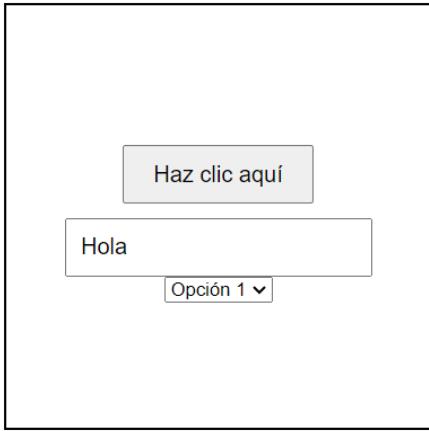
En esta práctica guiada, veremos diferentes tipos de eventos mediante ejemplos prácticos. Cubriremos eventos de clic, mouseover, teclado y cambio en un elemento de selección.

Paso 1: Preparar el Entorno

Primero, crea un archivo HTML básico. Este archivo servirá como base para nuestra práctica.

```
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Práctica de Eventos en JavaScript</title>
    <style>
        #contenedor {
            width: 300px;
            height: 300px;
            border: 2px solid black;
            display: flex;
            justify-content: center;
            align-items: center;
            flex-direction: column;
        }
        #miBoton {
            padding: 10px 20px;
            font-size: 16px;
            margin-top: 10px;
        }
        #miInput {
            padding: 10px;
            font-size: 16px;
            margin-top: 10px;
        }
    </style>
</head>
<body>
    <div id="contenedor">
        <button id="miBoton">Haz clic aquí</button>
        <input type="text" id="miInput" placeholder="Escribe algo...">
        <select id="miSelect">
            <option value="opcion1">Opción 1</option>
            <option value="opcion2">Opción 2</option>
        </select>
    </div>

    <script src="script.js"></script>
</body>
</html>
```



(resultado esperado)

Paso 2: Asociar Manejadores de Eventos

Vamos a asociar manejadores de eventos a nuestro botón, contenedor, campo de texto (input) y un menú desplegable (select).

```
// Seleccionamos los elementos del DOM
const contenedor = document.getElementById('contenedor');
const miBoton = document.getElementById('miBoton');
const miInput = document.getElementById('miInput');
const miSelect = document.getElementById('miSelect');

// Evento de clic en el botón
miBoton.addEventListener('click', function() {
  alert('¡Botón clicado!');
});

// Evento de mouseover en el contenedor
contenedor.addEventListener('mouseover', function() {
  contenedor.style.backgroundColor = 'lightblue';
});

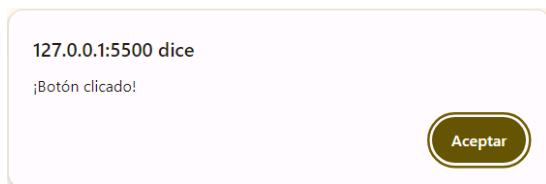
// Evento demouseout en el contenedor
contenedor.addEventListener('mouseout', function() {
  contenedor.style.backgroundColor = 'white';
});

// Evento de keyup en el campo de texto
miInput.addEventListener('keyup', function(event) {
  console.log('Tecla presionada:', event.key);
});
```

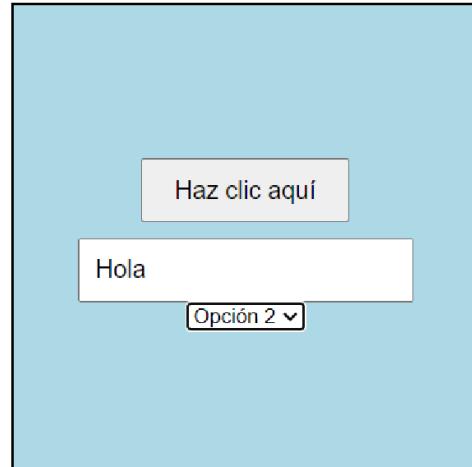
```
// Evento de change en el menú desplegable  
miSelect.addEventListener('change', function() {  
  alert('Opción seleccionada: ' + miSelect.value);  
});
```

Probar y Observar Resultados

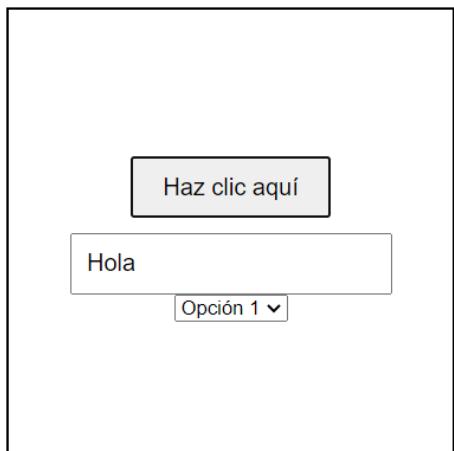
Guarda los cambios y abre tu archivo HTML en un navegador. Observa cómo se activan los diferentes manejadores de eventos al interactuar con los elementos:



1. **Clic en el botón:** Aparecerá una alerta diciendo "¡Botón clicado!".



Mouseover y mouseout en el contenedor:
El color de fondo del contenedor cambiará cuando el cursor pase por encima y volverá a su color original cuando salga.



127.0.0.1:5500 dice
Opción seleccionada: opcion2

Aceptar

Cambio en el menú desplegable:
Aparecerá una alerta mostrando la opción seleccionada.

```
DevTools is now available in Spanish! Always match Chrome's language
Elements Console Sources Network
top ▾ Filter
Tecla presionada: H
Tecla presionada: Shift
Tecla presionada: o
Tecla presionada: l
Tecla presionada: a
```

Teclado en el campo de texto: En la consola del navegador, verás las teclas que presionas mientras escribes en el campo de texto.

4.8. FORMULARIOS EN JS

El manejo de formularios es una parte fundamental en el desarrollo web, ya que permite a los usuarios ingresar y enviar datos que pueden ser procesados por tu aplicación. En este capítulo, aprenderás cómo interactuar con formularios utilizando JavaScript.

Los formularios son una vía de entrada a la base de datos y, por ende, a la zona más sensible de la aplicación web. La información que recibiremos debe pasar un exhaustivo control de calidad: La edad será un número, no una dirección de correo; una foto no debe ser un archivo comprimido zip. Y así podríamos continuar con una larga lista de especificaciones que no perdonaremos al visitante ni en un solo acento.

4.8.1. EXPERIENCIA DE USUARIO

No pretendemos castigar, sino orientar. Con mensajes claros y bien explicados, te indicaremos la forma correcta de completar cada campo siguiendo el formato adecuado. Hasta que se

cumplan todos los requisitos, ignoraremos cualquier información enviada a la base de datos. Incluso, evitaremos desde el frontend que se pueda presionar el botón de envío. Al final, somos los responsables de todo lo que pueda ocurrir.

La validación desde el frontend (JavaScript) no garantiza que la información llegue al servidor en el formato correcto. Siempre es necesario realizar comprobaciones desde el backend y devolver un código 400 si los datos no son válidos.

A continuación, exploraremos cómo construir un formulario básico y los atributos clave que puedes utilizar para mejorar la funcionalidad y accesibilidad de tus formularios.

4.8.2. ATRIBUTOS COMUNES DE LOS FORMULARIOS

- **action:** Define la URL a la que se enviarán los datos del formulario.
- **method:** Especifica el método de envío de datos (GET o POST).
- **id:** Un identificador único para el formulario.
- **name:** Un nombre para el formulario, que puede ser útil en scripts.
- **autocomplete:** Indica si el navegador debe completar automáticamente los valores del formulario (on o off).

4.8.3. ATRIBUTOS DE LOS ELEMENTOS DE ENTRADA

- **type:** Define el tipo de entrada (text, email, password, submit, etc.).
- **name:** El nombre del campo, que se usa para identificar los datos cuando se envían.
- **id:** Un identificador único para el campo, útil para asociar etiquetas y referenciar en scripts.
- **value:** El valor predeterminado del campo.
- **placeholder:** Un texto de sugerencia que aparece en el campo cuando está vacío.
- **required:** Indica que el campo debe ser completado antes de enviar el formulario.
- **pattern:** Especifica una expresión regular que el valor del campo debe coincidir.
- **min y max:** Definen los valores mínimo y máximo para los campos numéricos.
- **maxlength:** Establece la longitud máxima del texto que se puede ingresar.

4.8.4. ESTRUCTURA BÁSICA DE UN FORMULARIO

```
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Formulario de Contacto</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            background-color: #f4f4f4;
            margin: 0;
            padding: 0;
            display: flex;
            justify-content: center;
            align-items: center;
            height: 100vh;
        }

        form {
            max-width: 500px;
            width: 100%;
            background: #fff;
            padding: 2em;
            border: 1px solid #ccc;
            border-radius: 1em;
            box-shadow: 0 2px 5px rgba(0, 0, 0, 0.1);
        }

        div + div {
            margin-top: 1em;
        }

        label {
            display: block;
            margin-bottom: 0.5em;
            font-weight: bold;
        }

        input, textarea {
            width: 100%;
            padding: 0.5em;
            margin-top: 0.5em;
            border: 1px solid #ccc;
        }
    </style>

```

```

        border-radius: 0.25em;
        box-sizing: border-box;
    }

    button {
        width: 100%;
        padding: 0.75em;
        background-color: #009DE0;
        color: #fff;
        border: none;
        border-radius: 0.25em;
        font-size: 1em;
        cursor: pointer;
        transition: background-color 0.3s ease;
    }

    button:hover {
        background-color: #007bb5;
    }

```

</style>

</head>

<body>

<form id="formularioContacto">

<div>

<label for="nombre">Nombre:</label>

<input type="text" id="nombre" name="nombre" required>

</div>

<div>

<label for="email">Correo Electrónico:</label>

<input type="email" id="email" name="email" required>

</div>

<div>

<label for="mensaje">Mensaje:</label>

<textarea id="mensaje" name="mensaje" required></textarea>

</div>

<div>

<button type="submit">Enviar</button>

</div>

</form>

<script src="formScript.js"></script>

</body>

</html>

Nombre:

Correo Electrónico:

Mensaje:

Enviar

(Resultado esperado)

4.8.5. CAPTURAR EVENTOS DE ENVÍO DE FORMULARIO

Vamos a capturar el evento de envío del formulario para poder procesar los datos ingresados antes de enviarlos.

```
// Seleccionamos el formulario
const formulario = document.getElementById('formularioContacto');

// Manejador del evento submit
formulario.addEventListener('submit', function(event) {
    // Prevenir el comportamiento por defecto del formulario
    event.preventDefault();

    // Obtener los valores de los campos del formulario
    const nombre = document.getElementById('nombre').value;
    const email = document.getElementById('email').value;
    const mensaje = document.getElementById('mensaje').value;

    // Mostrar los valores en la consola
    console.log('Nombre:', nombre);
    console.log('Correo Electrónico:', email);
    console.log('Mensaje:', mensaje);

    // Aquí podrías agregar lógica para enviar los datos a un servidor
});
```

```
Live reload enabled.  
Nombre: Juan Luis  
Correo Electrónico: juanluis@digitechfp.com  
Mensaje: Hola Juan Luis desde el Form.  
>
```

(Resultado esperado: Datos mostrados por consola)

4.8.6. VALIDACIÓN DE FORMULARIOS

Es importante validar los datos del formulario antes de procesarlos. Vamos a agregar validaciones personalizadas usando JavaScript.

```
formulario.addEventListener('submit', function(event) {  
    event.preventDefault();  
  
    const nombre = document.getElementById('nombre').value;  
    const email = document.getElementById('email').value;  
    const mensaje = document.getElementById('mensaje').value;  
  
    // Validar que todos los campos estén llenos  
    if (nombre === '' || email === '' || mensaje === '') {  
        alert('Todos los campos son obligatorios.');//  
        return;  
    }  
  
    // Validar el formato del correo electrónico  
    const emailRegex = /^[^@\s]+@[^\s@]+\.[^\s@]+$/;  
    if (!emailRegex.test(email)) {  
        alert('Por favor, introduce un correo electrónico válido.');//  
        return;  
    }  
  
    // Si todo es válido, procesar el formulario  
    alert('Formulario enviado con éxito.');//  
    console.log('Nombre:', nombre);  
    console.log('Correo Electrónico:', email);  
    console.log('Mensaje:', mensaje);  
  
    // Aquí podrías agregar lógica para enviar los datos a un servidor  
});
```

127.0.0.1:5500 dice
Formulario enviado con éxito.

Aceptar

(Alert disparada si el formulario esta correctamente completado)

Nombre:
Juan perez

Correo Electrónico:
alumno@dsd

Mensaje:
Hola Alumnos

Enviar

127.0.0.1:5500 dice

Por favor, introduce un correo electrónico válido.

Aceptar

(Vease que en este caso el campo email no pasa la expresion regular por lo tanto se ejecuta el Alert.)

4.9. EXPRESIONES REGULARES

Las expresiones regulares, conocidas como RegExp o RegEx, son herramientas extremadamente útiles y versátiles para buscar, capturar y reemplazar texto mediante patrones definidos. Estos patrones ofrecen una manera abstracta y eficiente de realizar búsquedas de texto que, de otro modo, podrían ser complejas o incluso inviables.

4.9.1. COMPRENDIENDO LOS PATRONES DE EXPRESIONES REGULARES

Las expresiones regulares se construyen con cadenas de texto en las que ciertos símbolos tienen significados especiales. Antes de sumergirnos en estos símbolos, hagamos un ejercicio práctico para entender mejor su funcionalidad y aplicación.

4.9.2. CUÁNDO EVITAR EL USO DE EXPRESIONES REGULARES

Aunque las expresiones regulares pueden parecer una solución elegante para muchos problemas, su uso conlleva ciertos riesgos. La simplicidad aparente de resolver problemas con menos líneas de código no siempre se traduce en soluciones más claras o eficientes. Es crucial ser consciente de que los patrones mal diseñados pueden permitir coincidencias no deseadas.

Además, un uso excesivo de expresiones regulares puede llevar a un código difícil de entender y mantener. Esto es especialmente importante para ti, como estudiante, ya que la claridad y simplicidad del código son fundamentales para tu aprendizaje y comprensión. En muchos casos, optar por soluciones más claras y directas puede ser más beneficioso que utilizar una expresión regular compleja.

4.9.3. PARTES FUNDAMENTALES DE UNA EXPRESIÓN REGULAR:

Las expresiones regulares consisten en varias partes, cada una con un propósito específico:

- **Patrón:** La secuencia de caracteres que define la búsqueda o manipulación de texto.
- **Modificadores:** Opciones que afectan el comportamiento de la expresión regular, como `i` para hacer la búsqueda insensible a mayúsculas y minúsculas.
- **Metacaracteres:** Caracteres especiales con significados específicos, como `^` para indicar el inicio de una cadena y `$` para el final.

4.9.4. EXPRESIONES REGULARES BÁSICAS

Crear patrones de expresiones regulares es todo un arte, y requiere cierta soltura y destreza. Aquí tienes algunos patrones básicos de las expresiones regulares que te pueden ayudar a empezar:

Expresión regular	Carácter especial	Significado	Descripción
.	Punto	Comodín	Cualquier carácter (o texto de tamaño 1)
A B	Pipe	Opciones lógicas	Opciones alternativas (o A o B)
C(A B)	Paréntesis	Agrupaciones	Agrupaciones alternativas (o CA o CB)
[0-9]	Corchetes	Rangos de caracteres	Un dígito (del 0 al 9)
[A-Z]			Una letra mayúscula de la A a la Z
[^A-Z]	^ en corchetes	Rango de exclusión	Una letra que no sea mayúscula de la A a la Z
[0-9]*	Asterisco	Cierre o clausura	Un dígito repetido 0 ó más veces (vacío incluido)
[0-9]+	Signo más	Cierre positivo	Un dígito repetido 1 ó más veces
[0-9]{3}	Llaves	Coincidencia exacta	Cifra de 3 dígitos (dígito repetido 3 veces)
[0-9]{2,4}		Coincidencia (rango)	Cifra de 2 a 4 dígitos (rep. de 2 a 4 veces)
b?	Interrogación	Carácter opcional	El carácter b puede aparecer o puede que no
\.	Barra invertida	Escape	El carácter . literalmente (no como comodín)

4.9.5. ¿CÓMO CREAR UNA EXPRESIÓN REGULAR EN JAVASCRIPT Y HTML?

JavaScript:

En JavaScript, puedes crear una expresión regular utilizando el constructor RegExp o mediante la notación literal. Aquí tienes ejemplos de ambas formas:

```
// Usando el constructor RegExp  
var expresionRegular1 = new RegExp('patron');  
  
// Usando notación literal  
var expresionRegular2 = /patron/;
```

HTML:

En HTML, las expresiones regulares se emplean principalmente en los atributos pattern de los elementos de formulario para validar la entrada del usuario. Por ejemplo:

```
<input type="text" pattern="[a-zA-Z0-9]+" title="Solo se permiten  
letras y números">
```

En este caso, la expresión regular **[a-zA-Z0-9]+** especifica que el campo solo acepta letras y números.

4.9.6. PRACTICA GUIADA UTILIZACIÓN DE EXPRESIONES REGULARES EN FORMULARIOS CON HTML.

Ejemplo 1 de expresiones regulares

Tipo de campo: Nombre de usuario

Campo obligatorio: required

Entre 5-40 caracteres: minlength="5" y maxlength="40"

Sólo se permiten letras (mayúsculas y minúsculas) y números: pattern="[A-Za-z0-9]+"

```
<form method="post" action="/send/">  
  <input type="text" name="username" placeholder="Nombre"  
        minlength="5" maxlength="40" required pattern="[A-Za-z0-9]+">  
</form>
```

Nótese que de no incluir los atributos minlength y maxlength el usuario no tendría limitación en cuanto al tamaño. Esto también puede incorporarse en la propia expresión regular, y prescindir de dichos atributos:

```
<form method="post" action="/send/">
  <input type="text" name="username" placeholder="usuario"
         required pattern="[A-Za-z0-9]{5,40}">
</form>
```

Ten en cuenta que al utilizar atributos básicos como maxlen o max, el texto del campo simplemente no pasa la validación de forma silenciosa. Sin embargo, con el atributo pattern, el navegador muestra un mensaje de advertencia en caso de no pasar la validación indicada en el mismo.

4.9.7. PRACTICA GUIADA, EXPRESIÓN REGULAR PARA MODELO DE COCHE

En el siguiente caso, se pide al usuario que indique el modelo de coche que posee, en un posible formulario de servicio técnico. Los modelos posibles son A1, A3, A4 y A15. En lugar de mostrar una lista de selección, podemos mostrar un campo de texto y colocar una validación como la siguiente:

Tipo de campo: Modelo de coche

Campo obligatorio: required.

Sólo se permiten las opciones: A1, A3, A4 y A15

Nótese que la expresión regular permite tanto el formato a1 como el formato A1. Por otro lado, ¿Para qué sirve aquí el atributo title? Te lo explico en el siguiente apartado.

```
<form method="post" action="/send/">
  <input type="text" name="model" placeholder="Su modelo de coche"
         required pattern="(A|a)(1|3|4|15)"
         title="Modelos posibles: A1, A3, A4 y A15">
</form>
```

4.9.8. HERRAMIENTAS PARA REGEXP

A continuación, algunas herramientas útiles para crear o probar expresiones regulares:

Herramienta	Descripción
RegExR	Herramienta para construir y testear expresiones regulares.
Debuggex	Herramienta de creación gráfica de expresiones regulares.
RegEx101	Herramienta para construir y testear expresiones regulares.
RegExBuddy	Herramienta profesional para el trabajo con expresiones regulares.
RegExplained	Herramienta visual para explicar el funcionamiento de una expresión regular.
RegExper	Herramienta para generar diagramas visuales de expresiones regulares.
Rubular	Editor de expresiones regulares.

4.10. MÉTODOS PARA EXPRESIONES REGULARES:

4.10.1. TEST(CADENA):

Devuelve **true** si la expresión regular tiene al menos una coincidencia en la cadena, y **false** si no hay coincidencias.

```
var expresionRegular = /patron/;  
console.log(expresionRegular.test('cadena de texto'));
```

4.10.2. EXEC(CADENA):

Devuelve un array con información sobre la coincidencia o **null** si no hay coincidencia. El array contiene la cadena coincidente y la información sobre grupos capturados.

```
var expresionRegular = /patron/;  
console.log(expresionRegular.exec('cadena de texto'));
```

4.10.3. MATCH(EXPRESIÓN):

Devuelve un array con todas las coincidencias encontradas en una cadena o **null** si no hay coincidencias.

```
var expresionRegular = /patron/g;
console.log('cadena de texto'.match(expresionRegular)); // Devuelve
[ 'patron' ]
```

4.10.4. REPLACE(EXPRESION, REEMPLAZO):

Reemplaza las coincidencias encontradas en una cadena con el valor especificado.

```
var expresionRegular = /patron/g;
var nuevaCadena = 'cadena de texto'.replace(expresionRegular,
'nuevo');
console.log(nuevaCadena); // Devuelve 'cadena de texto' -> 'cadena
de texto' fue reemplazado por 'nuevo'
```

4.10.5. SEARCH(EXPRESION):

Devuelve la posición de la primera coincidencia encontrada en una cadena o -1 si no hay coincidencias.

```
var expresionRegular = /patron/;
console.log('cadena de texto'.search(expresionRegular)); // Devuelve
la posición o -1
```

4.11. EJERCICIOS

Objetivos: Estos ejercicios están diseñados para que practiques y domines la manipulación del DOM (Document Object Model) utilizando JavaScript. Aprenderás a seleccionar y modificar elementos, manejar eventos, crear y adjuntar nuevos elementos, capturar eventos de teclado y ratón, validar formularios con expresiones regulares, y manipular clases CSS. También trabajarás con eventos de desplazamiento, clonación y eliminación de elementos del DOM, lo cual es fundamental para crear aplicaciones web interactivas y dinámicas.

Duración: 3.5hs.

Entrega: Para la entrega de los ejercicios de la Unidad 4, debes crear una carpeta llamada Unidad4_Ejercicios que contenga un archivo HTML (index.html) y un archivo JavaScript (scripts.js) en los cuales hayas implementado y comentado todas las soluciones de los ejercicios propuestos. Asegúrate de que el archivo HTML esté correctamente enlazado al archivo JavaScript y que todo el código esté claramente organizado y documentado. Una vez que hayas terminado, comprime la carpeta Unidad4_Ejercicios en un archivo ZIP y súbelo a la plataforma.

1. Acceso a Elementos del DOM:

- Selecciona un elemento por su ID y cambia su contenido de texto a "Hola, Mundo".

2. Modificar Atributos de Elementos:

- Crea un botón que, al hacer clic, cambie el color de fondo de un párrafo a azul.

3. Crear y Adjuntar Nuevos Elementos:

- Crea un script que genere un nuevo elemento con el texto "Nuevo ítem" y lo añada a una lista desordenada ().

4. Manejo de Eventos del Ratón:

- Diseña un script que detecte cuando el ratón entra y sale de un div, cambiando su color de fondo a rojo cuando entra y a verde cuando sale.

5. Capturar Eventos de Teclado:

- Escribe un script que muestre en un párrafo (<p>) el código de la tecla presionada por el usuario.

6. Validación de Formularios con Expresiones Regulares:

- Crea un formulario con un campo de entrada para el correo electrónico y valida que el formato sea correcto utilizando una expresión regular.

7. Manipulación de Clases CSS:

- Escribe un script que, al hacer clic en un botón, añada o quite una clase CSS a un párrafo, alternando su color de texto entre rojo y negro.

8. Evento de Scroll:

- Crea un script que detecte el evento de desplazamiento (scroll) en la ventana y muestre la posición actual del scroll en un div.

9. Clonación de Elementos del DOM:

- Escribe un script que clone un elemento existente en el DOM y lo añada a la página.

10. Remover Elementos del DOM:

- Crea un script que elimine un elemento específico del DOM cuando se haga clic en un botón.

4.12. RESUMEN DE LA UNIDAD 4

En esta unidad, hemos explorado en profundidad el manejo del DOM (Document Object Model) y BOM (Browser Object Model) utilizando JavaScript. El DOM es una representación en memoria de la estructura de un documento HTML o XML, permitiendo a los desarrolladores acceder y manipular los elementos del documento de manera programática. Por otro lado, el BOM se refiere a los objetos proporcionados por el navegador web para interactuar con la ventana del navegador.

Hemos cubierto los siguientes temas clave:

1. Acceso y Selección de Elementos del DOM:

- Métodos como getElementById, getElementsByTagName, getElementsByClassName, y querySelector.

2. Manipulación del DOM:

- Modificar el contenido y los atributos de los elementos usando propiedades como textContent, innerHTML, y outerHTML.

3. Creación y Manipulación de Nuevos Elementos:

- Crear nuevos elementos HTML mediante document.createElement y adjuntarlos al DOM con métodos como appendChild y insertBefore.

4. Eventos:

- Manejo de eventos del ratón, teclado, y otros eventos del usuario utilizando addEventListener.
- Uso del objeto event para obtener información detallada sobre los eventos.

5. Formularios y Validación:

- Captura y manejo de eventos de envío de formularios.
- Validación de formularios utilizando expresiones regulares.

6. Manipulación de Clases CSS:

- Añadir, quitar y alternar clases CSS en elementos del DOM con classList.

7. Eventos de Desplazamiento (Scroll):

- Detección y manejo del evento de desplazamiento en la ventana para mostrar la posición actual del scroll.

8. Clonación y Remoción de Elementos del DOM:

- Clonar elementos existentes y eliminarlos del DOM mediante cloneNode y removeChild.

4.13. BIBLIOGRAFÍA

- Mozilla Developer Network (MDN) URL: <https://developer.mozilla.org/>
- W3Schools URL: <https://www.w3schools.com/>
- DesarrolloWeb.com URL: <https://desarrolloweb.com/>
- AMX Web URL: <https://amxweb.com/>