

# HTML

# CSS



Nuevas APIs de HTML5. Responsive Design.

## 1. Objetivos

En este sexto tema se pretenden conseguir los siguientes objetivos:

- ✚ Conocer las diferentes APIs que aparecen con el estándar HTML5.
- ✚ Utilizar la API Web Storage tanto a nivel de sesión como a nivel local.
- ✚ Entender las ventajas del Responsive Web Design.
- ✚ Crear páginas web adaptables que se visualicen correctamente en distintos dispositivos.
- ✚ Modificar páginas web no adaptables para convertirlas en adaptables.

## 2. Introducción

A lo largo del curso hemos estudiado varias APIs nuevas que nos proporciona HTML5, en concreto, hemos visto la API para manejo de audio y vídeo en el tema 3, el API para manejo de formularios en el tema 4 y la API de dibujo por medio del elemento canvas en el tema 5.

En este tema, veremos otras nuevas APIs de desarrollo que nos proporciona este estándar que nos ayudarán a desarrollar aplicaciones web con una potencia similar a las aplicaciones de escritorio. Estas nuevas APIs son: **Drag & Drop, Geolocation, Storage, File Access, Communication, Web Workers, History y Offline.**

No estudiaremos estas API en profundidad, simplemente, las introduciremos y trabajaremos algo más una de ellas, la API Web Storage, pero sin entrar en demasiado detalle. En lo que se refiere a CSS3, veremos cómo conseguir que nuestras páginas web se adapten a distintas resoluciones o, lo que es lo mismo, a distintos dispositivos. Esto se conoce como **Responsive Web Design**.

Estudiaremos el origen del Responsive Web Design, las ventajas que nos aporta y de qué herramientas disponemos para llevarlo a cabo. En concreto, hablaremos del meta viewport y de las @media queries de CSS3. Finalmente, veremos cómo poner en marcha estas técnicas con algunos ejemplos en los que nuestra página pasará de estar en columnas a estar en cascada o cambiaremos el aspecto de nuestra barra de navegación en función del espacio disponible que tengamos.

### 3. APIs HTML5

En este momento, javascript es tan potente como cualquier otro lenguaje de desarrollo y, por la misma razón que lenguajes de programación profesionales poseen librerías para crear elementos gráficos, motores 3D para videojuegos o interfaces para acceder a bases de datos, javascript cuenta con APIs para ayudar a los programadores a lidiar con actividades complejas.

HTML5 introduce varias APIs (interfaces de programación de aplicaciones) para proveer acceso a poderosas librerías desde código javascript. El potencial de estas incorporaciones es muy importante, pero, desgraciadamente, no disponemos de tiempo suficiente para estudiarlas en profundidad.

A continuación, veremos rápidamente sus características para obtener una perspectiva de las posibilidades que nos ofrecen y, posteriormente, para que sirva como ejemplo, estudiaremos algo más en detalle una de ellas.

#### Características de las nuevas APIs

Las que vamos a ver a continuación no son las únicas APIs que se han incorporado al lenguaje javascript con el estándar HTML5, pero si son las más utilizadas y las que mayor número de navegadores soportan.

#### Drag & Drop

Drag & Drop incorpora la posibilidad de arrastrar y soltar elementos en la pantalla como lo haríamos comúnmente en aplicaciones de escritorio. Ahora, con unas pocas líneas de código, podemos hacer que un elemento esté disponible para ser arrastrado y soltado dentro de otro elemento en la pantalla. Estos elementos no sólo pueden incluir gráficos, también textos,

enlaces, archivos o datos en general.

### Geolocation

Geolocation es utilizada para establecer la ubicación física del dispositivo usado para acceder a la aplicación. Existen varios métodos para acceder a esta información, desde señales de red hasta el Sistema de Posicionamiento Global (GPS). Los valores retornados incluyen latitud y longitud, posibilitando la integración de esta API con otras como Google Maps, por ejemplo, o acceder a información de localización específica para la construcción de aplicaciones prácticas que trabajen en tiempo real.

### Storage

Se crearon dos APIs para el almacenamiento de datos: Web Storage e **Indexed Database**. Básicamente, estas APIs transfieren la responsabilidad del almacenamiento de datos del servidor al ordenador del usuario, pero, en el caso de Web Storage y su atributo **sessionStorage**, esta incorporación también incrementa el nivel de control y la eficiencia de las aplicaciones web.

Web Storage contiene dos importantes atributos que, en ocasiones, son considerados APIs por sí mismos: **sessionStorage** y **localStorage**.

El atributo **sessionStorage** es el responsable de mantener la consistencia de la aplicación durante la sesión de una página web y preservar información temporal como el contenido de un carro de compras, asegurando los datos en caso de accidente o mal uso (cuando la aplicación es abierta en una segunda ventana, por ejemplo).

Por otro lado, el atributo **localStorage** nos permite grabar contenidos extensos de información en el ordenador del usuario. La información

almacenada es persistente y no expira, excepto por razones de seguridad. Ambos atributos, `sessionStorage` y `localStorage`, reemplazan la anterior función del sistema de cookies y fueron creados para superar sus limitaciones.

La segunda API, agrupada dentro de las APIs de almacenamiento, pero independiente del resto, es **Indexed Database**. La función elemental de un sistema de base de datos es la de almacenar información indexada. Web Storage API trabaja sobre el almacenamiento de grandes o pequeñas cantidades de información, datos temporales o permanentes, pero no datos estructurados. Esta es una posibilidad disponible sólo para sistemas de base de datos y la razón de la existencia de esta API.

Indexed Database es una sustitución de la API Web SQL Database. Debido a desacuerdos acerca del estándar apropiado a utilizar, ninguna de estas dos APIs ha sido completamente adoptada. De hecho, en la actualidad, el desarrollo de Web SQL Database API (la cual había sido recibida con los brazos abiertos al comienzo), ha sido cancelado.

La API Indexed Database, también conocida como **IndexedDB**, es más prometedora, ya que tiene el apoyo de Microsoft, los desarrolladores de Firefox y Google. Sin embargo, esta situación podría cambiar en el futuro cercano.

## File Access

HTML5 proporciona APIs muy potentes que permiten interactuar con datos binarios y con el sistema de archivos local del usuario. Las API de archivos permiten que las aplicaciones web realicen tareas como leer archivos de forma síncrona o asíncrona, crear objetos BLOB arbitrarios, escribir

---

archivos en una ubicación temporal, leer un directorio de archivos de forma recurrente, arrastrar archivos del escritorio y soltarlos en el navegador y subir datos binarios con XMLHttpRequest2 (Versión 2 del objeto XMLHttpRequest de Ajax).

Las API de archivos se pueden utilizar, por ejemplo, para crear una vista previa en miniatura de las imágenes que se envían al servidor o para permitir que una aplicación guarde un archivo de referencia mientras el usuario no está conectado.

Con la API de audio web, la aplicación podría leer un archivo .mp3 y mostrar una visualización de la canción mientras se reproduce. También se podría utilizar lógica de cliente para verificar si el tipo MIME de un archivo subido coincide con su extensión o para limitar el tamaño de una subida.

## Communication

Algunas APIs tienen un denominador común que nos permite agruparlas juntas. Este es el caso de XMLHttpRequest Level 2, Cross Document Messaging, y Web Sockets. Internet ha estado siempre relacionado con comunicaciones, por supuesto, pero algunos asuntos no resueltos hacían el proceso complicado y en ocasiones imposible. Tres problemas específicos fueron abordados en HTML5:

1. La API utilizada para la creación de aplicaciones Ajax no estaba completa y era complicada de implementar a través de distintos navegadores.
2. La comunicación entre aplicaciones no relacionadas no existía.
3. No había forma de establecer una comunicación bidireccional efectiva para acceder a información en el servidor en tiempo real.

El primer problema fue resuelto con el desarrollo de XMLHttpRequest Level 2. El objeto **XMLHttpRequest** ha sido usado durante mucho tiempo para crear aplicaciones Ajax (scripts que acceden al servidor sin recargar la página web). El nivel 2 de esta API incorpora nuevos eventos, provee más funcionalidad (con eventos que permiten hacer un seguimiento del proceso), portabilidad (la API es ahora estándar), y accesibilidad (usando solicitudes cruzadas, desde un dominio a otro).

La solución para el segundo problema fue la creación de Cross Document Messaging. Esta API ayuda a los desarrolladores a superar las limitaciones existentes para comunicar diferentes cuadros y ventanas entre sí. Ahora esta API nos ofrece una comunicación segura a través de diferentes aplicaciones utilizando mensajes.

La solución para el último de los problemas listados anteriormente es Web Sockets. Su propósito es proveer las herramientas necesarias para la creación de aplicaciones de red que trabajan en tiempo real (por ejemplo, salas de chat). La API les permite a las aplicaciones obtener y enviar información al servidor en períodos cortos de tiempo, haciendo posible las aplicaciones en tiempo real para la web.

### Web Workers

Esta es una API única que expande Javascript a un nuevo nivel. Este lenguaje no es un lenguaje multitarea, lo que significa que no puede hacerse cargo de dos o más tareas simultáneamente. Web Workers provee la posibilidad de procesar código en segundo plano (ejecutado aparte del resto), sin interferir con la actividad en la página web y del código principal. Gracias a esta API javascript, ahora podemos ejecutar múltiples tareas al mismo tiempo.

---



## History

Ajax cambió la forma en la que los usuarios interactúan con sitios y aplicaciones web. Y los navegadores no estaban preparados para esta situación. History fue implementada para adaptar las aplicaciones modernas a la forma en que los navegadores hacen seguimiento de la actividad del usuario. Esta API incorpora técnicas para generar artificialmente URLs por cada paso en el proceso, ofreciendo la posibilidad de volver a estados previos de la aplicación utilizando procedimientos estándar de navegación.

## Offline

Incluso hoy día, con acceso a Internet en cada lugar que vamos, quedar desconectado es todavía posible. Dispositivos portátiles se encuentran en todas partes, pero no la señal para establecer comunicación. Y los ordenadores de escritorio también pueden dejarnos desconectados en los momentos más críticos. Con la combinación de atributos HTML, eventos controlados por javascript y archivos de texto, Offline permitirá a las aplicaciones trabajar en línea o desconectados, de acuerdo a la situación del usuario.

## API Web Storage

Una de las características más importantes en cualquier aplicación es la posibilidad de almacenar datos para disponer de ellos cuando sean necesarios. Antes de HTML5 no existía ningún mecanismo efectivo para este fin. Las cookies (archivos de texto almacenados en el ordenador del usuario) han sido utilizadas durante años para guardar información, pero, debido a su naturaleza, se limitaban a pequeñas cadenas de texto, lo que las hacía útiles sólo en determinadas circunstancias.

La API Web Storage es básicamente una mejora de las Cookies. Esta API nos permite almacenar datos en el disco duro del usuario y utilizarlos luego del mismo modo que lo haría una aplicación de escritorio. El proceso de almacenamiento provisto por esta API puede ser utilizado en dos situaciones particulares: cuando la información tiene que estar disponible sólo durante la sesión en uso y cuando tiene que ser preservada todo el tiempo que el usuario desee.

Para hacer estos métodos más claros y comprensibles para los desarrolladores, la API fue dividida en dos partes llamadas `sessionStorage` y `localStorage`.

🚦 **`sessionStorage`:** Este es un mecanismo de almacenamiento que mantendrá los datos disponibles únicamente durante la duración de la sesión de una página. De hecho, a diferencia de sesiones reales, la información almacenada a través de este mecanismo, sólo es accesible desde una única ventana o pestaña y es preservada hasta que la ventana se cierra. La especificación aún nombra "sesiones" debido a que la información es preservada, incluso cuando la ventana es actualizada o una nueva página desde el mismo sitio web es cargada.

🚦 **`localStorage`:** Este mecanismo trabaja de forma similar a un sistema de almacenamiento para aplicaciones de escritorio. Los datos son grabados de forma permanente y se encuentran siempre disponibles para la aplicación que los creó.

Ambos mecanismos trabajan a través de la misma interface, compartiendo los mismos métodos y propiedades. Y ambos son dependientes del origen, lo que quiere decir que la información está disponible únicamente a través del sitio web o la aplicación que los creó. Cada sitio web tendrá designado su propio

---

espacio de almacenamiento que durará hasta que la ventana se cierre o será permanente, de acuerdo al mecanismo utilizado. La API diferencia claramente datos temporales de permanentes, facilitando la construcción de pequeñas aplicaciones que necesitan preservar sólo unas cadenas de texto como referencia temporal (por ejemplo, carros de compra) o aplicaciones más grandes y complejas que necesitan almacenar documentos completos durante todo el tiempo que sea necesario.

### sessionStorage

Esta parte de la API, sustituye las cookies de sesión. Las Cookies, así como sessionStorage, mantienen los datos disponibles durante un período específico de tiempo, pero mientras las cookies de sesión usan el navegador como referencia, sessionStorage usa una simple ventana o pestaña. Esto significa que las Cookies creadas para una sesión estarán disponibles mientras el navegador continúe abierto, mientras que los datos creados con sessionStorage estarán disponibles mientras la ventana que los creó no se cierre.

Veamos un ejemplo:

```
<section id="cajaformulario">
  <form name="formulario">
    <p>Clave:<br><input type="text" name="clave" id="clave"></p>
    <p>Valor:<br><textarea name="text" id="texto"></textarea></p>
    <p><input type="button" name="grabar" id="grabar" value="Grabar"></p>
  </form>
</section>

<section id="cajadatos">
  <div>No hay información disponible</div>
</section>
```

Estilos:


```
#cajaformulario {  
  float: left;  
  padding: 20px;  
  border: 1px solid #999999;  
}
```

```
#cajadatos {  
  float: left;  
  width: 400px;  
  margin-left: 20px;  
  padding: 20px;  
  border: 1px solid #999999;  
}
```

```
#clave, #texto { width: 200px; }
```

```
#cajadatos > div {  
  padding: 5px;  
  border-bottom: 1px solid #999999;  
}
```

La página del ejemplo se mostrará como se ve en la siguiente imagen.



The image shows two side-by-side elements. On the left is a form with a light gray border. Inside the form, there are two labels: 'Clave:' followed by a text input field, and 'Valor:' followed by a larger text area. Below these is a button labeled 'Grabar'. On the right is a rectangular box with a light gray border containing the text 'No hay información disponible'.

Ambos, `sessionStorage` y `localStorage`, almacenan datos como ítems. Los ítems están formados por un par clave/valor, y cada valor será convertido en una cadena de texto antes de ser almacenado. Un ítem será similar a una

---

variable, con un nombre y un valor, que puede ser creado, modificado o eliminado.

Existen dos nuevos métodos específicos de esta API incluidos para crear y leer un valor en el espacio de almacenamiento:

✚ **setItem(clave, valor):** Este es el método que tenemos que llamar para crear un ítem. El ítem será creado con una clave y un valor de acuerdo a los parámetros especificados. Si ya existe un ítem con la misma clave, será actualizado al nuevo valor, por lo que este método puede utilizarse también para modificar datos previos.

✚ **getItem(clave):** Para obtener el valor de un ítem, debemos llamar a este método especificando la clave del ítem que queremos leer. La clave en este caso es la misma que declaramos cuando creamos al ítem con `setItem()`.

A continuación, veremos un ejemplo de uso:

```
function iniciar() {  
    var boton=document.getElementById('grabar');  
    boton.addEventListener('click', nuevoitem, false);  
}  
  
function nuevoitem() {  
    var clave=document.getElementById('clave').value; var  
    valor=document.getElementById('texto').value;  
    sessionStorage.setItem(clave,valor); mostrar(clave);  
}  
  
function mostrar(clave) {  
    var cajadatos=document.getElementById('cajadatos');  
    var valor=sessionStorage.getItem(clave);  
    cajadatos.innerHTML='<div>'+clave+' - '+valor+'</div>';  
}
```

---

```
window.addEventListener('load', iniciar, false);
```

En el ejemplo, la función `nuevoitem()` se ejecuta cada vez que el usuario hace clic en el botón del formulario. Esta función crea un ítem con la información insertada en los campos del formulario y luego llama a la función `mostrar()`. Esta última función lee el ítem de acuerdo a la clave recibida usando el método `getItem()` y muestra su valor en la pantalla utilizando la propiedad `innerHTML` que nos permite cambiar los contenidos de un elemento.

Además de estos métodos, la API también ofrece una sintaxis abreviada para crear y leer ítems desde el espacio de almacenamiento. Podemos usar la clave del ítem como una propiedad del objeto `sessionStorage`.

Tenemos dos posibilidades de sintaxis, además de la ya vista, para acceder al ítem:

- ✚ Encerrar la clave entre corchetes (por ejemplo, `sessionStorage[clave]=valor`).

- ✚ Usar directamente el nombre de la propiedad (por ejemplo, `sessionStorage.clave=valor`).

Utilizando esta sintaxis, el ejemplo anterior quedaría así:

```
function iniciar() {  
    var boton=document.getElementById('grabar');  
    boton.addEventListener('click', nuevoitem, false);  
}  
  
function nuevoitem() {  
    var clave=document.getElementById('clave').value; var  
    valor=document.getElementById('texto').value; sessionStorage.clave =  
    valor; mostrar(clave);  
}
```

```
function mostrar(clave) {  
    var cajadatos=document.getElementById('cajadatos');  
    var valor=sessionStorage.clave; cajadatos.innerHTML='<div>'+clave+' -  
    '+valor+'</div>';  
}
```

**window.addEventListener('load', iniciar, false);**

En el ejemplo sólo estamos leyendo el último ítem grabado. Vamos a mejorar este código aprovechando más métodos y propiedades provistos por la API con el propósito de manipular ítems:

🚦 **length:** Esta propiedad retorna el número de ítems guardados por esta aplicación en el espacio de almacenamiento. Trabaja exactamente como la propiedad length de un array.

🚦 **key(índice):** Los ítems son almacenados secuencialmente, enumerados con un índice automático que comienza por 0. Con este método podemos leer un ítem específico o crear un bucle para obtener toda la información almacenada.

Vamos a mejorar el ejemplo anterior de forma que no se muestre sólo el último ítem que se introdujo en el formulario, sino que se vaya mostrando una lista con todos los ítems que se hayan introducido.


```
function iniciar() {  
    var boton=document.getElementById('grabar');  
    boton.addEventListener('click', nuevoitem, false);  
    mostrar();  
}
```

```
function nuevoitem() {
    var clave=document.getElementById('clave').value; var
    valor=document.getElementById('texto').value;
    sessionStorage.setItem(clave,valor);
    mostrar();
    document.getElementById('clave').value='';
    document.getElementById('texto').value='';
}

function mostrar() {
    var cajadatos=document.getElementById('cajadatos');
    cajadatos.innerHTML='';
    for(var i=0;i<sessionStorage.length;i++) {
        var clave=sessionStorage.key(i);
        var valor=sessionStorage.getItem(clave);
        cajadatos.innerHTML+<div>'+clave+' - '+valor+'</div>';
    }
}

window.addEventListener('load', iniciar, false);
```

Ahora, si actualizamos la página del navegador, vemos que se siguen manteniendo todos los ítems que se han introducido anteriormente, ya que, en la función `iniciar()` llamamos a la función `mostrar()`. Evidentemente, los ítems, además de ser creados y leídos, también pueden ser eliminados. La API ofrece dos métodos para este propósito:

 **removeItem(clave):** Este método eliminará un ítem individual. La clave para identificar el ítem es la misma declarada cuando el ítem fue creado con el método `setItem()`.

---



✚ **clear():** Este método vaciará el espacio de almacenamiento. Todos los ítems serán eliminados.

Vamos a ampliar la funcionalidad del ejemplo anterior, dotándolo de la posibilidad de eliminar ítems.

```
function iniciar() {
    var boton=document.getElementById('grabar');
    boton.addEventListener('click', nuevoitem, false); mostrar();
}

function nuevoitem() {
    var clave=document.getElementById('clave').value; var
    valor=document.getElementById('texto').value;
    sessionStorage.setItem(clave,valor); mostrar();
    document.getElementById('clave').value='';
    document.getElementById('texto').value='';
}

function mostrar() {
    var cajadatos=document.getElementById('cajadatos');
    cajadatos.innerHTML='<div><button onclick="eliminarTodo()">Eliminar
    Todo</button></div>';
    for(var f=0;f<sessionStorage.length;f++) {
        var clave=sessionStorage.key(f);
        var valor=sessionStorage.getItem(clave);
        cajadatos.innerHTML+='<div>'+clave+' - '+valor+'<br>';
        cajadatos.innerHTML+='<button
        onclick="eliminar(\''+clave+'\')">Eliminar</button></div>';
    }
}
```

```
function eliminar(clave) {  
    if(confirm('Está Seguro?')) {  
        sessionStorage.removeItem(clave);  
        mostrar();  
    }  
}  
  
function eliminarTodo() {  
    if(confirm('Está Seguro?')) {  
        sessionStorage.clear();  
        mostrar();  
    }  
}  
window.addEventListener('load', iniciar, false);
```

En este nuevo ejemplo, hemos cambiado la función `mostrar()` para incorporar el manejador de eventos `onclick` y llamar a las funciones que eliminarán un ítem individual o vaciarán el espacio de almacenamiento. La lista de ítems presentada en pantalla se construye de la misma manera que antes, pero esta vez se añade un botón "Eliminar" junto a cada ítem para poder eliminarlo. Además, también se añade un botón para eliminar todos los ítems de una sola vez.

Las funciones `eliminar()` y `eliminarTodo()` se encargan de eliminar el ítem seleccionado o limpiar el espacio de almacenamiento, respectivamente. Cada función llama a la función `mostrar()` al final para actualizar la lista de ítems en pantalla. En ocasiones, nos interesa comprobar si una determinada clave ha sido creada anteriormente, por ejemplo, para inicializar valores. Podemos hacer esto de una forma muy sencilla:

---

**if (!sessionStorage.record)**

**sessionStorage.record = 0;**

En el ejemplo, comprobamos si existe la clave record, y si no existe la creamos con el valor inicial 0.

Si reproducimos el ejemplo, veremos que después de introducir varios ítems si recargamos la página con F5, los ítems siguen apareciendo en la lista, pero si cerramos la ventana y volvemos a abrirla, los ítems habrán desaparecido. Si necesitamos que los ítems se mantengan después de haber cerrado la ventana, debemos utilizar **localStorage**, en lugar de **sessionStorage**.

## localStorage

Disponer de un sistema confiable para almacenar datos durante la sesión de una ventana puede ser extremadamente útil en algunas circunstancias, pero cuando intentamos simular poderosas aplicaciones de escritorio en la web, un sistema de almacenamiento temporal no es suficiente.

Para cubrir este aspecto, la API Web Storage ofrece un segundo sistema que reservará un espacio de almacenamiento para cada aplicación y mantendrá la información disponible permanentemente. Con **localStorage** podemos grabar largas cantidades de datos y dejar que el usuario decida si la información es útil y debe ser conservada o no.

El sistema usa la misma interface que **sessionStorage**, por lo tanto, los métodos y propiedades vistos para **sessionStorage** están también disponibles para **localStorage**. De esta forma, el último ejemplo del punto anterior cambiando los **sessionStorage** por **localStorage** quedaría así:

```
function iniciar() {
    var boton=document.getElementById('grabar');
    boton.addEventListener('click', nuevoitem, false);
    mostrar();
}

function nuevoitem() {
    var clave=document.getElementById('clave').value;
    var valor=document.getElementById('texto').value;
    localStorage.setItem(clave,valor);
    mostrar();
    document.getElementById('clave').value='';
    document.getElementById('texto').value='';
}

function mostrar() {
    var cajadatos=document.getElementById('cajadatos');
    cajadatos.innerHTML='<div><button onclick="eliminarTodo()">Eliminar
    Todo</button></div>';
    for(var f=0;f<localStorage.length;f++) {
        var clave=localStorage.key(f);
        var valor=localStorage.getItem(clave);
        cajadatos.innerHTML+='<div>'+clave+' - '+valor+'<br>';
        cajadatos.innerHTML+='<button
        onclick="eliminar(\''+clave+'\')">Eliminar</button></div>';
    }
}

function eliminar(clave) {
    if(confirm('Está Seguro?')) {
        localStorage.removeItem(clave);
        mostrar();
    }
}
```

```
}  
  
function eliminarTodo() {  
    if(confirm('Está Seguro?')) {  
        localStorage.clear();  
        mostrar();  
    }  
}  
  
window.addEventListener('load', iniciar, false);
```

Como se puede observar, simplemente, hemos reemplazado `sessionStorage` por `localStorage` en el código del ejemplo anterior. Ahora, cada ítem creado será conservado entre diferentes ejecuciones de la aplicación, incluso después de que todas las ventanas del navegador se cierren.

En el caso de `localStorage`, también podemos comprobar si una determinada clave existe antes de crearla, como hacíamos con `sessionStorage`.

```
if (!localStorage.record)
```

```
    localStorage.record = 0;
```

Llegados a este punto nos encontramos con un problema, para reproducirlo haremos lo siguiente:

- ✚ Abriremos la página que contiene nuestro ejemplo en el navegador. Introduciremos un nuevo ítem en la lista.
- ✚ Abriremos otra instancia de nuestra página en otra ventana o pestaña del navegador (observaremos que aparece el ítem que introdujimos anteriormente).
- ✚ Iremos a la primera ventana e introduciremos un nuevo ítem.
- ✚ Volveremos de nuevo a la segunda ventana y observaremos que, hasta

que no actualicemos la página, el nuevo ítem no aparecerá en la lista.

Debemos hacer que las ventanas se comuniquen entre sí y mantener la información actualizada en cada una de ellas. En respuesta a este problema, la especificación incluye el evento **storage**. Este evento será disparado por la ventana cada vez que un cambio ocurra en el espacio de almacenamiento. Puede ser usado para informar a cada ventana abierta con la misma aplicación que los datos han cambiado en el espacio de almacenamiento y que se debe hacer algo al respecto.

La solución al problema en el ejemplo anterior es muy simple, sólo tenemos que añadir la siguiente línea de código en la función `iniciar()`: `window.addEventListener("storage", mostrar, false);`

Ahora cada vez que en una ventana se produzca algún cambio en el espacio de almacenamiento, el resto de ventanas recibirán el evento `storage`, y como tenemos un manejador para este evento que ejecuta la función `mostrar()`, la información estará actualizada. Si después de hacer el cambio, volvemos a realizar los pasos anteriores, observamos que el problema que teníamos a desaparecido.

### Espacio de almacenamiento

La información almacenada por `localStorage` será permanente a menos que el usuario decida que ya no la necesita. Esto significa que el espacio físico en el disco duro ocupado por esta información crecerá cada vez que la aplicación sea usada. La especificación de HTML5 recomienda a los fabricantes de navegadores que reserven un mínimo de 5 megabytes para cada origen (cada sitio web o aplicación).

Esta es solo una recomendación que, probablemente, cambiará en los próximos años. Algunos navegadores consultan al usuario si expandir o no el espacio disponible cuando la aplicación lo necesita, pero debemos ser conscientes de esta limitación y tenerla en cuenta a la hora de desarrollar nuestras aplicaciones.

Algunos navegadores sólo trabajan de forma adecuada con esta API cuando la fuente es un servidor real.

## 4. Páginas adaptables (Responsive Web Design)

El diseño web adaptable o adaptativo (en inglés, Responsive Web Design) es una filosofía de diseño y desarrollo web que, mediante el uso de estructuras e imágenes fluidas, así como de **@media queries** en la hoja de estilo CSS, consigue adaptar el sitio web al entorno del usuario.



El diseñador y autor norteamericano Ethan Marcotte creó y difundió esta técnica a partir de una serie de artículos en "A List Apart" (<http://alistapart.com/article/responsive-web-design/>), una publicación en línea especializada en diseño y desarrollo web, idea que luego extendería en su libro Responsive Web Design.

### Origen

Tanto la idea como el propósito del diseño web adaptable fueron previamente discutidos y descritos por el consorcio W3C en julio de 2008 en su recomendación "Mobile Web Best Practices" bajo el subtítulo "One Web"<sup>1</sup>.

Dicha recomendación, aunque específica para dispositivos móviles, puntualiza que está hecha en el contexto de "One Web", y que, por lo tanto, no sólo engloba la experiencia de navegación en dispositivos móviles, sino también en dispositivos de mayor resolución de pantalla como dispositivos de sobremesa.

---

<sup>1</sup> El concepto de "One Web" hace referencia a la idea de construir una Web para todos (Web for All) y accesible desde cualquier tipo de dispositivo (Web on Everything).



Hoy en día, la variedad de dispositivos existentes en el mercado ha provocado que la información disponible no sea accesible desde todos los dispositivos, o bien es accesible, pero la experiencia de navegación es muy pobre.

## Ventajas

La principal ventaja que encontramos con el diseño web adaptable es que la web se visualizará correctamente en todos los dispositivos que usemos y se adaptará a los giros en dispositivos móviles.

Además, Google tiene en cuenta las páginas web que tienen diseños sensibles o adaptables, gracias a su Googlebot-Mobile. El personal de Google recomienda que se use el Responsive Design para crear páginas web que se adapten a todos los dispositivos con el uso de @media-queries. La alternativa sería usar distinto código según el agente del dispositivo, o incluso, distintas URLs. Google nos incita a usar este método porque, de esta forma, Googlebot no necesita analizar tanto contenido de la misma web, lo que le facilita la labor de asignar el tipo de contenido de la página y de esta forma mejorará el SEO de la web.

"Google can discover your content more efficiently as we wouldn't need to crawl a page with the different Googlebot user agents to retrieve and index all the content".

Otro factor, que normalmente no tenemos en cuenta a la hora de hacer un diseño, es la accesibilidad del mismo, que dificulta algunas veces el uso de una web a un disminuido visual. Con un diseño adaptable que aproveche el espacio de la página podemos ofrecer una página de calidad para este colectivo.

A continuación, veremos lo esencial para crear un diseño web adaptable: el meta- tag Viewport, las @media queries, etc.

## Viewport

Esta meta-etiqueta fue creada en principio por Apple para su móvil predilecto, pero se ha convertido en todo un estándar que es soportado por la mayoría de los dispositivos móviles (smartphones, tablets y gran parte de móviles de gama media y baja).

Su uso es totalmente necesario, ya que sino el navegador establece el ancho con el que prefiere visualizar una página, en lugar de usar el ancho del que dispone, es decir, si la pantalla de nuestro móvil tiene 400px y el navegador detecta que lo óptimo sería visualizarla con 700px, así lo hará, a no ser que usemos este meta).

Como siempre, el meta lo añadiremos en el <head> de nuestra página:

```
<meta name="viewport" content="width=device-width, initial-scale=1.0, user-scalable=no">
```

Se pueden usar los siguientes parámetros (separados por comas):

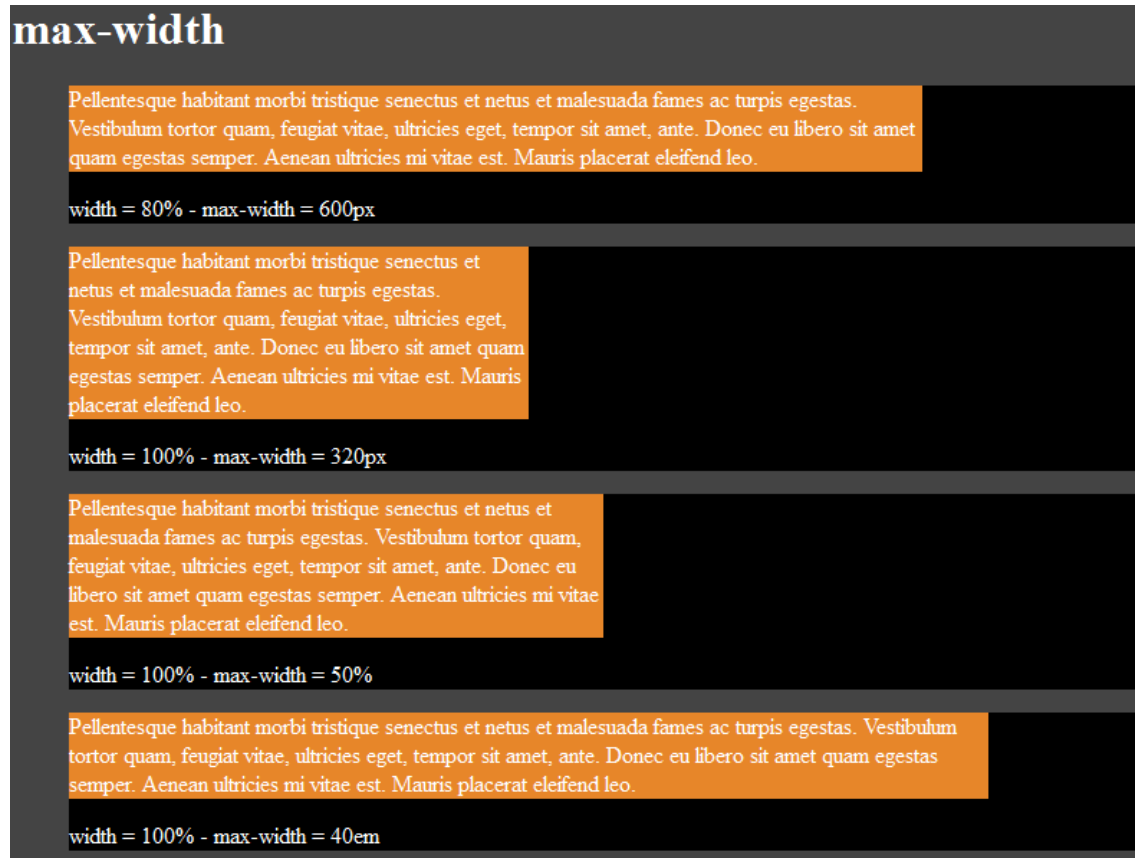
- ✚ **width:** ancho de la página (se puede establecer en píxeles o como device-width y usará el ancho del que dispone el dispositivo).
  - ✚ **height:** alto de la página, actúa igual que el width.
  - ✚ **initial-scale:** escala o zoom inicial de la página (este y los demás tipos de escala se establecen con valores como 1.0 para no tener zoom o 2.5 para tener un zoom de 2 aumentos y medio, por ejemplo).
  - ✚ **minimum-scale:** zoom mínimo que podemos hacer en la página.
  - ✚ **maximum-scale:** zoom máximo que podemos hacer en la página.
  - ✚ **user-scalable:** establece si está permitido o no hacer zoom (yes/no).
-

## Preparar nuestra página para el diseño adaptable

Para lograr que nuestra web se adapte a los anchos de pantalla, debemos tener en cuenta varias cosas importantes:

- ✚ La primera, y la más importante, es dejar de usar píxeles en todos los sitios, en su lugar, usaremos porcentajes (por ejemplo: `width: 60%`). Para saber qué porcentajes debemos de poner, haremos la siguiente operación: Supongamos que tenemos un elemento con un ancho de 960 píxeles y, dentro de este, tenemos otro elemento de 300 píxeles. Al elemento externo le pondremos, por ejemplo, un `width` del 90% para que se adapte al tamaño de la ventana y para que el elemento interno mantenga la proporción con respecto al externo calcularemos el ancho que debería tener así:  $300/960$ , lo cual, nos dará 0,3125. Por lo tanto, el `width` que le pondremos al elemento interno será 31,25%.
- ✚ Para limitar el ancho (o alto si se terciara) debemos de usar el parámetro `max-width`(`max-height` en el caso del alto). Para establecer el mínimo usaríamos `min-width` y `min-height`, aunque estos se utilizan menos. El valor de estas propiedades sobrescribirá el valor de las propiedades `width` y `height`. Por ejemplo, si indicamos `width: 100%` y `max-width: 600px`, cuando visualicemos la página con una resolución de 1200px, el ancho será de 600px, ya que es el valor máximo que hemos indicado para el ancho (puedes ver un ejemplo de uso en el fichero `max-width.html` de la carpeta de recursos). En este ejemplo, tenemos cuatro elementos configurados con distintas combinaciones de `width` y `max-width`. Si vamos cambiando el ancho de la ventana del navegador, veremos cómo los anchos se van adaptando hasta llegar a los anchos máximos configurados, momento en el cual, el ancho permanecerá

invariable. En la siguiente imagen podemos observar cómo se muestra el ejemplo.



- ✚ No debemos usar posiciones absolutas ni fijas (salvo contadas excepciones).
- ✚ Nunca debemos permitir que una imagen de fondo que está pensada para no repetirse, llegue a repetirse por el cambio de dimensiones. Para evitarlo, debemos adaptarla, normalmente, usando @media-queries.
- ✚ Tampoco debemos permitir que las imágenes y los vídeos se salgan de la estructura, sino aparecerá un scroll lateral en los dispositivos móviles que destruirá totalmente el diseño.

## @media queries

Desde los días de CSS 2.1, nuestras hojas de estilos han disfrutado de "una conciencia del dispositivo" a través de los media types. Por ejemplo cuando creamos una hoja de estilos para impresión, estamos utilizando este concepto:

```
<link rel="stylesheet" type="text/css" href="core.css" media="screen" />
```

```
<link rel="stylesheet" type="text/css" href="print.css" media="print" />
```

Esperando que utilizáramos este concepto para diseñar algo más que "bonitos formatos de páginas para imprimir", la especificación CSS nos suministró un grupo de media types aceptable, cada uno de ellos diseñado para una clase específica de dispositivo listo para la web. Pero la mayoría de los navegadores y los dispositivos no se han adherido al espíritu de la especificación, dejando varios media types implementados imperfectamente, o completamente ignorados.

Por suerte, el W3C creó las media queries como parte de la especificación CSS3, mejorando la promesa de los media types. Una media query nos permite apuntar, no sólo a ciertas clases de dispositivos, sino realmente inspeccionar las características físicas del dispositivo que está renderizando nuestro trabajo. Por ejemplo, siguiendo el reciente crecimiento de WebKit mobile, las media queries se han convertido en una popular técnica del lado del cliente para entregar una hoja de estilos a medida para el iPhone, los smartphones Android o los tablets. Para hacerlo, podemos incorporar una query al atributo media de una hoja de estilos linkada:

```
<link rel="stylesheet" type="text/css" media="screen and (max-device-width: 480px)" href="shetland.css" />
```

La query contiene dos componentes: Un media type (screen), y la consulta entre paréntesis, que contiene una característica a inspeccionar (max-device-width) seguida por el valor al que apuntamos (480px). En otras palabras, le estamos preguntando al dispositivo, si su resolución horizontal (max-device-width) es igual o menor que 480px. En el caso de que visualicemos la página en un dispositivo con una pantalla pequeña como el iPhone, entonces el dispositivo cargará shetland.css. De lo contrario, el link se ignora.

En el pasado, este tipo de cosas necesitaban de soluciones javascript, pero la especificación de media query provee una serie de características del medio que se extienden mucho más allá de la resolución de la pantalla, ampliando el alcance de lo que podemos testear con nuestras queries. Además, podemos testear múltiples valores de las propiedades en una sola query, encadenándolos con la palabra clave and:

```
<link rel="stylesheet" type="text/css" media="screen and (max-device-width: 480px) and (resolution: 163dpi)" href="shetland.css" />
```

Además del uso de la etiqueta <link> para incluir media queries, también tenemos la posibilidad de incluirlas en nuestro CSS como parte de una regla

**@media:**

```
@media all and (max-device-width: 480px) {  
  .column {  
    float: none;  
  }  
}
```

O como parte de una directiva @import:

```
@import url("shetland.css") all and (max-device-width: 480px);
```

En cada caso, el efecto es el mismo: si el dispositivo pasa el examen planteado por nuestra media query, el CSS que corresponda es aplicado a





---

nuestro código. La palabra `all` indica que la regla será para todos los tipos de medios y no para uno específico.

Con esto, disponemos de nuevas capacidades que nos permiten definir conjuntos de estilos dependiendo de propiedades comunes de los dispositivos que acceden a nuestros sitios. Propiedades como el alto y el ancho, la relación de aspecto o el número de colores disponible. Las reglas `@media` pueden ser utilizadas para adaptar nuestras páginas, no solo para dispositivos comunes, sino para todo tipo de dispositivos que nuestros lectores usen para visitarlas.

Aunque las `@media` queries nos permite conocer muchas propiedades diferentes, las más importantes son las siguientes:

`aspect-ratio`: Hace referencia a las dimensiones relativas del dispositivo expresadas como una relación de aspecto: 16:9 por ejemplo.




-  `Width` y `height`: Se refiere a las dimensiones del área de visualización. Además pueden ser expresadas en valores mínimos y máximos.
-  `orientation`: Detecta si el layout es panorámico (el ancho es mayor que el alto) o vertical (el alto es mayor que el ancho). Esto nos permite ajustar los diseños para dispositivos con propiedades de giro de la pantalla como el iPhone y otros smartphones o los tablets.
-  `resolution`: Indica la densidad de los pixeles en el dispositivo de salida. Esto es especialmente útil cuando queremos aprovecharnos de las ventajas de los dispositivos que tiene una resolución mayor a 72 dpi.
-  `color`, `color-index` y `monochrome`: Detectan el número de colores o bits por color. Esto nos permite crear diseños específicos para dispositivos monocromáticos.

## Puesta en marcha

Ya hemos visto lo básico que necesitamos conocer sobre el Responsive Design, pero, la mayoría de las veces, lo difícil es aplicarlo de la forma que necesitamos o mostrar el contenido de una forma eficaz sin perder en diseño ni usabilidad, y en esto es en lo que nos centraremos ahora, en estructurar correctamente el contenido de una página que se adapte a distintos tipos de pantalla.

Para empezar veremos cómo hacer fluir elementos como imágenes, bloques de texto o cualquier cosa en general, de forma que se reubiquen los elementos en función del tamaño disponible en la pantalla. En primer lugar, nos olvidaremos de los float y usaremos en su lugar la propiedad display.

La propiedad display de CSS nos permite establecer cómo se comportará un elemento respecto a los demás. Nos interesan, principalmente, tres valores posibles:

-  **block:** los elementos se ubican uno debajo del otro.
-  **inline:** los elementos se ubican uno junto al otro, pero sin respetar las propiedades de margen.
-  **inline-block:** esta última forma de visualizarse un elemento es una mezcla de las dos anteriores, comportándose como un bloque, pero poniéndose en línea con otros elementos iguales.

Una de las cosas que se dan mucho en los diseños adaptativos es crear un conjunto de bloques con imágenes y texto que, según el tamaño de la ventana, se van recolocando.

Por ejemplo, en la web de la imagen se ven seis bloques, tres encima y tres debajo, pero si los visualizáramos con un ancho de pantalla mayor se verían en línea.



## 2012 | 2013 Season



Atlanta Ballet's Nutcracker  
December 7-26, 2012



Dracula  
February 8-16, 2013



Cinderella  
January 4-6 & February 16-17, 2013



New Choreographic Voices  
March 22-24, 2013



Carmina Burana  
April 12-14, 2013



Love Stories  
May 10-12, 2013

Veamos cómo podríamos crear una estructura como esa:

```
<div id="contenedor">
```

```
  <figure class="bloque">[Texto/Imágenes]</figure>
```

```
  <figure class="bloque">[Texto/Imágenes]</figure>
```

```
  <figure class="bloque">[Texto/Imágenes]</figure>
```

```
  <figure class="bloque">[Texto/Imágenes]</figure>
```

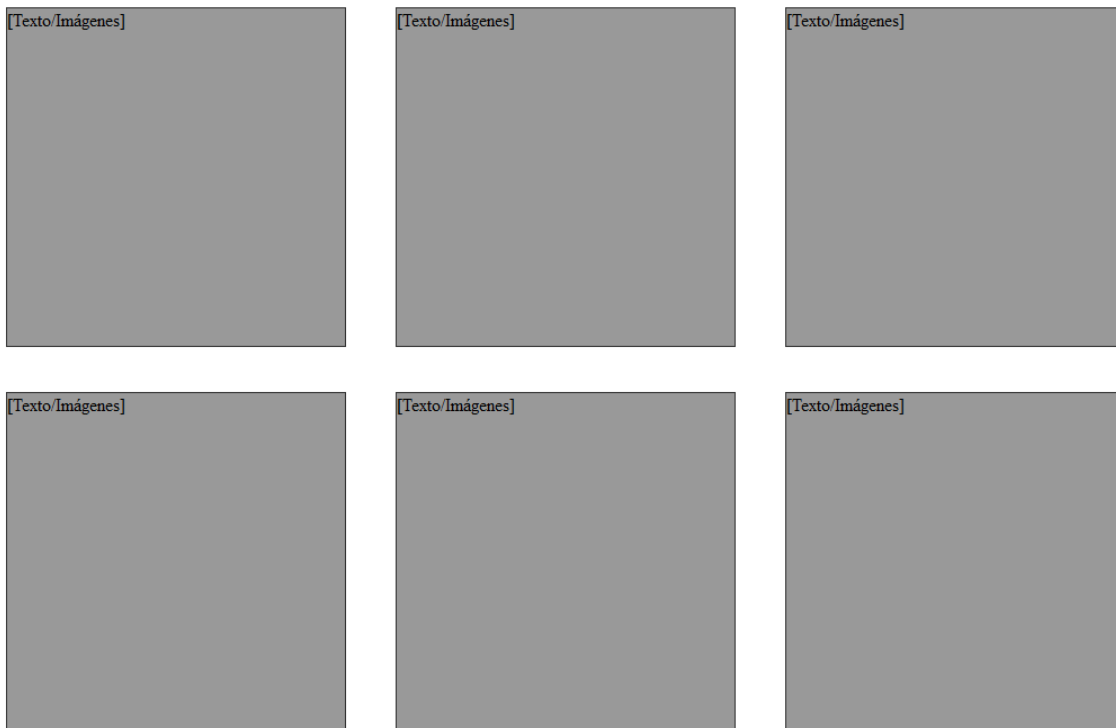
```
  <figure class="bloque">[Texto/Imágenes]</figure>
```

```
  <figure class="bloque">[Texto/Imágenes]</figure>
```

```
</div>
```

En la hoja de estilos, simplemente, estableceremos el modo de visualización `display:inline-block` para el elemento `figure`, y cuando los elementos no quepan en el contenedor se reajustarán sin perder su tamaño.

```
#contenedor .bloque {  
    display: inline-block;  
    height:300px;  
    width: 300px;  
    border:1px solid #333;  
    background: #999;  
    margin:20px;  
}
```



Podéis ver este ejemplo en la carpeta **bloques1** del fichero de recursos. Si probáis a ampliar y reducir el tamaño de la ventana de vuestro navegador, veréis que los elementos se van reubicando automáticamente gracias a la propiedad display. Muy bien, pero ¿qué ocurre si queremos que los elementos se comporten de una forma diferente cuando el tamaño de ventana disponible sea inferior a un valor determinado? En ese caso, utilizaremos las media queries. Por ejemplo, si quisiéramos que a partir de los 800 píxeles los bloques anteriores se mostraran unos debajo de otros, centrados y ajustando su ancho

al espacio disponible, haremos lo siguiente:

```
@media all and (max-width: 800px) {  
  #contenedor .bloque {  
    /* Cuando el ancho sea inferior a 800px el  
    elemento será un bloque */  
    display: block !important;  
    width: auto !important;  
  }  
}  
  
#contenedor .bloque {  
  display: inline-block;  
  height:300px;  
  width: 300px;  
  border:1px solid #333;  
  background: #999;  
  margin:20px;  
}
```



Ahora, utilizamos la media query cuando la resolución es inferior a 800 píxeles (max-width: 800px) aplicando los estilos:

```
#contenedor .bloque {  
  display: block !important;  
  width: auto !important;  
}
```

Indicamos que los elementos se comporten como bloques, es decir, que se ubiquen uno debajo de otro (display: block) y, para que se adapten al espacio disponible, indicamos el ancho automático (width: auto). Al poner ancho automático se ubicará el elemento dejando 20 píxeles de margen a cada lado (margin:20px) y ocupará el resto del espacio.

Podéis ver este ejemplo en la carpeta bloques2 del fichero de recursos. Es importante observar que cuando se cumple la condición indicada en la media query, en primer lugar, se aplican los estilos que hemos indicado fuera de ella:

```
#contenedor .bloque {  
    display: inline-block;  
    height:300px;  
    width: 300px;  
    border:1px solid #333;  
    background: #999;  
    margin:20px;  
}
```

y, posteriormente, se aplicarán los estilos de dentro:

```
#contenedor .bloque {  
    display: block !important;  
    width: auto !important;  
}
```

por lo tanto, el resultado será que el elemento recibirá los estilos que se indican en ambos sitios:

```
#contenedor .bloque {  
    display: inline-block;  
    display: block !important;  
    height:300px;  
    width: 300px;  
    width: auto !important;  
    border:1px solid #333;  
    background: #999;  
    margin:20px;  
}
```

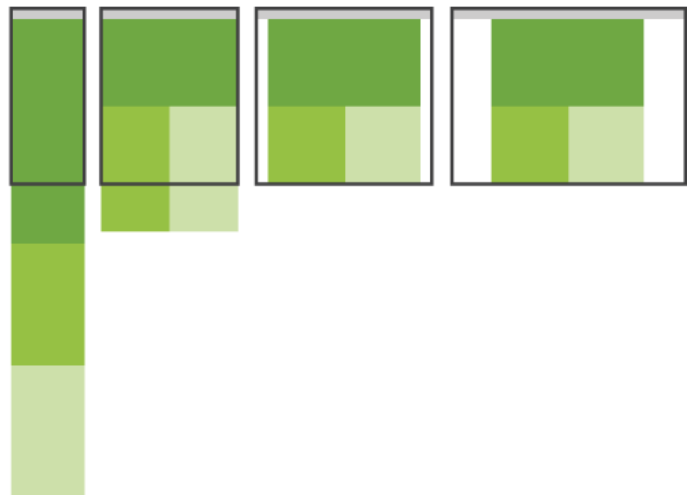
Como vemos, la propiedad display se aplica dos veces (display: inline-block; display: block !important;) y lo mismo ocurre con la propiedad width

---

¿cómo sabrá el navegador cuál de los dos aplicar? en estos casos, lo mejor que podemos hacer es indicar la palabra `!important` en aquellas propiedades que queramos que tengan preferencia sobre el resto. De esta forma, nos aseguramos que el navegador aplica la propiedad que nos interesa. Sólo es necesario que lo indiquemos en aquellas propiedades que se repitan en la media query y fuera de ella, en las que no se repiten, no será necesario.

### Pasar de columnas a cascada

Uno de los métodos que se utiliza más a menudo para cambiar la estructura de una web adaptativa, que suele tener una cabecera y dos o tres columnas (con el menú, el contenido, enlaces, etc.) es el paso de la disposición en columnas a la disposición en cascada.



En la imagen se puede ver cómo, en función del ancho disponible se cambia la disposición. En la siguiente url (<http://www.hsgac.senate.gov/>) podéis ver un ejemplo que utiliza esta técnica.

A continuación, veremos cómo hacer una estructura adaptable similar a la de la imagen, que estaría formada por un título con un menú, una cabecera y dos columnas, la primera con una agenda con eventos y la segunda columna con los contenidos.

Lo que haremos para que sea totalmente adaptativo es realizar pequeños cambios según las necesidades, pero, para el ejemplo, sólo tendremos dos

estados: la estructura bien maquetada y la estructura en cascada, como se representa en la imagen anterior.

Nuestro html será el siguiente:

```
<nav>[Menú]</nav>
<div class="pagina">
  <header>[Cabecera]</header>
  <aside>[Agenda]</aside>
  <main>[Contenido]</main>
</div>
```

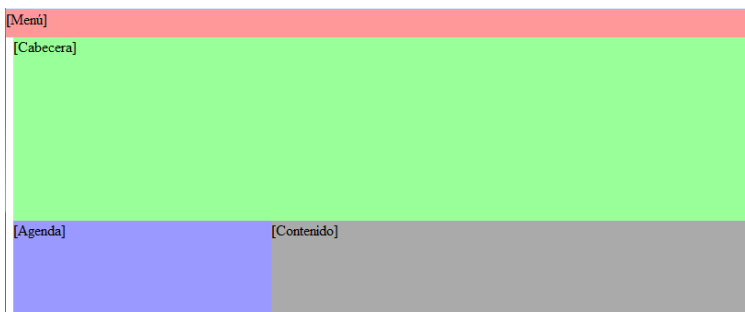
Y aplicaremos los siguientes estilos:

```
@media all and (max-width: 600px) {
  div, nav, header, aside, main {
    /* Cuando el ancho sea inferior a 600px el elemento será un bloque */
    display: block !important;
    /* Se ajustarán al ancho de la ventana */
    width: 100% !important;
    margin: auto !important;
    /* La posición será estática (flujo normal del documento) */
    position: static !important;
    /* los elementos dejarán de ser flotantes */
    float: none !important;
  }
}

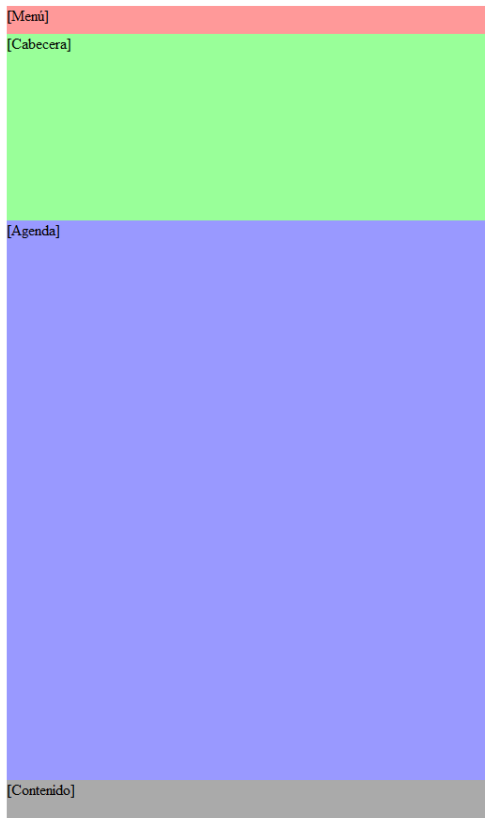
nav {
  background: #F99;
  position: fixed;
  top: 0px;
  left: 0px;
  width: 100%;
  height: 30px;
}
```

```
.pagina {  
  /* la página será como máximo de 1000px de ancho */  
  max-width: 1000px;  
  margin:auto;  
}  
  
header {  
  background: #9F9;  
  margin-top: 30px;  
  height: 200px;  
}  
  
aside {  
  background: #99F;  
  float: left;  
  /* flotamos el elemento a la izquierda */  
  width: 35%;  
  height: 600px;  
}  
  
main {  
  background: #AAA;  
  width: 65%;  
  margin-left: 35%;  
  height: 900px;  
}
```

La página se mostrará así para resoluciones de más de 600 píxeles:



y así para resoluciones menores:



Podéis ver este ejemplo en la carpeta cascada del fichero de recursos. Para probar el ejemplo, simplemente, tenéis que cargar la página en el navegador y reducir la ventana hasta que tenga una resolución inferior a 600 píxeles.

### Adaptar imágenes y vídeos

Muchas veces, para crear diseños muy optimizados puede ser importante usar varias imágenes distintas, una con un tamaño superior y otra con uno más pequeño, pero vamos a centrarnos en el uso de una misma imagen que, en la medida de lo posible, debería de estar optimizada y se adaptará como nosotros queramos.

Podemos hacer que una imagen se comporte de las siguientes formas:



- ✚ Ocupar el ancho de la página. Puede ser útil para cabeceras o imágenes principales y lo haremos estableciendo su ancho al 100%.

```
img { width:100%; }
```

- ✚ Tener una imagen con un tamaño máximo. En este caso, la imagen tendrá un tamaño que no sobrepasará, pero cuando se disminuya el tamaño del contenedor se encogerá adaptándose a la página.

```
img { width:100%; max-width:400px; }
```

- ✚ Tener las imágenes con su tamaño original como máximo. Este caso es muy similar al anterior, con la diferencia que pondremos como tamaño máximo el ancho de la imagen. De esta forma, la imagen no se deformará al ser redimensionada a un tamaño superior al original.

```
img{width:100%; max-width:100%;}
```

Para adaptar el tamaño de los vídeos usaremos las mismas técnicas que utilizamos para las imágenes.

Una cosa a tener en cuenta, es que si hemos puesto un alto fijo a un vídeo, y después, en una @media query, ponemos, por ejemplo, un ancho relativo utilizando porcentajes, el elemento <video> se redimensionará para mantener las proporciones al redimensionar la ventana, pero el espacio reservado para el alto inicial se mantendrá reservado, por lo que los elementos que se encuentren alrededor no se reubicarán. Para solucionar este problema, bastará con indicar en la @media query que el alto será automático (height:auto).

## La barra de navegación

En las versiones para menores resoluciones o tamaños de pantalla el menú suele convertirse hoy en día, casi un estándar de facto, en un icono con

tres rayitas que se despliega. Pasamos de tener un menú lineal y visible a un menú desplegable. A continuación, veremos cómo podemos implementar este menú basándonos en un ejemplo. Tenemos el siguiente código html:

```
<nav>
  <a href="#">Menú</a>
  <ul>
    <li><a href="#">Home</a></li>
    <li><a href="#">About</a></li>
    <li><a href="#">Collection</a></li>
    <li><a href="#">Blog</a></li>
    <li><a href="#">Contact</a></li>
    <li><a href="#">Directions</a></li>
  </ul>
</nav>
<script src="menu.js"></script>
```

A este código le aplicaremos los siguientes estilos:

```
* { margin: 0; padding: 0; }
```

```
nav {
  width:100%;
  background-color: #fff;
  border-bottom: 1px solid #ccc;
}
```

```
nav ul {
  list-style: none;
  padding: 0px;
  margin: 0px;
  font-weight: bold;
  text-align: center;
}
```

```
nav ul li {
    display: inline-block;
    text-align: left;
}

nav ul li a {
    display: block;
    padding: 15px 10px;
    text-decoration: none;
    color: #444;
}

nav ul li a:hover {
    background-color: #ccc;
}

nav > a {
    display: none;
}
```

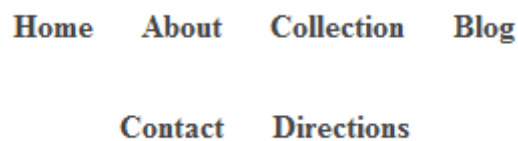
Con estos estilos nuestra barra de navegación quedará como se muestra en la siguiente imagen:



Home About Collection Blog Contact Directions

---

Se puede observar que el enlace con el texto Menú queda oculto (display:none). Si reducimos la ventana del navegador, los enlaces de la barra se reubicarán de la siguiente forma:



Home About Collection Blog

Contact Directions

---

lo cual, podría no ser deseable. Para mejorarlo, vamos a hacer que cuando el tamaño de la ventana sea menor de 475 píxeles, el menú se oculte y se muestre sólo el enlace con el texto Menú. Después, al hacer click con el ratón sobre este enlace el menú se desplegará y al volver a pulsar se plegará.

Añadiremos la siguiente @media query:

```
@media all and (max-width: 475px) {  
  nav ul { display: none; } /* ocultamos el menú de navegación */  
  nav > a /* mostramos el enlace con el texto Menú */ {  
    display: block;  
    padding: 0 1em 0;  
    text-align: center;  
    padding: 10px 15px;  
    color: #fff;  
    background-color: #0084B4;  
    text-decoration: none;  
    margin: 3px;  
  }  
  /* Con la clase desplegado el menú se mostrará verticalmente */  
  ul.desplegado {  
    display: block;  
    list-style: none;  
  }  
  ul.desplegado li {  
    display: block;  
    text-align: center;  
  }  
  ul.desplegado li a {  
    display: block;  
    border-bottom: 1px solid #ccc;  
  }  
}
```

Con esta @media query cuando reducimos la ventana el menú se mostrará así:



Ya sólo nos queda hacer que al pulsar el enlace el menú se despliegue. Para ello, utilizaremos el siguiente código javascript:

```
var enlaceMenu;
function iniciarMenu() {
    enlaceMenu = document.querySelector("nav>a");
    enlaceMenu.addEventListener("click", despliegaMenu, false);
}

function despliegaMenu() {
    document.querySelector("nav>ul").classList.toggle('desplegado');
}

window.addEventListener("load", iniciarMenu, false);
```

En el código anterior, al pulsar el enlace menú se le aplicará la clase desplegado al <ul> si no la tiene aplicada o se eliminará dicha clase si ya la tiene aplicada. Esto lo hacemos con el método `classList.toggle('desplegado')`. Con esto, al pulsar sobre el enlace se desplegará el menú y quedará así:

Menú
Home
About
Collection
Blog
Contact
Directions

Podéis ver el ejemplo completamente funcional en la carpeta nav del fichero de recursos.

## 5. Ejercicios

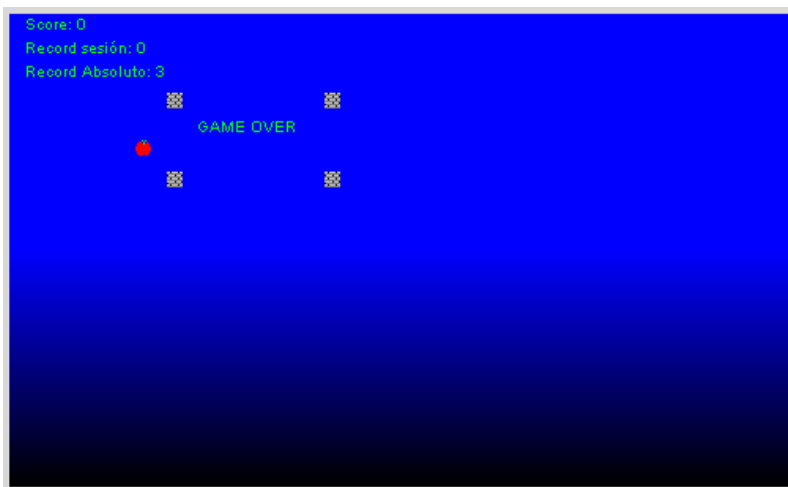
### Ejercicio 1

Hemos visto en el punto 3 del tema las nuevas APIs que incorpora HTML5 y, en concreto, hemos hablado sobre la API Web Storage. En este ejercicio, vamos a añadir una nueva funcionalidad a nuestro videojuego del tema anterior para trabajar con esta nueva API.

La nueva funcionalidad que vamos a añadir es el registro de las puntuaciones máximas obtenidas en el juego. Tendremos dos tipos de puntuaciones máximas:

- ✚ La puntuación máxima de la sesión actual, es decir, la mayor puntuación que se haya conseguido desde que cargamos el juego en la ventana del navegador.
- ✚ La puntuación máxima de todos los tiempos, es decir, la mayor puntuación que se haya conseguida en un navegador en todas las ejecuciones del juego.

En la ventana del juego se mostrarán, además de la puntuación actual (Score), la puntuación máxima de la sesión (Record sesión) y la puntuación máxima de absoluta (Record Absoluto).

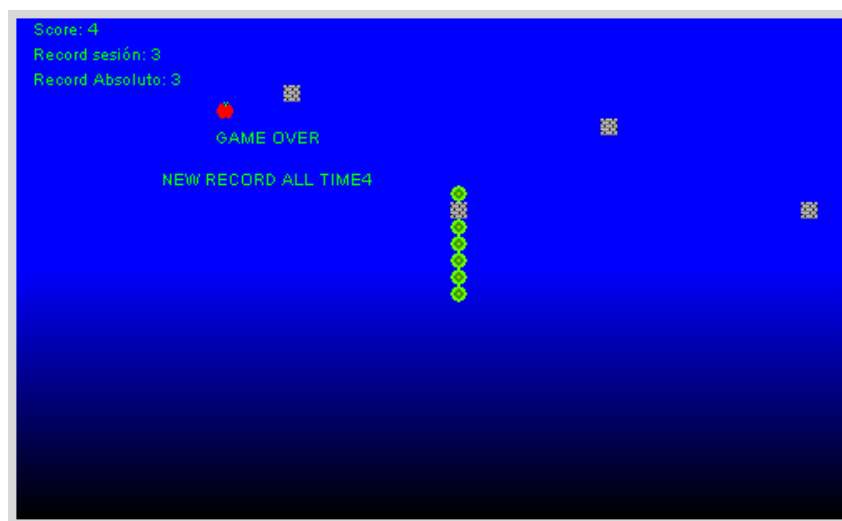


En la primera ejecución del juego, ambos records estarán a 0 y, a medida que vayamos jugando, se irán actualizando los records. Si cerramos el navegador y volvemos a abrirlo, la puntuación máxima de la sesión se reiniciará, pero la absoluta se mantendrá.

Al terminar la partida, si se ha superado el record de la sesión, se mostrará el mensaje "NEW RECORD" seguido de la puntuación obtenida.



Si se supera el record absoluto, se mostrará el mensaje "NEW RECORD ALL TIME" seguido de la puntuación obtenida.





## Ejercicio 2

Vamos a modificar nuestro sitio web para convertirlo en un sitio web adaptable.

Comenzaremos por añadir el meta viewport a todos los documentos: index.html, videos.html, contacto.html y serpiente.html.

```
<meta name="viewport" content="width=device-width, initial-scale=1.0, user-scalable=yes">
```

## Ejercicio 3

En este ejercicio vamos a preparar nuestra página index.html para que se adapte al tamaño de la ventana.

Trataremos de que nuestra página se vea correctamente en cualquier resolución de pantalla, con lo que conseguiremos que se vea bien en todos los dispositivos, ordenadores, portátiles, tablets y/o móviles. Para ello, crearemos tres @media queries:

🚦 La primera con un ancho máximo de 1024 píxeles. En esta @media query haremos los siguientes cambios:

- El <div id="page"> tendrá un ancho del 90% y un ancho mínimo de 320 píxeles.
- El <header> tendrá un tamaño de fuente de 0.7em.
- Los <li> del menú existente en la barra de navegación (#page>nav>ul>li) tendrán los siguientes márgenes externos: 0% el superior, 8% el izquierdo y el derecho y 2% el inferior.

🚦 La segunda con un ancho máximo de 700 píxeles. En esta @media query haremos los siguientes cambios:

- Los <li> del menú existente en la barra de navegación

(#page>nav>ul>li) tendrán los siguientes márgenes externos: 0% el superior, 4% el izquierdo y el derecho y 1% el inferior.

🚦 La tercera con un ancho máximo de 475 píxeles. En esta @media query prepararemos nuestra barra de navegación para que se oculte y se pueda desplegar como hemos visto en el tema. Antes de nada prepararemos nuestro documento:

- Añadiremos un enlace vacío dentro del <nav>, justo antes del <ul>:  
`<a href="#"></a>`
- Enlazaremos el fichero de script menu.js que podéis encontrar en la carpeta scripts del fichero de recursos.  
`<script src="scripts/menu.js"></script>`
- En el fichero de estilos principal (estilos.css) ocultaremos (display:none) el enlace que hemos introducido anteriormente en el <nav> para que, inicialmente, este oculto y sólo aparezca cuando el ancho de la ventana sea inferior a 475 píxeles.

A continuación, crearemos la @media query siguiendo los pasos que se han explicado durante el tema. Tenemos que tener en cuenta que:

- Los márgenes de los <li> de la barra de navegación estarán a 0.
- El enlace vacío que hemos introducido tendrá las siguientes características:
  - Tendrá la imagen de fondo menu.png que podéis encontrar en la carpeta imgs del fichero de recursos.
  - La imagen no se repetirá y estará centrada y a 10 píxeles de distancia del borde superior.
  - El ancho y alto del enlace será de 80px.
  - El enlace tendrá unos márgenes externos superior e inferior

de 0 y el izquierdo y derecho serán automáticos.

- Debes crear la clase `.desplegado`, tal y como hemos visto en el tema.

Estas @media queries las crearemos en el fichero `estilos.css`. De esta forma, se aplicarán a todos los documentos de nuestra web.

### Ejercicio 4

En este ejercicio vamos a convertir en adaptable la página `videos.html`. Para ello, haremos los siguientes cambios en el fichero `reproductor.css`:

✚ En el elemento `#reproductor`:

- Eliminaremos la flotación y lo mostraremos como bloque en línea (`inline-block`).
- El ancho será del 95% con un ancho máximo de 720 píxeles.
- Tendrá unos márgenes externos superior e inferior de 20 píxeles y el izquierdo y derecho serán automáticos.

✚ Crearemos una regla de estilo para el elemento `<video>`:

- El ancho será del 95% con un ancho máximo de 720 píxeles.
- El alto será automático.
- Los márgenes externos serán de 5 píxeles.

✚ En la barra de navegación del reproductor:

- Eliminaremos la flotación y lo mostraremos como bloque en línea (`inline-block`).
- Eliminaremos el ancho.
- La alineación vertical será arriba (`vertical-align:top`).

✚ En el elemento `#barra`:

- El ancho será del 95% con un ancho máximo de 705 píxeles.
- Eliminaremos el margen interno.

Finalmente, crearemos una @media query para un tamaño máximo de 970 píxeles. En esta @media query, simplemente, indicaremos que los botones de la barra de navegación del reproductor se muestren como bloques en línea (inline-block).

Una vez finalizado el ejercicio nos encontraremos con un problema, y es que, si recordamos, en el tema 3 iniciábamos la variable máximo en la función iniciar y le dábamos un valor de 700. Esta variable la utilizábamos para calcular el ancho de la barra de progreso del vídeo en las funciones redimensionaBarra y desplazarMedio. Ahora el ancho de esta barra no es fijo y dependerá del tamaño de la ventana, por lo que, en lugar de iniciar la variable máximo al principio, lo que haremos será calcular el ancho cada vez que lo necesitemos.

Las funciones redimensionaBarra y desplazarMedio quedarán así:

```
function redimensionaBarra() {
    if(!medio.ended) {
        var maximo=parseInt(getStyle('barra', "width"));
        var total=parseInt(medio.currentTime*maximo / medio.duration);
        progreso.style.width=total+'px';
    }
    else {
        progreso.style.width='0px';
        play.value='\u25BA';
        window.clearInterval(bucle);
    }
}

function desplazarMedio(e) {
```

---

```
        if(!medio.paused && !medio.ended) {  
            var ratonX=e.pageX-barra.offsetLeft;  
            var maximo=parseInt(getStyle('barra', "width"));  
            var nuevoTiempo=ratonX*medio.duration/maximo;  
            medio.currentTime=nuevoTiempo;  
            progreso.style.width=ratonX+'px';  
        }  
    }  
}
```

Sólo nos falta la función `getStyle`. Esta función es necesaria porque estamos obteniendo el valor de la propiedad de estilo `width` que no se inicia con un valor fijo, sino que tiene un valor relativo al tamaño del elemento que la contiene (recordad que hemos puesto un ancho del 95%). Para obtener el valor del ancho de la barra en un momento determinado, tenemos que llamar al método `window.getComputedStyle`. Este método nos devolverá el valor calculado de una propiedad en un momento determinado. Con todo esto, crearemos la función `getStyle` de la siguiente forma:

```
function getStyle(nombreElemento, nombrePropiedad) {  
    var elemento = document.getElementById(nombreElemento);  
    return  
    window.getComputedStyle(elemento,null).getPropertyValue(nombrePropiedad);  
}
```

## Ejercicio 5

En este ejercicio convertiremos en adaptable la página `contacto.html`. Para ello, haremos los siguientes cambios en el fichero `contacto.css`:

🌈 Crearemos una `@media` query para un tamaño máximo de 1024 píxeles en

la que modificaremos las siguientes reglas:

- El elemento `#contenedorFormulario`:
  - Márgenes externos del 4%.
  - Ancho del 58%.
  - Márgenes internos superior e inferior del 2% e izquierdo y derecho del 4%.
- El elemento `<h1>` del `#contenedorFormulario`:
  - Posición estática.
  - Márgenes internos superior e inferior del 0% e izquierdo y derecho del 1%.
  - Márgenes externos superior e inferior del 2% e izquierdo y derecho del 0%.
- Los input que no sean submit:
  - Alto del 5%.
  - Ancho del 55%.
- Los input que no sean submit cuando tienen el foco:
  - Ancho del 61%.
- El elemento `<textarea>`:
  - Ancho del 75%.
- El submit:
  - Se mostrará como un elemento en bloque (`display:block`).

## Ejercicio 6

---

Para terminar, convertiremos en adaptable la página `serpiente.html`. Para ello, haremos los siguientes cambios en el fichero `videojuego.css`:

- ✚ Crearemos una `@media` query para un tamaño máximo de 1024 píxeles en la que modificaremos las siguientes reglas:
  - El elemento `#contenedorVideojuego`:
    - Márgenes externos del 5%.
    - Márgenes internos superior e inferior del 2% e izquierdo y derecho del 5%.
    - Ancho del 80%.
  - El elemento `<canvas>`:
    - Ancho del 80%.

Con esto habremos terminado los ejercicios del tema. En el aula virtual podéis encontrar un vídeo de demostración en el que podéis ver cómo debería quedar nuestra página web después de finalizar los ejercicios. Recordad que para realizar la entrega del ejercicio, debéis comprimir la carpeta curso dentro de un fichero llamado:

`nombre-alumno-tema6.zip`

No olvidéis pasar el validador tanto a los `html` como a los `css` antes de efectuar la entrega.