Unidad 5. Modelo de objetos del documento en JavaScript.
Boletín de actividades

Eventos

JavaScript es un lenguaje que se pensó para dotar de interactividad a las páginas web, para que las páginas reaccionaran ante las acciones de los usuarios.

Cada vez que el usuario realiza alguna acción sobre la página, el navegador lanza automáticamente un evento. Actualmente hay más de 200 eventos que pueden ocurrir en un sitio no todos relacionados con las acciones del usuario. Por ejemplo, el evento `load` sucede cuando la página se ha cargado. Desde JavaScript se puede implementar qué sucederá cuando esos eventos ocurran referenciando una función que se llevará a cabo en tal caso. Para programar a qué función llamar cuando ocurra un determinado evento, se debe añadir *manejador de evento* o *event listener*.

Existen varios modelos de gestión de eventos:

- Modelo en línea/tradicional: atributos/propiedades `onclick`, `onsubmit` entre otras.
- Modelo avanzado (W3C): funciones `addEventListener` y `removeEventListener`.
- Modelo de Microsoft: funciones `attachEvent` y `detachEvent`.

Cada modelo utiliza una forma de añadir y quitar manejadores de eventos. No se puede usar un modelo para quitar un manejador de evento que se ha añadido con otro.

Actividad 1. Añadir un manejador de evento.

Implementa una aplicación que abra una ventana emergente (`alert`) al pulsar sobre un botón. Añade un manejador del evento `click` al botón usando:

a) El modelo de eventos en línea/tradicional.
b) El modelo de eventos avanzado.
c) El modelo de eventos avanzado compatible con versiones de Internet Explorer anteriores a la 9. Consulta el ejemplo final de W3Schools.

Actividad 2. Objeto `Event`.

Implementa una aplicación que abra una ventana emergente (`alert`) al pulsar una tecla sobre un campo de texto. La ventana debe mostrar la tecla que se ha pulsado.

Consulta el artículo KeyboardEvent key Property de W3Schools.

Actividad 3. Eliminar un manejador de evento.

Modifica la aplicación anterior para que al mostrarse la ventana emergente por primera vez se deje de llamar a la función para mostrar la tecla pulsada.

Actividad 4. Captura o burbujeo.

La función addEventListener cuenta con un tercer parámetro booleano que indica si la función asociada al evento debería llamarse en la fase de captura o de burbujeo.

a) ¿Qué es la fase de captura y de burbujeo al producirse un evento?
b) ¿Qué utilidad tiene la función `stopPropagation`?

Referencias:

- Artículo de addEventListener en W3Schools.
- Artículo de stopPropagation en W3Schools (Prueba el primer ejemplo *Try it yourself*)

Actividad 5. Evitar el comportamiento por defecto.

Hay elementos en HTML a los que el navegador ya asocia un comportamiento por defecto al producirse un evento sobre ellos. Por ejemplo al crear un botón `submit` en un formulario o al crear un enlace.

Evitar este comportamiento predeterminado es posible usando la función `preventDefault` del objeto `Event`. Prueba el ejemplo del artículo de preventDefault en la web de W3Schools para conocer su comportamiento.

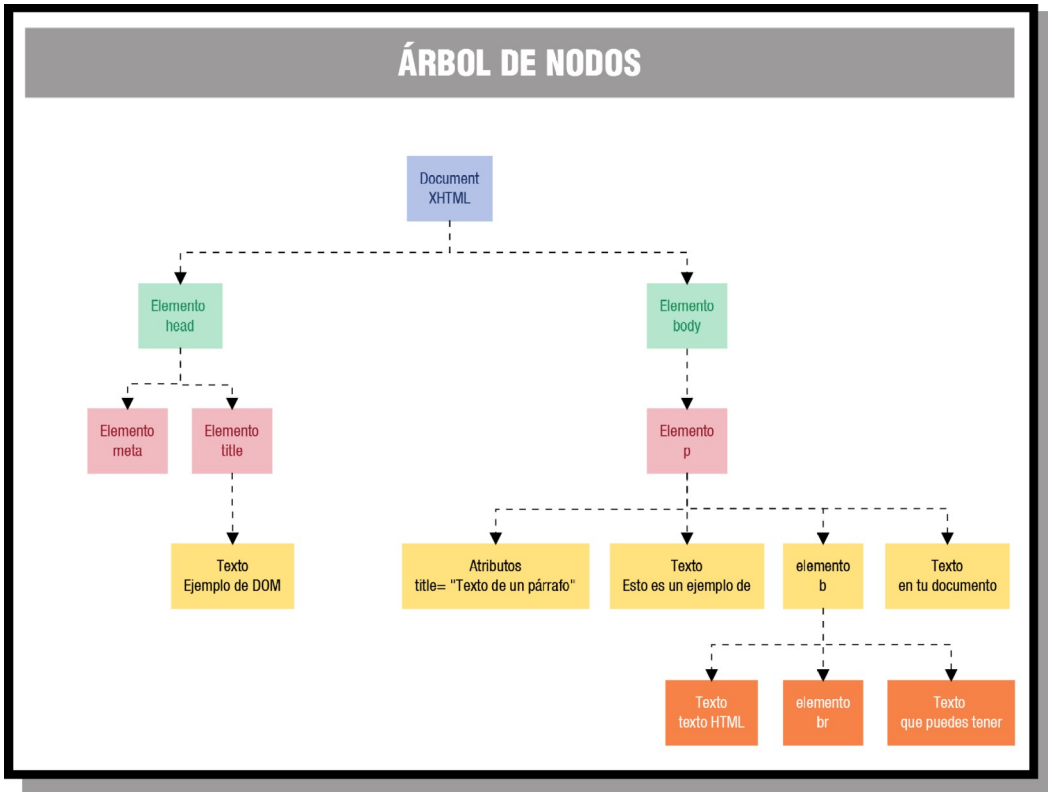
Modelo de objetos del documento

El DOM (Modelo de Objetos del Documento) es un conjunto de objetos que representan los elementos HTML de una página. Desde JavaScript se pueden manipular estos objetos para hacer cambios en una página.

Al abrir una nueva página, los navegadores transforman cada elemento en un objeto y los relaciona de la misma forma que en el documento creando una estructura jerárquica en forma de árbol. Dentro de la terminología del DOM, a cada objeto se le denomina *nodo* y a la estructura resultante *árbol de nodos*.

Por ejemplo, para el siguiente código HTML, el navegador creará una estructura de objetos como la mostrada:

```
<!DOCTYPE html>
<html>
	<head>
		<meta charset=utf-8">
		<title>Ejemplo de DOM</title>
	</head>
	<body>
		<p title="Texto de un párrafo">Esto es un ejemplo de <b>texto
		HTML<br>que puedes tener</b> en tu documento.</p>
	</body>
</html>
```



La raíz del árbol es un nodo especial denominado `document`. A partir de este nodo, cada elemento se transformará en un nodo de tipo `element`, `attribute`, `text` o `comment`. Los nodos de tipo `text` contienen el texto encerrado en una etiqueta. Para cada nodo, el nodo inmediatamente superior será el *nodo padre* y todos los que están por debajo serán los *nodos hijos*.

Carpeta de recursos de la unidad: Google Drive.

Actividad 1. Nodos de tipo `Element`.

Consulta el apartado *1.3. Acceso a los nodos* de los contenidos de la plataforma y realiza las siguientes tareas:

a) Recupera el `div` identificado como `content`.
b) Recupera todos los nodos que tengan asignada la clase `callout`.
c) Recupera todos los nodos que sean párrafos.
d) Recupera el primer párrafo dentro del elemento identificado como `content`.

Actividad 2. Nodos de tipo `Attribute`.

Consulta el apartado *1.4. Acceso a los nodos de tipo atributo* de los contenidos de la plataforma y realiza las siguientes tareas:

a) Muestra por consola el nombre y el valor de todos los atributos del elemento `div` identificado como `callout2`.

b) Añade el atributo `title` con el valor `Información sobre identificadores` al tercer párrafo del elemento identificado como `content`.

c) Añade el atributo `title` con el valor `Información sobre clases` al cuarto párrafo del elemento identificado como `content`.

d) Obtén el nombre de la hoja de estilo del documento HTML.

e) Obtén la codificación de caracteres del documento HTML.

f) Elimina los atributos `title` añadidos anteriormente a los párrafos.

Actividad 3. Nodos de tipo `Text`.

Consulta el apartado *1.5. Acceso a los nodos de tipo texto* de los contenidos de la plataforma y a continuación recupera el contenido del párrafo dentro del `div` identificado como `callout2` usando:

a) La propiedad `innerHTML`.
b) La propiedad `textContent`.
c) La propiedad `nodeValue`. Ten en cuenta que el primer hijo de un elemento es el texto que contiene pero comprueba que no haya espacios en blanco entre el elemento `div` y `p`. Puede tomar el primer hijo de `div` como un espacio en blanco.

Actividad 4. Creación y borrado de nodos.

Consulta los apartados *1.6. Creación y borrado de nodos*, *1.5. Acceso a los nodos de tipo texto (*enlace *Alternativas al uso de innerHTML)* y *1.7. Propiedades y métodos de los objetos nodo.*

A continuación realiza las siguientes tareas:

a) Crea un nuevo párrafo con el texto `Me cuelo en primera posición` y añádelo antes del primer hijo del elemento identificado como `content`.

b) Crea un nuevo párrafo con el texto `Me coloco en última posición` y añádelo como último hijo del elemento identificado como `content`.

c) Crea la siguiente estructura y añádela a continuación del elemento `h1`:

```
<p title="Página sencilla" id="descripcion">Esto es un
ejemplo de <em>página HTML</em> sencilla.</p>
```

d) Elimina la estructura anterior.

e) Crea una estructura igual a la del elemento HTML con clase `callout` y añádela justo después de éste.

Actividad 5. Manipulación de elementos con atributo `class`.

El atributo `class` permite marcar un conjunto de elementos del documento HTML. Mediante la propiedad `classList` se pueden consultar las clases asignadas a un determinado elemento y usando los métodos `add` y `remove` se pueden añadir o eliminar clases sobre un elemento.

Implementa las siguientes funciones:

a) `function destacar(palabra) { ... }`
Esta función aplica la clase CSS `destacado` a todos los párrafos del documento que contengan la palabra recibida como parámetro.

b) `function eliminarDestacados() { ... }`
Esta función elimina la clase CSS `destacado` de todos los párrafos del documento que la tengan.

Clase CSS `destacado`:

```
.destacado {
     background: #ff0;
     font-style: italic;
}
```

Actividad 6. Manipulación de elementos con atributo `data`.

Los atributos `data` permiten añadir atributos personalizados a los elementos de una página. Por ejemplo podría utilizar atributos data-sexo para añadir información sobre el sexo en una lista de superhéroes:

```
<li id="marvel-1" data-sexo="M">Capitán América</li>
<li id="marvel-2" data-sexo="F">Capitana Marvel</li>
```

Desde JavaScript se puede acceder a estos atributos con la propiedad `dataset`. En el ejemplo anterior:

```
let capitanAmerica = document.getElementById("marvel-1");
let sexoCapitanAmerica = capitanAmerica.dataset.sexo;
```

Un uso práctico de estos atributos puede ser marcar botones con una determinada acción para luego identificarlos desde javascript. Para probar su funcionamiento añade al documento el siguiente código HTML:

```
<button data-accion=”destacar” data-contenido=”único”>
     Destacar los párrafos que contengan la palabra “único”
</button>

<button data-accion=”eliminarDestacados”>
     Eliminar párrafos destacados
</button>
```

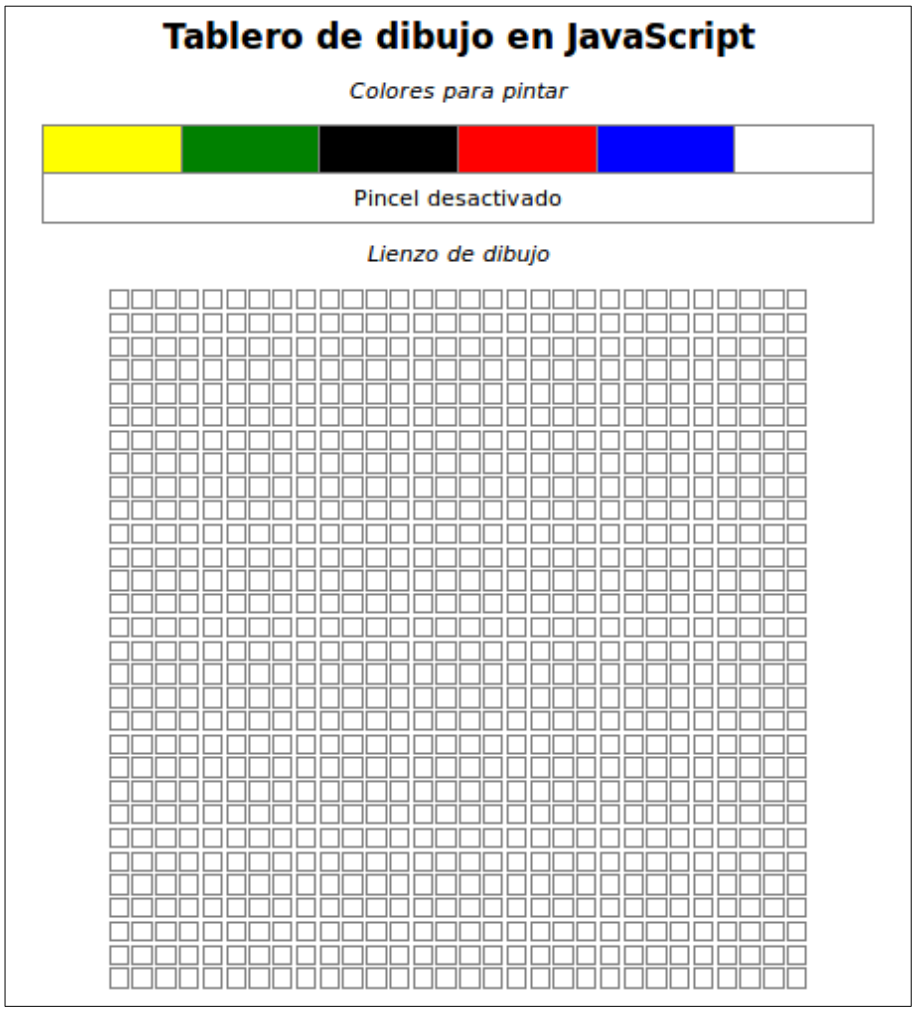
A continuación implementa estas funciones:

a) Al hacer clic sobre cualquier elemento marcado con el atributo `accion` valor `destacar`, se aplique la clase CSS `destacado` a todos los párrafos que contengan la palabra definida por el atributo `contenido`. Utiliza la función `destacar` implementada en la actividad anterior.

b) Al hacer clic sobre cualquier elemento marcado con el atributo `accion` valor `eliminarDestacados`, se elimine la clase `destacado` de todos los párrafos que la tengan. Utiliza la función `eliminarDestacados` implementada en la actividad anterior.

Actividad 2. Tablero de dibujo [carpeta: tableroDibujo]

Crea un tablero de dibujo como el siguiente a partir de los ficheros proporcionados.



Apartado A. Estructura.

El título “Colores para pintar” y la tabla de colores se ubican en el div `paleta`.

El título “Lienzo de dibujo” y la tabla en la que pintar se ubican en el div `lienzo`.

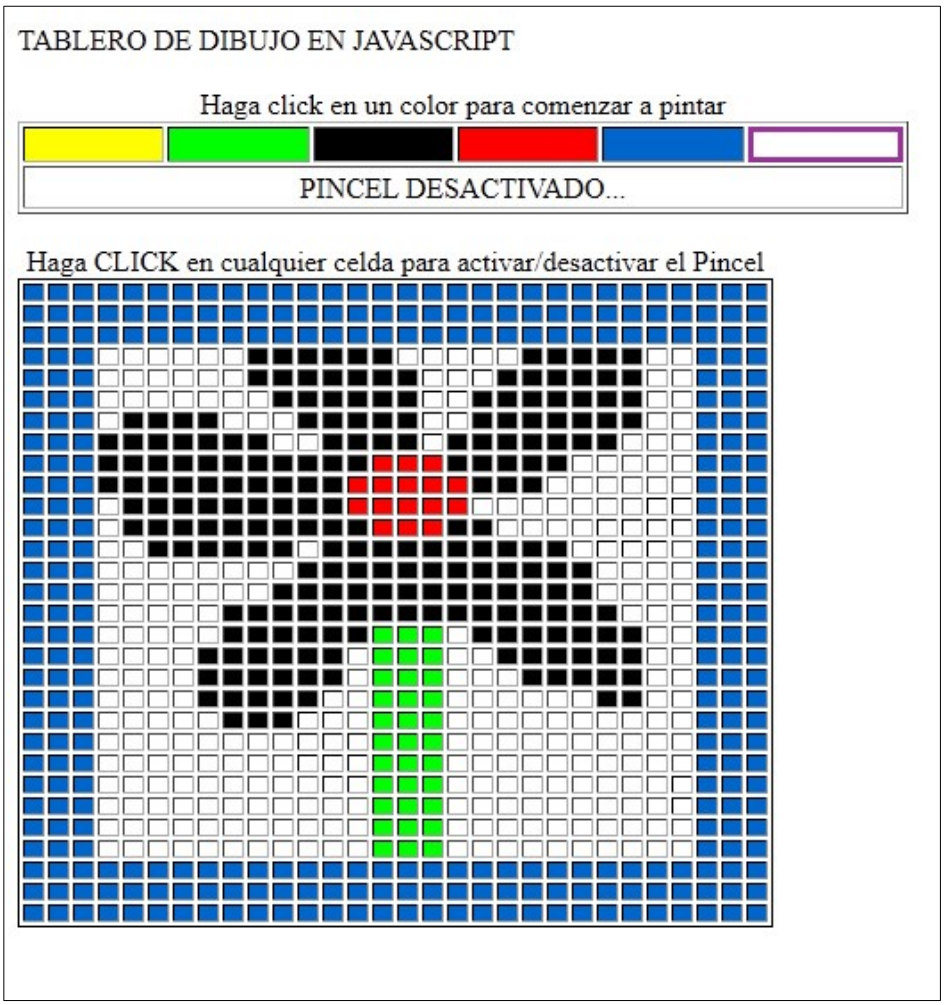
Crea estos elementos desde JavaScript utilizando de 3 funciones:

- `crearPaleta`: Crea una tabla de colores (table) como la mostrada. Se debe asignar a cada celda una clase CSS para definir su color (clases ya incluidas en la hoja de estilos).
- `crearLienzo`: Crea una tabla para pintar (table) como la mostrada de 30x30 celdas. Las dimensiones del lienzo se deben poder variar fácilmente.
- `crearTitulos`: crea los títulos “Colores para pintar” (h2) y “Lienzo de dibujo” (h2) en su ubicación correspondiente.

Apartado B. Comportamiento.

Implementa la siguiente funcionalidad para la aplicación:

- El usuario hace clic en alguno de los colores de la paleta quedándole asignada la clase “seleccionado”.
- A continuación, el usuario hace clic en una celda del tablero que se pintará del color seleccionado y desde ese momento se pintarán de ese color todas las celdas por las que el usuario pase el ratón. Para dejar de pintar, el usuario debe volver a hacer clic en una celda.
- El usuario puede escoger un color diferente y repetir el proceso, incluso sobre celdas que ya han sido pintadas.
- Cada vez que el pincel esté activado, se mostrará un mensaje debajo de la paleta indicando “Pincel activado”. En caso contrario, se mostrará “Pincel desactivado” (color blanco).

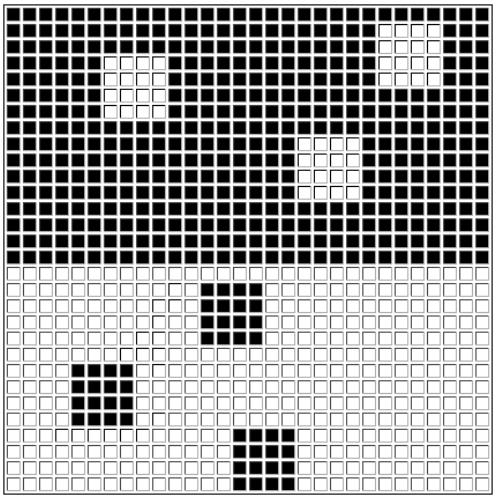


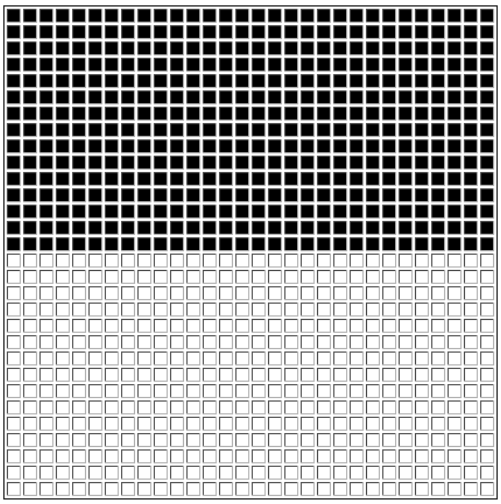
Apartado C. Cross-browser.

Añade las modificaciones necesarias para que la aplicación funcione en todos los navegadores.

Ejercicio 3. El buen equilibrio.

Implementa el juego “el buen equilibrio”. El buen equilibrio se consigue cuando la mitad superior del tablero es de color negro y la mitad inferior es de color blanco.





El tablero tiene 30x30 celdas de tamaño 10x10 píxeles dentro del div “zonatablero”.

El funcionamiento del juego es el siguiente:

- Al comenzar el juego cada mitad tendrán 3 cuadrados de 4x4 celdas del color contrario situados de forma aleatoria.
- Cada 10 segundos aparecerá un nuevo cuadrado en cada lado.
- El usuario debe instaurar el buen equilibrio con una palera de dos colores, blanco y negro, situada debajo del tablero. El funcionamiento de la paleta de colores es el mismo de la actividad anterior.
- El usuario gana el juego cuando la mitad superior es completamente negra y la inferior blanca. Si el usuario no restablece el buen equilibrio en 1 minuto, pierde la partida.

Añade las modificaciones necesarias para que la aplicación funcione en todos los navegadores.

Ejercicio 4. Lista interactiva [carpeta: listaInteractiva]

Implementa una lista en la que los elementos puedan cambiar de posición utilizando dos botones para subir y bajar.

 Item de la lista.

Cada elemento de la lista interactiva será un elemento div:

```
<div id="orden">
      <div id="uno" class='item'>Primer elemento.</div>
      <div id="dos" class='item'>Segundo elemento.</div>
      <div id="tres" class='item'>Tercer elemento.</div>
      <div id="cuatro" class='item'>Cuarto elemento.</div>
</div>
```

Apartado A. Añadir flechas.

Añade las imágenes para subir y bajar cada elemento de la lista. La estructura resultante debe ser como la siguiente:

```
<div id="orden">
      <div id="uno" class="item">
            <div class="img UP"/>
            <div class="img DW"/>
            <div class="texto">Primer elemento.</div>
      </div>
      <div id="dos" class="item">
            <div class="img UP"/>
            <div class="img DW"/>
            <div class="texto">Segundo elemento.</div>
...
```

Apartado B. Lista secuencial.

Implementa el funcionamiento de los botones para que los elementos de la lista suban y bajen cuando corresponda. Si el primer elemento intenta subir o el último bajar, las acciones de los botones quedarán sin efecto.

Apartado C. Lista circular.

Cambia el comportamiento del primer y último elemento, de manera que cuando el primer elemento suba se coloque en la última posición y cuando el último elemento baje se ponga en la primera.

Apartado D. Información del orden.

Implementa un botón que muestre el orden de los elementos de la lista en un determinado momento.

Añade las modificaciones necesarias para que la aplicación funcione en todos los navegadores.