Unidad 5. Modelo de objetos del documento en JavaScript.

Boletín de actividades

Contenido

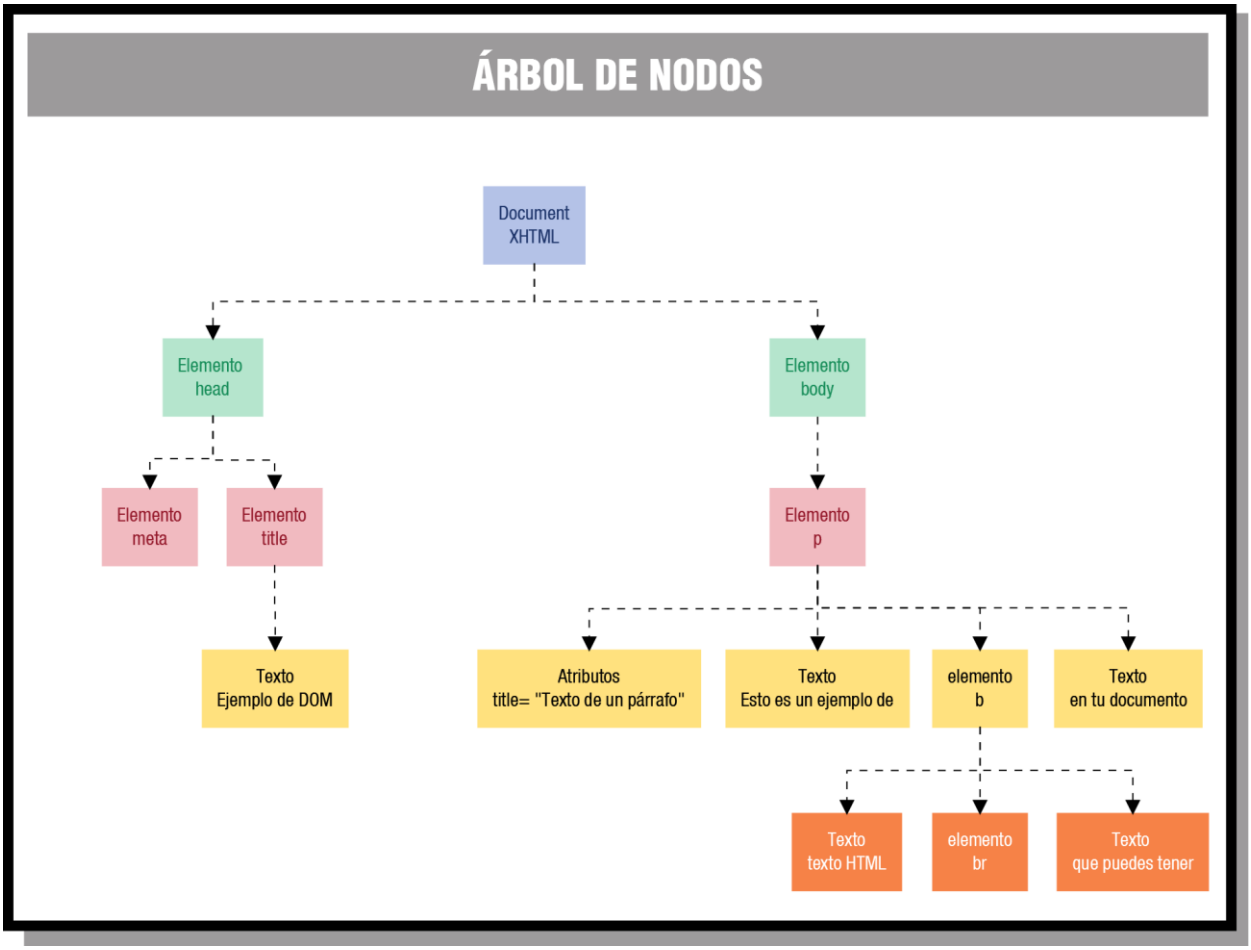
1. Modelo de objetos del documento

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*.

Descarga la carpeta `introduccion` de la 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 realiza las siguientes tareas:

a) Recupera el contenido del párrafo dentro del `div` identificado como `callout2` usando:

- La propiedad `innerHTML`.
- La propiedad `textContent`.
- 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.

b) Modifica el contenido del primer párrafo del `div callout2` para añadir el texto "*pero bien formado*."

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) Realiza las actividades a) y b) usando la función `insertAdjacentElement`.

d) 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>
```

e) Elimina la estructura anterior.

f) 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. Otros métodos muy útiles de `classList` son:

- `contains(String)`: método que comprueba si en la lista de clases existe una clase determinada, devolviendo un booleano.
- `replace(claseExistente, claseNueva)`: reemplaza una clase existente por otra clase nueva (elimina la primera y añade la segunda a la lista).
- Otros métodos que pueden resultar útiles y que puedes consultar en la página de referencia de MDN.

Implementa las siguientes funciones:

a) `function destacar(sPalabra) { ... }`
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.

2. Eventos

Un evento se puede entender como cualquier acción que ocurre en una web ya sea realizada por el usuario o por el propio navegador. En JavaScript se puede suscribir un *manejador* a un evento para conseguir que una función se ejecute cuando el evento suceda.

Existen una gran cantidad de eventos, quizás los más sencillos de entender sean los eventos de ratón como *click* o *dblclick*. Puedes ver un listado extenso de eventos en este artículo de HTML DOM Event de W3Schools.

A la hora de suscribir manejadores para los eventos encontramos 3 modelos de gestión:

- 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.

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`.

Cada vez que un evento ocurre, el navegador pone a disposición del programador un objeto Event que contiene información sobre el evento. 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.

3. Prácticas propuestas.

Carpeta de recursos de la unidad: Google Drive.

Actividad 1. Cartel que persigue al ratón.

Primera crea una página web con bastantes párrafos. Para generarlos de forma automática con *VS Code* puedes escribir en el `body: p*50>lorem` y pulsar tabulador.

A continuación haz que en cada movimiento del ratón sobre la página, un pequeño cartel con fondo amarillo y texto *Soy una lapa* siga al cursor del usuario.

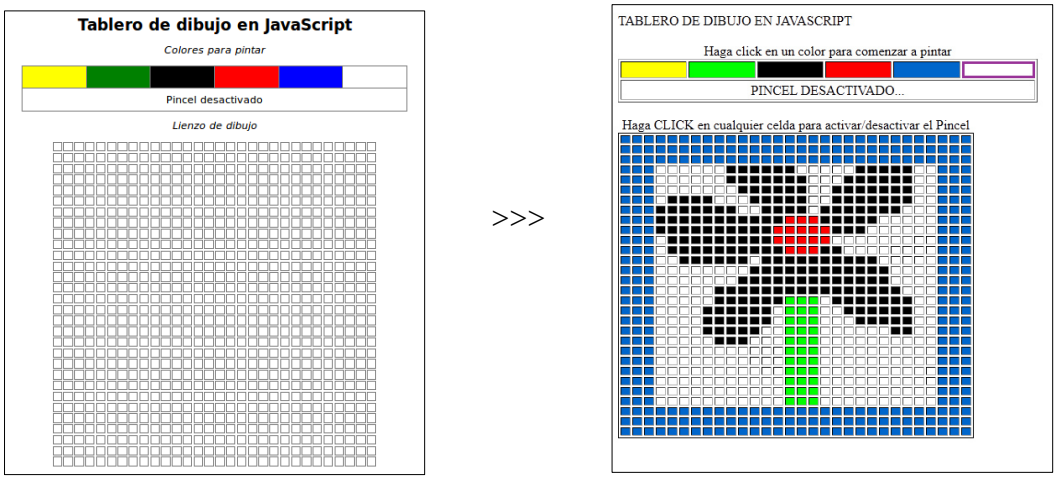
Actividad 2. Control de velocidad

Crea una aplicación que muestre el texto Velocidad 0 y si pulsamos la flecha hacia arriba del teclado la velocidad se incremente en 1 y si la pulsamos hacia abajo baje en 1.

La velocidad no puede exceder de 120 ni bajar de 0.

Actividad 3. Tablero de dibujo.

Partiendo de los ficheros de la carpeta `tableroDibujo`, crea un tablero de dibujo como el éste:



Apartado A. Creación de elementos.

Crea los elementos HTML utilizando las siguientes funciones:

- `crearPaleta`: Crea una tabla HTML para representar la paleta de colores. A cada celda se le aplicará una clase CSS que definirá su color. Las clases están incluidas en la hoja de estilos.
- `crearLienzo`: Crea una tabla HTML de 30 x 30 celdas para representar el lienzo sobre el que pintar. Las dimensiones del lienzo se deben poder variar con facilidad.
- `crearTitulos`: crea los títulos `h2` *Colores para pintar* y *Lienzo de dibujo*.

Sobre la ubicación de los elementos:

- El título *Colores para pintar* y la tabla de colores se ubicarán en el div `paleta`.
- El título *Lienzo de dibujo* y la tabla en la que pintar se ubicarán en el div `lienzo`.

Apartado B. Comportamiento.

La aplicación debe tener el siguiente comportamiento:

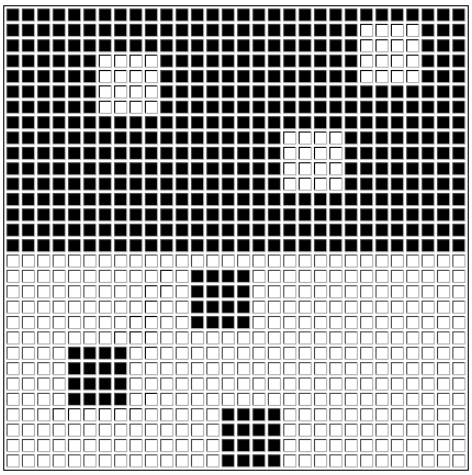
- Cuando el usuario haga clic en alguno de los colores de la paleta se le asignará la clase `seleccionado` a la celda con dicho color.
- Cuando el usuario hace clic en una celda del lienzo el pincel se activa quedando pintada la celda del color seleccionado. Desde ese momento todas las celdas por las que el usuario pase el ratón por encima también se pintarán de dicho color.
- Cuando el usuario haga clic en cualquier celda, el pincel se desactivará y dejará de pintar.

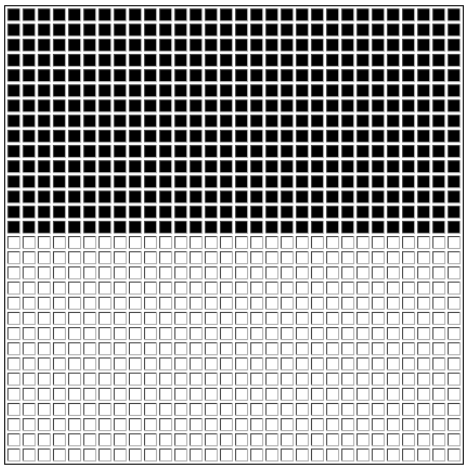
Otras consideraciones:

- Las celdas del lienzo se pintarán del color seleccionado incluso si fueron previamente pintadas.
- Se mostrará un mensaje debajo de la paleta indicando si el pincel está activado o desactivado (color Blanco).

Ejercicio 4. El buen equilibrio.

Implementa el juego de *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.





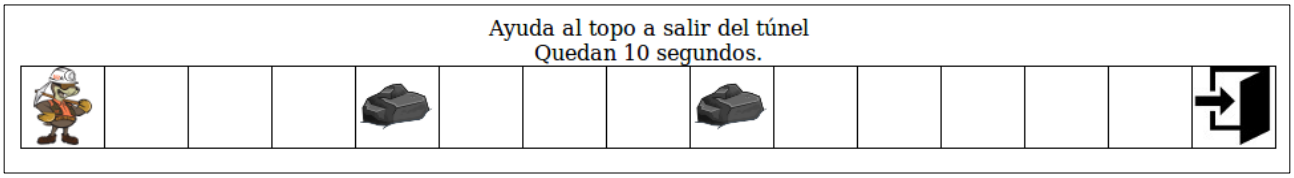
El tablero tiene unas dimensiones de 30x30 celdas de tamaño 10x10 píxeles que se ubicarán dentro del div `zonatablero`.

El funcionamiento del juego es el siguiente:

- Al comenzar el juego cada mitad tendrán 3 cuadrados de 4x4 celdas del color contrario. Los cuadrados se situarán en posiciones aleatorias respetando los límites del tablero.
- Cada 15 segundos aparecerá un nuevo cuadrado en cada lado.
- El usuario debe instaurar el buen equilibrio pintando de blanco o negro según corresponda. Se usará una paleta de dos colores con el mismo funcionamiento que la actividad anterior.
- Si el usuario no restablece el buen equilibrio en 1 minuto, pierde la partida.

Ejercicio 5. La huida del topo.

Implementa el juego *La huida del topo*. El objetivo es hacer llegar el topo a la salida antes de que se acabe el tiempo.



El juego funciona de la siguiente forma:

- El topo avanza una celda cuando se hace clic encima. Si la siguiente casilla es una roca, el topo no puede avanzar.
- Las rocas se pican haciendo doble clic sobre ellas. Una roca sólo se puede picar cuando el topo está junto a ella.
- Si el topo llega a la salida antes de 15 segundos el usuario gana la partida.

Ejercicio 6. What-a-mole.

Implementa el juego ¡Atrapa el topo! que consiste en conseguir hacer clic sobre el topo el mayor número de veces. Descarga los archivos del juego y realiza las siguientes actividades en el fichero JavaScript:

1. Crea el tablero de juego. El tablero siempre será un cuadrado que se creará en función de una constante I_TAM_TABLERO. El tablero se debe colocar en el div "board". A continuación se muestra el código HTML para un tablero de tamaño 2 y de tamaño 3.

```html
<!--Tablero de tamaño 2-->
<div id="board">
  <div class="row">
    <div class="square mole"></div>
    <div class="square"></div>
  </div>
  <div class="row">
    <div class="square"></div>
    <div class="square"></div>
  </div>
</div>
```

```html
<!--Tablero de tamaño 3-->
<div id="board">
  <div class="row">
    <div class="square mole"></div>
    <div class="square"></div>
    <div class="square"></div>
  </div>
  <div class="row">
    <div class="square"></div>
    <div class="square"></div>
    <div class="square"></div>
  </div>
  <div class="row">
    <div class="square"></div>
    <div class="square"></div>
    <div class="square"></div>
  </div>
</div>
```

2. Declara una variable "iPuntuacion" para almacenar la puntuación del usuario. Inicialízala a 0. Declara otra variable "iTiempoRestante" para almacenar el tiempo que le queda al usuario en la partida. Inicialízala a 30. Muestra estos valores iniciales al usuario a través de los elementos identificados como "score" y "time-left" en el HTML. Identifícalos como "oPuntuacion" y "oTiempoRestante" respectivamente.

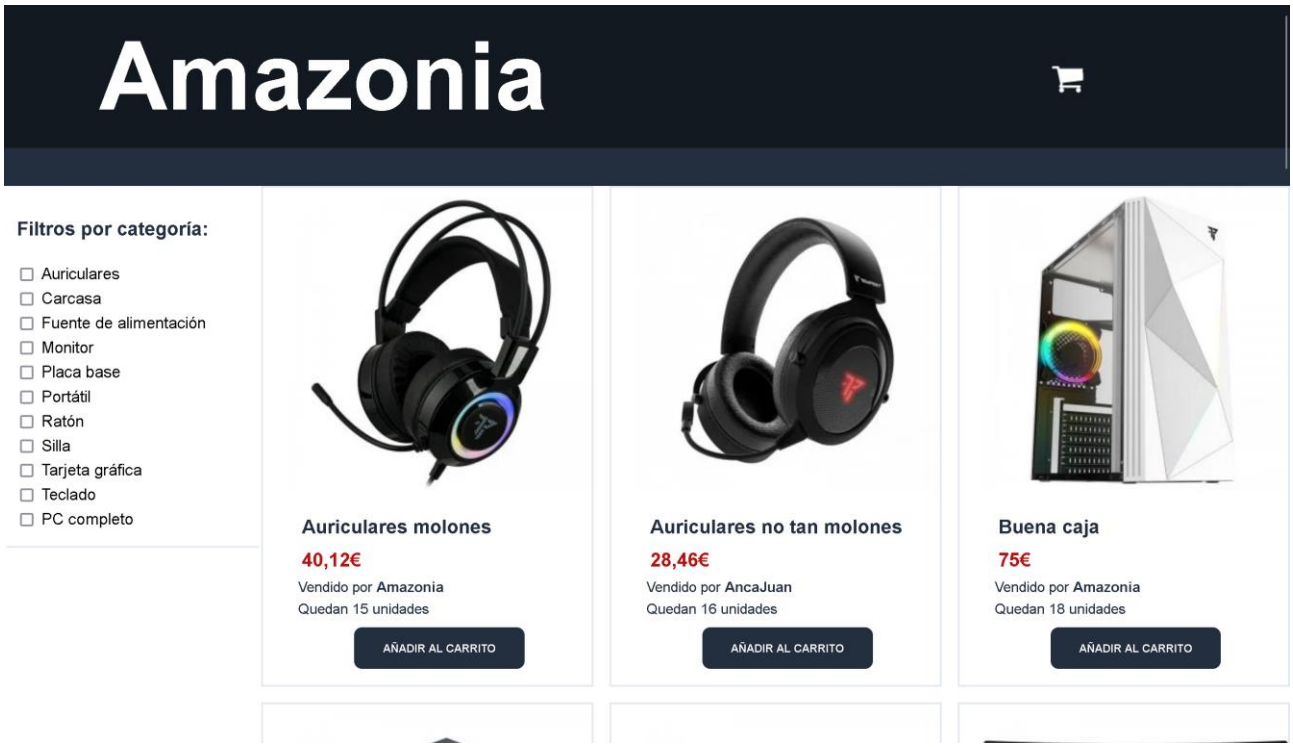
3. Añade un manejador para el evento "click" a la casilla en la que se encuentre el topo. Cuando suceda se debe llamar a la función "aumentarPuntuacion". Implementa dicha función que aumenta la puntuación del usuario.

4. Implementa un temporizador que llame a la función "cambiarTopoCasilla" cada medio segundo. Implementa la función "cambiarTopoCasilla" que quita el topo de la casilla en la que se encuentre y lo coloca en una casilla elegida de forma aleatoria.

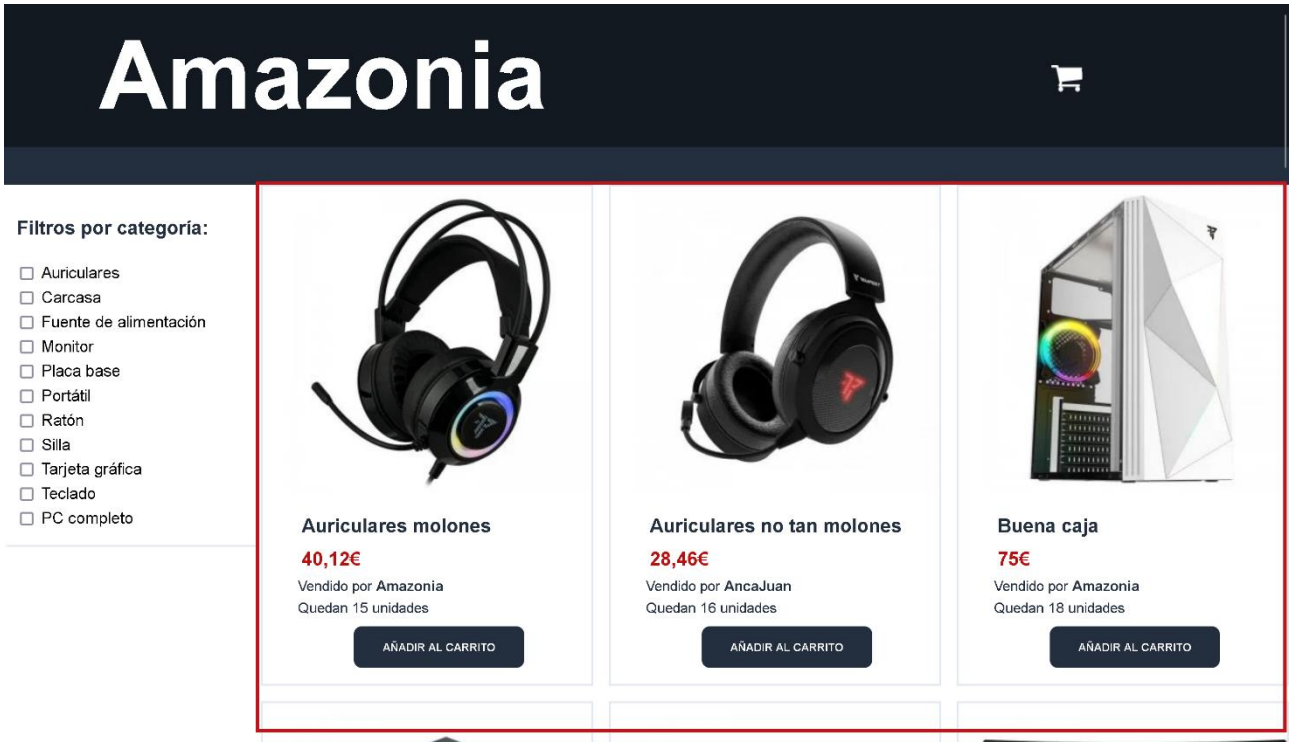
5. Implementa un temporalizador que llame a una función "decrementarTiempo" cada segundo. Implementa la función "decrementarTiempo" que disminuye el tiempo restante. Si no queda más tiempo, el topo debe pararse y mostrar al usuario la puntuación final obtenida en una ventana emergente.

Ejercicio 7. Tienda online.

Implementa la funcionalidad que se indica en cada apartado para la página web de una tienda online con la siguiente apariencia:



Apartado 1) A partir del array de productos llamado ***productos*** que se importa en el archivo ***script.js***, rellena el ***div*** con ***id="products-container"*** con los productos como se indica en la siguiente imagen:



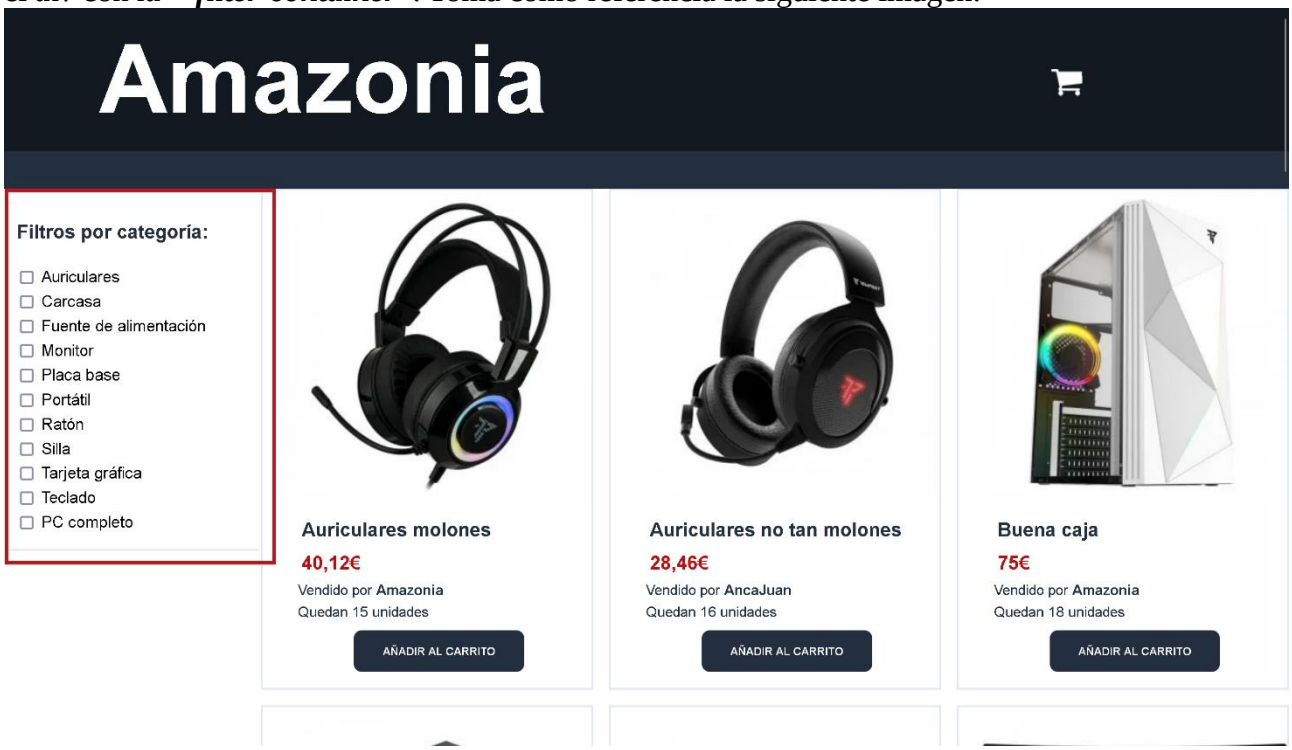
Debes tener en cuenta el siguiente código HTML correspondiente a cada uno de los productos de que debes incluir:

```
<article id="007" class="location-listing" data-categoria="005">
    <div class="location-image">
        <a href="#">
            <img src="res/art007.webp" alt="Placa base Asus">
        </a>
    </div>
    <div class="data">
        <h4>Placa base Asus</h4>
        <p class="price">187,45€</p>
        <p>Vendido por <strong>Amazonia</strong></p>
        <p>Quedan 6 unidades</p>
        <div class="button-container">
            <a class="button add" href="#" target="_blank">Añadir al carrito</a>
        </div>
    </div>
</article>
```

Notas acerca del código HTML:

- El atributo ***id*** del ***article*** será el ***id*** de cada producto.
- El atributo ***data-categoria*** será el contenido del atributo ***categoría*** (el id de la categoría)
- El atributo ***src*** de la imagen será el contenido en el atributo ***imagen*** de cada producto.
- El atributo ***alt*** de la imagen será el nombre del producto.
- El texto del ***h4*** será el ***nombre*** del producto.
- El precio (187,45) coincidirá con el atributo ***precio*** del producto.
- El vendedor (Amazonia) será el valor del atributo ***vendedor*** del producto.
- El número de productos que quedan (6) coincidirá con el atributo ***stock*** del producto.

Apartado 2) Se debe implementar la funcionalidad del filtro por categorías. Para ello se deben obtener las categorías diferentes que hay entre los productos e inyectar el código correspondiente en el ***div*** con ***id="filter-container"***. Toma como referencia la siguiente imagen:



El código HTML que hay que incluir es el siguiente (ejemplo de dos categorías):

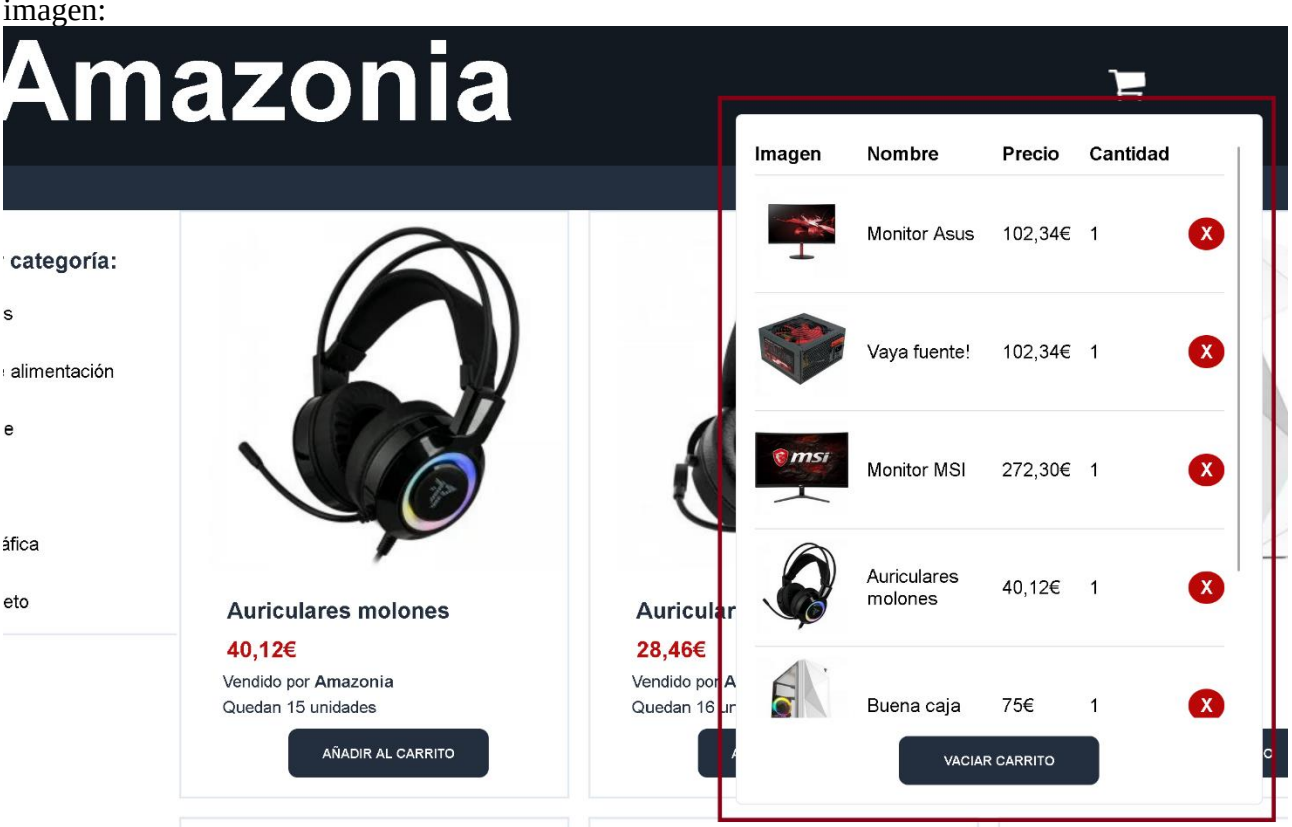
```
<form>
    <fieldset id="filtro-categoria" name="filtro-categoria">
        <legend>Filtros por categoría:</legend>
        <div class="contenedor-categoria">
            <input type="checkbox" id="001" name="001" value="001">
            <label for="001">Auriculares</label>
        </div>
        <div class="contenedor-categoria">
            <input type="checkbox" id="004" name="004" value="004">
            <label for="004">Monitor</label>
        </div>
    </fieldset>
</form>
```

Notas acerca del código HTML:

- Los atributos ***id***, ***name*** y ***value*** del ***checkbox*** coinciden con el ***id*** de la categoría.
- El atributo ***for*** del ***label*** coincide con el ***id*** de la categoría.
- El texto contenido en el ***label*** será el ***nombre*** de la categoría.

Apartado 3) Implementa la funcionalidad de incluir productos en el carrito de la compra, teniendo en cuenta lo siguiente:

- Al pulsar uno de los botones de los productos "AÑADIR AL CARRITO", el producto se debe almacenar en ***LocalStorage*** y representarse en el carrito como se indica en la siguiente imagen:



- El código HTML del carrito se añadirá dinámicamente dentro del elemento ***tbody*** de la ***table*** con ***id="lista-carrito"***.
- El código que hay que añadir es:

```
<tr>
    <td>
            <img src="res/art005.webp">
    </td>
    <td>Monitor Asus</td>
    <td>102,34€</td>
    <td>1</td>
    <td>
            <a href="#" class="borrar-curso" data-id="005">X</a>
    </td>
</tr>
```

Notas del código: se ve claramente de arriba abajo que se debe inyectar alguna información de los productos: ***imagen***, ***nombre***, ***precio***, ***cantidad*** (siendo cantidad un nuevo atributo que hay que añadir en los productos antes de almacenarlos en ***LocalStorage***.

- Si un producto se vuelve a añadir (se vuelve a pulsar el botón "AÑADIR AL CARRITO") se suma uno a la columna de cantidad. También se debe actualizar el producto en el ***LocalStorage***.
- Si se pulsa un botón rojo (con "x" en el centro) que aparece en cada entrada del carrito, dicho elemento se eliminará del carrito y del ***LocalStorage***.

- Si se pulsa en el botón “VACIAR CARRITO” del carrito, se eliminan todos los elementos del carrito y del ***LocalStorage***.

Notas importantes:

- Se recomienda utilizar el patrón **MVC** tanto para la fuente de datos que tenemos en el archivo ***db.js*** como para los datos del ***LocalStorage***.
- Cada fuente de datos tendrá un controlador diferente.
- Podemos definir un archivo ***.js*** nuevo para los controladores (***controlador.js***).
- Para encapsular mejor la aplicación, se podría definir una clase para cada controlador, por ejemplo: ***ControladorDB*** (para trabajar sobre el archivo ***db.js***) y ***ControladorCarrito*** (para trabajar sobre el ***LocalStorage***). Las clases tendrán todos sus métodos estáticos y simplemente se usarán para encapsular cada funcionalidad.
- Mediante la implementación del patrón **MVC** se facilitará la adaptación de la aplicación en temas posteriores, cuando se utilice como fuente de datos la consulta a un servidor mediante peticiones asíncronas.
- También es recomendable crearse una clase de utilidad (***class Util***) en otro archivo (***util.js***) que será una clase que contendrá un compendio de métodos estáticos que se suelen utilizar entre diferentes proyectos. De este modo, se evitará el desarrollo reiterativo de algoritmos que pueden ser reutilizados.
- Puedes integrar toda la funcionalidad mediante importación y exportación de módulos (como vimos en la unidad 3).