CAPÍTULO 13.- MAS SOBRE ASPECTOS VISUALES Y ORGANIZATIVOS DE LA WEB.

MÁS SOBRE LA PROPIEDAD CSS DISPLAY

Hemos utilizado anteriormente la propiedad **CSS** “**display**” con alguno de sus valores. Ahora que ya sabemos mucho de **HTML** y **CSS** podemos profundizar más en ella para entender mejor su funcionamiento, conocer todo su alcance, su potencial para el diseño y el posicionamiento de los diferentes elementos en nuestra web y todos sus valores.

En **HTML** todos los elementos son una caja que contiene otros elementos y sobre los que podemos aplicar unos estilos **CSS**. En apartados anteriores aprendimos como calcular la dimensión total de una caja teniendo en cuenta su anchura, su “**padding**” y su “**margin**” y como colocarla en la web indicando que se comportara como un elemento en línea, en bloque o como las celdas de una tabla.

Pues bien, la propiedad “**display**” define el **tipo de caja** que genera un elemento **HTML,** el posicionamiento y espacio que ocupa y el comportamiento con respecto a los elementos adyacentes. Su sintaxis es:

**display: inline / block / list-item / run-in / inline-block / table / inline-table / table-row-group**

**/ table-header-group / table-footer-group / table-row / table-column-group**

**/ table-clomn / table-cell / table-caption / flex / inline-flex / none;**

A tener en cuenta:

* **Display: block**; Indica que el elemento se comporte como un bloque. En este caso:
  + El elemento no permitirá la ubicación de elementos adyacentes.
  + Si no se especifican sus dimensiones, el elemento ocupará todo el ancho disponible dentro de su contenedor. Su altura se determinará por su contenido.
  + Si se especifica un ancho, el elemento ocupará solo la extensión determinada pero sin permitir elementos adyacentes.
* **Display: inline**; Indica que el elemento **HTML** se comporte como un elemento en línea. En este caso:
  + El elemento permitirá la ubicación de elementos adyacentes.
  + No se podrá especificar las dimensiones del elemento, ya que como elementos en línea, sus dimensiones son determinadas por el contenido.
* **Display: none**; Indica al elemento **HTML** que no genere ninguna caja con lo que el elemento simplemente desaparece igual que si estuviera ausente del código fuente. En este caso:
  + El elemento no ocupa ningún espacio.
  + Los elementos adyacentes ignoran su presencia y ocupan su espacio.
  + Sus elementos descendientes, en caso de que los tenga, también desaparecen.
* **Display: inline-block**; Indica al elemento **HTML** que se comporte como un elemento en bloque pero permitiendo elementos adyacentes como si fuera un elemento en línea. En este caso:
  + Se les puede asignar las propiedades de los elementos de texto.
  + Se puede determinar sus dimensiones como los elementos en bloque.
  + Si no se especifica su anchura, ocupan todo el espacio disponible como los elementos en bloque.
  + La posición que ocupan estos elementos, es similar a los elementos en línea, lo que quiere decir que la parte inferior del elemento se apoyará en la base de la línea.
* **Display: list-item**; Indica al elemento **HTML** que se comporte como un ítem de lista, es decir, que se comporte exactamente como si se tratara de un elemento <**li**>. En este caso:
  + No permite la presencia de elementos adyacentes como cualquier <**li**>.
  + Pueden especificarse todas sus dimensiones.
  + A este elemento se le pueden aplicar todas las propiedades **CSS** de listas que ya conocemos: “**list-style-type**”, “**list-style-image**”, etc, etc…
* **Display: run-in**; Indica al elemento **HTML** que se comporte de forma diferente dependiendo de los elementos adyacentes. Cuando un elemento cuya propiedad “**display**” tiene el valor “**run-in**” va seguido de un elemento cuya propiedad “**display**” tiene como valor “**block**”, el elemento se comporta como si el valor fuera “**inline**” y se ubica **dentro** del elemento con “**display: block**”. Para todos los demás casos, el elemento marcado con este valor **se comporta como un elemento de bloque**.

Este valor, en la actualidad, solo es reconocido por algunos navegadores como **Ópera** y **Chrome**. Quizás sea por esta circunstancia el que no sea muy utilizado.

* **Display: table**; Indica al elemento **HTML** que se comporte como una tabla.
* **Display: table-row**; Indica al elemento **HTML** que se comporte como una fila de una tabla.
* **Display: table-row-group**; Indica al elemento **HTML** que se comporte como un grupo de filas de una tabla.
* **Display: table-cell**; Indica al elemento **HTML** que se comporte como las celdas de una tabla.
* **Display: Inline-table**; Indica al elemento **HTML** que se comporte como una tabla pero que se mantenga como un elemento en línea, sin salto de línea anterior ni posterior.
* **Display: table-column**; Indica al elemento **HTML** que se comporte como una columna de una tabla.
* **Display: table-column-group**; Indica al elemento **HTML** que se comporte como un grupo de columnas de una tabla.
* **Display: table-header-group**; Indica al elemento **HTML** que se comporte como un grupo de filas de encabezado de tabla.
* **Display: table-footer-group**; Indica al elemento **HTML** que se comporte como un grupo de filas de pie de tabla.
* **Display: table-caption**; Indica al elemento **HTML** que se comporte como si se tratara del título de una tabla.

Todos estos valores de la propiedad “**display**” nos permiten manejar distintos elementos de nuestra web y conseguir que se coloquen como si de una tabla se tratara. Volvemos a recordar aquí que la recomendación es usar las tablas de **HTML** con el propósito de mostrar datos tabulados a semejanza de una hoja de cálculo. Mientras que estos valores de la propiedad “**display**” se utilizarán para mejorar la presentación de los distintos elementos dentro de la página.

* **Display: flex**; Crea un elemento **HTML** flexbox de bloque. Profundizaremos más en este modelo de caja proximamente.
* **Display: Inline-flex**; Crea un elemento **HTML** flexbox en línea. Profundizaremos más en este modelo de caja proximamente.

Para ilustrar el funcionamiento de esta potente propiedad practicaremos sobre una web sencilla que nos permita distinguir sus diferentes valores. ***Para ello realice la práctica 13.1. y la primera ejercitación del capítulo 13.***

COMPATIBILIDAD DE LOS NAVEGADORES

Hasta ahora hemos venido trabajando con el navegador Firefox. Pero este no es el único naveagador que pueden utilizar los visitantes de nuestra web. ¿No se ha preguntado cómo se ven en otros navegadores las páginas web que hemos ido creando en nuestras prácticas?

Aunque hemos hablado brevemente sobre ello a lo largo del curso, en este apartado vamos a profundizar sobre la compatibilidad de los diferentes navegadores con la normativa **HTML5** y **CSS3**. Esta tecnología punta sólo se tiene en cuenta en las últimas versiones de los navegadores. Por tanto, un problema con el que se encuentra el diseñador web, es que no todos los navegadores soportan con la misma eficacia las especificaciones de la **W3C** y debe tener en cuenta esta realidad al crear sus webs.

En la actualidad un diseñador web se puede encontrar, entre otros, con los siguientes navegadores:

* **Internet Explorer**. Aplica **HTML5** y **CSS3** a partir de su versión 9. Pero con una implementación muy pobre. Además, un problema añadido es que un buen número de internautas aún utiliza versiones mucho más antiguas, como Internet Explorer 6, que para nada soportan estas nuevas especificaciones. Utiliza el motor de renderizado “**Trident**”.
* **Firefox**. Es uno de los navegadores preferidos por los diseñadores web por sus extensiones que facilitan el desarrollo de aplicaciones web. Es además uno de los que más han integrado **HTML5** y **CSS3**. Utiliza el motor de renderizado “**Gecko**”. (A la hora de realizar este manual ha anunciado que adoptará, en colaboración con la empresa Samsung, un nuevo motor de renderizado llamado “**Servo**”.)
* **Chrome**. Es ya el tercer navegador más usado en la red. Es moderno, dinámico y soporta prácticamente todas las especificaciones **HTML5** y **CSS3**. Utiliza el motor de renderizado “**Webkit**”. (A la hora de realizar este manual Chrome ha anunciado que abandona el motor de renderizado “**Webkit**” y adoptará uno nuevo llamado “**Blink**”.)
* **Safari**. Es el navegador por defecto de los equipos Mac aunque también está disponible para Windows. Está a la última en lo que respecta a la integración de **HTML5** y **CSS3**. También utiliza el motor de renderizado “**Webkit**”.
* **Opera**. Es poco utilizado pero es el más innovador y soporta en su totalidad todas las especificaciones **HTML5** y **CSS3**. En un principio utilizaba el motor de renderizado “**Presto**” pero posteriormente se cambió a “Webkit”. (A la hora de realizar este manual también ha anunciado que se pasará a “**Blink**”.)
* **IOS Safari**. Es el navegador móvil más compatible con **HTML5** y **CSS3**. Utiliza el motor de renderizado “**Webkit**”.
* **Opera Mini**. Se presenta en dos versiones, Opera Mini y Opera Mobile. Es probablemente uno de los navegadores para móviles más utilizado en el mundo. Como su hermano “mayor” soporta **HTML5** y **CSS3** en su totalidad. Utiliza el motor de renderizado “**Presto**”.
* **Android Browser**. Es bastante compatible con **HTML5** y **CSS3**. Utiliza el motor de renderizado “**Webkit**”.

Algunos de estos navegadores se adelantan a las nuevas funcionalidades definidas por el **W3C** y para que puedan entender las nuevas etiquetas **HTML5** o propiedades **CSS3** necesitan de los llamados prefijos privativos. Así lo hemos hecho en el caso de aplicar los degradagos en la propiedad “**Background**” en los que aplicábamos el prefijo “**-moz**” a los valores del degradado. Esto hace que sólo reconozcan correctamente esta propiedad **CSS** aquellos navegadores cuyo motor de renderizado sea “**Gecko**” pero no otros***. ¡Veámoslo realizando la práctica 13.2!***

Para navegadores **webkit** la sintaxis es:

**background: -webkit-gradient(linear, origen, fin, from(color), to(color));**

A tener en cuenta:

* **Linear**. Indica un degradado lineal.
* **Origen**. Es el punto inicial del degradado. Está determinado por dos valores separados por un espacio. Se puede indicar números, porcentajes o las palabras claves **top**, **right**, **bottom** o **left**. Por ejemplo: **left top**.
* **Fin**. Es el punto final del degradado. Está determinado por dos valores separados por un espacio. Se puede indicar números, porcentajes o las palabras claves **top**, **right**, **bottom** o **left**. Por ejemplo: **right top**.
* **From(color)**. Indica el color de inicio del degradado.
* **To(color)**. Indica el color de fin del degradado.

Para Internet Explorer la sintaxis es parecida a la de Firefox pero con el prefijo “**-ms**”:

En el caso de los degradados circulares, la sintaxis para “**Webkit**” es:

**background: -webkit-gradient(radial, origen, radio1, fin, radio2, from(color), to(color));**

A tener en cuenta:

* **Radial**. Indica un degradado circular.
* **Origen**. Es el punto inicial del degradado, en este caso del círculo interior. Está determinado por dos valores separados por un espacio. Se puede indicar números, porcentajes o las palabras claves **top**, **right**, **bottom**, **left** y **center**. Por ejemplo: **center center**.
* **Radio1**. Es el radio del círculo interior.
* **Fin**. Es el punto final del degradado. Está determinado por dos valores separados por un espacio. Se puede indicar números, porcentajes o las palabras claves **top**, **right**, **bottom**, **left** y **center**.
* **Radio2**. Es el radio del círuclo exterior.
* **From(color)**. Indica el color de inicio del degradado, es decir del círculo interior.
* **To(color)**. Indica el color de fin del degradado, es decir del círculo exterior.

Y para Internet Explorer la sintaxis del degradado radial es igual que la de Firefox sólo que cambiando el prefijo “**-moz**” por el prefijo “**-ms**”.

Y por último, ¿Cómo será la sintaxis de los degradados para el navegador Opera? Pues si, ha acertado. Igual que para Firefox pero con el prefijo “**-o**”.

***Para terminar de dominar el tema realice ahora la 2ª ejercitación del capítulo 13.***

SOPORTE PARA ELEMENTOS HTML5 EN NAVEGADORES ANTIGUOS

En apartados anteriores de este tema hemos vislumbrado otro problema para los desarrolladores web más modernos… ¿Qué pasa con mis webs creadas con **HTML5** y **CSS3** cuando se visualizan en navegadores antiguos, como el caso que hemos comentado de Internet Explorer 6, que no entienden estas implementaciones?

Cuando un navegador se encuentra con un elemento desconocido simplemente lo ignora y no lo muestra o lo muestra de una manera diferente, por ejemplo, considerando un elemento en bloque como un elemento en línea. Esto significaría que nuestra página web desarrollada en **HTML5** se mostraría de una forma totalmente diferente y deformada.

Una forma de conseguir que los navegadores más antiguos apliquen adecuadamente los estilos **CSS3** y los elementos **HTML5** es utilizar la librería JavaScript **MODERNIZR**. Se trata de una utilidad pensada para detectar funcionalidades dentro de cualquier navegador y añadirles el soporte necesario para poder utilizar los nuevos objetos y estilos si no están soportados.

Para añadir la librería **MODERNIZR** a una web basta con añadir el siguiente código en el <**head**> de la página justo debajo de la línea que llama a la hoja de estilos **CSS** externa:

**<script src=”modernizr.js” type=”text/javascript”></script>**

Para más información y para ver si existen versiones más recientes de **MODERNIZR** visite su web: [**www.modernizr.com**](http://www.modernizr.com).

Desde la página web anterior, se puede descargar esta librería que tiene dos versiones, una completa, llamada versión de desarrollo que contiene todas las características de **HTML5** y **CSS3** y otra, llamada versión de producción donde se puede elegir la característica a compatibilizar.

***Detengámonos brevemente para comprobar el funcionamiento de esta librería realizando la práctica 13.3.***

Para tener una visión completa y actualizada de la compatibilidad de los diferentes navegadores con las recomendaciones **HTML5** y **CSS3** de la **W3C** puede visitar la web: [**www.caniuse.com**](http://www.caniuse.com).

Otra utilidad que le invitamos a conocer es **BOILERPLATE**, (**html5boilerplate.com**). Se trata de una plantilla que pretende simplificar los comienzos de un proyecto web incluyendo el código necesario optimizado para solucionar, entre otros, estos problemas de compatibilidad.

ORGANIZACIÓN DE UN SITIO WEB

Como hemos podido comprobar a medida que hemos ido avanzando en el curso hemos ido completando un ejemplo de sitio web dedicado al turismo en España. Un sitio web se compone de varias páginas web que a su vez contienen varios archivos de estilos, de imágenes, etc, etc.

Hasta ahora hemos ido guardando todos los archivos necesarios para nuestro sitio en una sola carpeta pero las buenas prácticas indican que se deben ordenar en varias carpetas.

Cada creador puede utilizar las carpetas que le parezcan oportunas. Nosotros proponemos, para nuestro ejemplo, la siguiente estructura:

* Todos los archivos “**.htm**” los dejamos en la carpeta principal en este caso su carpeta de prácticas.
* Los archivos para descargar, tenemos un archivo “**.pdf**” que utilizamos al principio, los colocaríamos en la carpeta “**descargas**”.
* Los archivos de estilos “.css” los almacenaríamos en la carpeta “**estilos**”.
* Los archivos correspondientes a tipos de letra los almacenaríamos en la carpeta “**fuentes**”.
* Los archivos de imágines los almacenaríamos en la carpeta “**imágenes**”.
* Las librerías los almacenaríamos en la carpeta “**librerias**”.

Hay que tener en cuenta que en todos y cada uno en los enlaces de nuestro sitio web hay que indicar la ruta correcta a cada archivo siguiendo los ejemplos “**href=estilos/reset.css**” o “**imagenes/turismovacaciones.jpg**”.

***Realice ahora la tercera ejercitación del tema 13 para dejar totalmente dominado este apartado.***

EL MODELO DE CAJA FLEXBOX

Eje

Principal

Eje Transversal

***Elementos distribuidos horizontalmente***

***Elementos distribuidos verticalmente***

Eje Principal

Eje

Transversal

Al finalizar el apartado correspondiente a la propiedad “**display**”, en este mismo capítulo, anunciábamos que explicaríamos más adelante sus valores “**flex**” e “**inline-flex**”. ¡Vamos a ello!

Hasta ahora los diferentes elementos se mostraban en el navegador siguiendo el orden en que aparecían en el documento **HTML**. Este orden lo alterábamos un poquito utilizando, entre otras, las propiedades **CSS** “**float**”, “**clear**” y “**position**”. Pues bien, el modelo flexible de caja, modifica radicalmente este planteamiento permitiendo colocar los diferentes elementos de un documento **HTML** alterando el orden de forma dinámica, apareciendo en el navegador, antes o después, con independencia de cómo estén en el archivo del código fuente. También permite adaptar el tamaño para rellenar el espacio disponible en cualquier dispositivo de visualización. Una caja flexible expande sus elementos para llenar el espacio libre disponible, o los reduce para que no haya desbordamientos.

Este modelo de caja es tan novedoso que, a la hora de escribir este manual, aún hay pocos navegadores que lo implementen, pero es de esperar que en el futuro se utilice másivamente. Además la notación aquí utilizada no es la recomendación definitiva del **W3C** aunque no se preveen cambios significativos. No obstante recomendamos que se esté atento por si hubiera algún cambio.

Para entender bien el funcionamiento del modelo de caja flexible debemos conocer su terminología:

* **Contenedor flexible o Flex container**. Es la caja contenedora o padre de los elementos flexibles. La mayoría de las propiedades FlexBox se aplican al contenedor flexible o principal.
* **Flex ítems**. Son los hijos del flex container y sobre los que se aplicará el modelo de caja flexible.
* **Ejes**. En cada flex container se definen dos ejes, (Ver gráfico en página anterior):
  + El **eje principal** a lo largo del cual se ordenan los elementos flexibles que puede ser horizontal, en filas, o vertical, en columnas.
  + El **eje transversal** que es el eje perpendicular al eje principal.

Veremos que una propiedad **CSS** define la dirección del eje principal, en línea o en columna y otras propiedades nos permitirán alinear los flex ítems teniendo en cuenta el eje principal y/o el eje transversal. También podremos cambiar el sentido de aparición de los elementos flexibles.

* **Líneas.** Los ítems pueden ser distribuidos en una sóla línea o en varias dentro del contenedor.

***Veamos un ejemplo realizando la práctica 13.4.***

La sintaxis de esta propiedad es:

**-webkit-flex-direction: row / row-reverse / column / column-reverse;**

A tener en cuenta:

* **Row**. Indica que el eje principal sea el horizontal y la dirección de izquierda a derecha. Como hemos visto, es el valor que se aplica por defecto si no utilizamos esta propiedad.
* **Row-reverse**. Indica que el eje principal sea el horizontal y la dirección de derecha a izquierda.
* **Column**. Indica que el eje principal sea el vertical y la dirección de arriba abajo.
* **Column-reverse**. Indica que el eje principal sea el vertical y la dirección de abajo arriba.

***Realice la práctica 13.5.***

La propiedad “**flex-wrap**” define si los flex ítems se colocan dentro de su contenedor en una o varias líneas. Su sintáxis es:

**-webkit-flex-wrap: nowrap / wrap / wrap-reverse;**

A tener en cuenta:

* **Nowrap**. Indica que los elementos flexibles no se coloquen en varias líneas. Tal como acabamos de comprobar en chrome, es el valor que se aplica por defecto si no utilizamos esta propiedad.
* **Wrap**. Indica que los elementos flexibles se coloquen en varias líneas.
* **Wrap-reverse**. Indica que los elementos flexibles se coloquen en varias líneas en orden inverso.

***¡Veámoslo en la práctica 13.6!***

Existen varias propiedades que nos permiten alinear los elementos flexibles siguiendo el eje principal y/o el eje transversal. ¡Vamos a ver una a una!

La propiedad “**align-items**” alínea los hijos FlexBox a lo largo del eje transversal. Su sintaxis es:

**-webkit-align-items: flex-start / baseline / flex-end / center / stretch;**

A tener en cuenta:

* **Flex-start o baseline**. Indica que el inicio de todos los elementos flexibles esté a ras del inicio del eje transversal.
* **Flex-end**: Indica que el final de todos los elementos flexibles esté a ras de la parte inferior del eje transversal.
* **Center**. Indica que el centro de todos los elementos flexibles esté a ras de la parte central del eje transversal.
* **Stretch**. Indica que todos los elementos flexibles se expandan para llenar toda la longitud del eje transversal. Esta es la opción por defecto si no utilizamos esta propiedad. Por lo tanto, el resultado de este valor, es lo que en estos momentos nos muestra chrome. Muy importante caer en la cuenta de que, utilizando este valor, todos los elementos flexibles ocupan toda la altura de su contenedor padre, sin importar su contenido y sin importar que cambie la anchura y la altura por cambios en el tamaño de la ventana, lo que facilita enormemente el diseño.

***Práctiquelo realizando la práctica 13.7.***

La propiedad “**justify-content**” especifica cómo se disponen los elementos sobre el eje principal, en término de lo que sucede con el espacio blanco sobrante entre hijos. Su sintaxis es:

**-webkit-justify-content: flex-start / flex-end / center / space-between / space-around;**

A tener en cuenta:

* **Flex-start**. Agrupa todos los hijos FlexBox al principio del eje principal, con todo el espacio al final.
* **Flex-end**. Agrupa todos los hijos FlexBox al final del eje principal, con el espacio todo al principio.
* **Center**. Agrupa todos los hijos FlexBox en el centro del eje principal, con el espacio repartido por igual a cada extremo.
* **Space-around**. Reparte todo el espacio entre los elementos hijos y en el exterior de todos ellos en cantidades iguales.
* **Space-between**. Tiene un efecto similar a “**space-around**”, pero sin espacio asignado a cada extremo de la serie de hijos FlexBox, es decir, los hijos de ambos extremos se ajustan a cada lado del contendor y se reparte el espacio exterior que hay entre los elementos hijos centrales.

***¡Veámoslo en la práctica 13.8!***

La propiedad “**align-content**” define como se alinean las líneas formadas por los elementos flexibles dentro del contenedor flexible cuando hay espacio adicional en el eje transversal. Sólo en el caso de que estos se distribuyan en varias líneas, es decir, tengan el valor “**wrap**” en la propiedad “**flex-wrap**” o “**flex-flow**”, (como sucede en el ejemplo que estamos empleando para nuestra práctica). Su sintaxis es:

**-webkit-align-content: flex-start / flex-end / center / space-between / space-around / stretch;**

A tener en cuenta:

* **Flex-start**. Agrupa todos los hijos FlexBox al principio del eje transversal, con todo el espacio al final.
* **Flex-end**. Agrupa todos los hijos FlexBox al final del eje transversal, con tod el espacio al principio.
* **Center**. Agrupa todos los hijos FlexBox en el centro del eje transversal, con el espacio repartido por igual a cada extremo.
* **Space-around**. Reparte todo el espacio entre los elementos hijos y en el exterior de todos ellos en cantidades iguales.
* **Space-between**. Tiene un efecto similar a “**space-around**”, pero sin espacio asignado a cada extremo de la serie de hijos FlexBox, es decir, los hijos de ambos extremos se ajustan a cada lado del contendor y se reparte el espacio exterior que hay entre los elementos hijos centrales.
* **Stretch**. Es el valor por defecto. Muestra los elementos flexibles alineados por el eje transversal.

***¡Veámoslo en la práctica 13.9!***

La propiedad “**order**” especifica en que orden se dibujan los elementos flexibles en su contenedor con independencia de cómo aparezcan en el **HTML**. Su sintaxis es:

**-webkit-order: nº entero;**

A tener en cuenta:

* Los elementos se disponen en orden ascendente según su valor de “**order**”. Cuanto mayor sea el valor más tarde aparecerá el elemento hijo.
* Por defecto, todos los hijos FlexBox tienen el valor “order” 0.
* Los elementos con el mismo valor en “**order**” se presentan en el orden en que aparecen en el código fuente.
* Esta propiedad admite números negativos.

La propiedad “**flex-grow**” establece el factor de crecimiento flex. Se asigna a los elementos flexibles e indica cuánto crecerá este elemento en relación a los demás cuando haya espacio libre.

La propiedad “**flex-shrink**” establece el factor de encogimiento flex. Se asigna a los elementos flexibles e indica cuanto decrecerá un elemento respecto a los demás cuando no haya espacio libre.

La propiedad “**flex-basis**” estalece el ancho base inicial del elemento flex. Acepta los mismos valores que la propiedad “**width**” y su funcionamiento es similar, salvo en caso de asignar el valor “**auto**”.

La propiedad “**flex**” es la forma acortada de “**flex-grow**”, “**flex-shrink**” y “**flex-basis**”.

***¡Veámoslo en funcionamiento realizando la práctica 13.10!***

MAPAS DE IMÁGENES

Anteriormente aprendimos a enlazar a otra página pulsando sobre una imagen. Un mapa de imágenes en **HTML** es una imagen permite enlazar a varias páginas diferentes en función de la parte de la imagen donde se haga el clic del ratón.

Aunque el uso de mapas de imágenes es una práctica cada vez más en desuso en las webs acutales por los problemas que plantean a la hora del posicionamiento en los motores de búsqueda y de visibilidad en las web móviles, no deja de ser una utilidad más de **HTML** y por eso lo estudiaremos al menos brevemente.

Existen varias formas de utilizar mapas. Nosotros trabajaremos con el método **CSIM** (*Client Side Image Map*) ya que es el más usado y por que se ejecuta en el lado del cliente no necesitando hacer llamadas al servidor web incrementando el tiempo de respuesta de la web.

Para realizar un mapa necesitamos:

* Una imagen en la que indicaremos las zonas donde se puede hacer clic mediante unas coordenadas en píxeles. Estas zonas se conocen en **HTML** como “**áreas**” o “**zonas activas**”.
* La etiqueta <**img**> en la que indicaremos que se trata de una imagen de mapa
* Las etiquetas propias del mapa y los enlaces a otras páginas.

***¡Veámoslo realizando la práctica 13.11!***

La etiqueta <**map**> indica que comienza un mapa y le asigna un nombre mediante el atributo “**id**” que se corresponde con el nombre indicado en el atributo “**usemap**” añadido en la etiqueta <**img**>.

La etiqueta <**area**> determina las zonas activas de la imagen. Su sintaxis es:

**<area shape=”forma” coords=”coordenadas” href=”destino” alt=”comentario” />**

A tener en cuenta:

* **Shape**. Determina la forma en la que indicamos las coordenadas de las zonas activas en la imagen del mapa. Puede ser:
  + **Rect**. La zona activa es un rectángulo que viene indicado por las coordenadas en píxeles de la esquina superior izquerda e inferior derecha del rectángulo. (**coords=”x, y, x1, y1”**).
  + **Circle**. La zona activa es un círculo que viene indicado por las coordenadas en píxeles del punto central y la longitud en píxeles del radio. (**coords=”x, y, r”**).
  + **Polygon**. La zona activa es un polígono que viene indicado por las coordenadas en píxeles de tantos puntos como vértices tenga el polígono. (**coords=”x, y, x1, y1,x2, y2,x3, y3, x4, y4”**).
  + **Default**. La zona activa es el resto de la imagen, la parte a la que no se le ha asignado ninguna otra zona activa con “**rect**”, “**circle**” o “**poligon**”.
* **Coords**. Especifica las coordenadas que permiten al navegador construir la forma geomética de la zona activa.
* **Href**. Indica el archivo o la web de destino asociada a la zona activa indicada en “**coords**”. Es opcional porque puede ser que el diseñador desee que alguna zona activa no incluya ninguna acción.
* **Alt**. Permite agregar un comentario. También es opcional.

Existen programas que ayudan a realizar mapas de imágenes como MapThis o Xmap y, por su puesto, Dreamweaver.

***Para terminar de dominar bien esta parte realice la cuarta ejercitación del capítulo 13***

OPACIDAD Y TRANSPARENCIA CON CSS

La propiedad “**opacity**” permite indicar la opacidad o transparencia de un elemento. Su sintaxis es:

**opacity: Valor comprendido entre 0 y 1;**

A tener en cuenta:

* Con el valor “**1**”, la opacidad es completa y la transparencia nula. Con “**0**” la opacidad es nula y el elemento completamente transparente.
* Se aplica al elemento y a todos sus descendientes.
* El elemento al que se le aplica la propiedad “**opacity**” es transparente y deja entrever el elemento que se encuentra situado por debajo de él.
* Esta propiedad es entendida por todos los navegadores.

***¡Veamos un ejemplo realizando la práctica 13.12!***

***Para terminar de dominar este amplio capítulo realice ahora su quinta y última ejercitación.***

CAPÍTULO 14.- ANIMACIONES Y MULTIMEDIA

CON HTML Y CSS

Las últimas especificaciones de CSS nos permiten aplicar animaciones a los distintos elementos de nuestra página web. En este capítulo veremos las diferentes propiedades CSS y etiquetas HTML con las que podremos transformar la visibilidad de nuestras webs aplicándoles transformaciones y animaciones a sus elementos así como añadiéndoles sonido y vídeo.

TRANSFORMACIONES

La propiedad “**transform**” permite aplicar distintas transformaciones: rotar, escalar, mover e inclinar o sesgar, en 2d y 3d, sobre un elemento. Su sintaxis es:

**transform: función(valor, valor) función(valor, valor);**

donde “**función**” es siempre una de las siguientes palabras claves:

**rotate(ángulo) / rotateX(ángulo) / rotateY(ángulo)**

**scale(númeroX, númeroY) / scaleX(número) / scaleY(número)**

**skew(ángulo) / skewX(ángulo) / skewY(ángulo)**

**translate(númeroX, númeroY) / translate(número) / translate(número)**

**matrix(número, número, número, número, número, número)**

A tener en cuenta:

* Esta propiedad es aceptada tanto por los elementos **HTML** en línea como los de bloque.
* Solo afecta a los elementos hasta su borde. Los márgenes se excluyen al aplicar las transformaciones.
* Las transformaciones no afectan al flujo del documento. Si un elemento se escala o se rota, para el resto no existen tales transformaciones, se comportan como si no hubiese cambiado su tamaño o su visualización.
* La propiedad “**transform**” está implementada en todos los navegadores y sólo necesita prefijo privativo para los navegadores basados en el motor “**webkit**”.
* **El valor rotate(ángulo)**, gira el elemento respecto a su posición original.
  + El valor del ángulo puede definirse en grados sexagesimales (deg), radianes (rad) o grados centesimales (grad).
  + Los valores positivos efectúan la rotación del elemento en el sentido de las agujas del reloj y los valores negativos lo hacen en sentido inverso.
  + Cuando se define un único valor, éste se aplica a los ejes X e Y. Con dos valores, se aplican respectivamente al eje de las X y al de las Y.
  + El punto de rotación coincide con el centro del elemento **HTML** que se gira.
  + **rotateX(ángulo)**, gira el elemento sólo en el eje X.
  + **rotateY(ángulo)**, gira el elemento sólo en el eje Y.

***¡Veamos un ejemplo realizando la práctica 14.1!***

* **El valor scale(númeroX, númeroY)**, permite aumentar o reducir el tamaño del elemento original.
  + Los ejes horizontal y vertical se controlan independientemente, por lo que podemos escalar el elemento de distinta manera en los dos ejes.
  + El “**número**” indica la proporción en la que se deber aumentar o reducir. Con el valor 1 el elemento queda como está. Los valores mayores a 1 realizan un zoom acercándose y los decimales menores a 1, un zoom alejándose.
  + Cuando se define un único valor, éste se aplica a los ejes X e Y. Con dos valores, se aplican respectivamente al eje de las X y al de las Y.
  + El punto de rotación coincide con el centro del elemento **HTML** que se gira.
  + **scaleX(número)**, escala el elemento sólo en el eje X.
  + **scaleY(número)**, escala el elemento sólo en el eje Y.

***¡Veamos un ejemplo realizando la práctica 14.2!***

* **El valor skew(ángulo, ángulo)** hace un sesgado del elemento original convirtiéndole en un romboide. Para ello se inclinan los lados horizontal y vertical en los ángulos indicados.
  + Los lados horizontal y vertical se controlan independientemente, por lo que podemos sesgar el elemento de distinta manera en los dos lados.
  + El “**ángulo**” indica la inclinación de los lados.
  + Cuando se define un único valor en “**ángulo**”, éste se aplica a los lados verticales y horizontales. Con dos valores, se aplican respectivamente al la vertical y al horizontal.
  + El valor del ángulo puede definirse en grados sexagesimales (deg), radianes (rad) o grados centesimales (grad).
  + **skewX(ángulo)**, inclina sólo los lados verticales.
  + **skewY(ángulo)**, inclina sólo los lados horizontales.

***¡Veamos un ejemplo realizando la práctica 14.3!***

* **El valor translate(número, número)**, desplaza el elemento hacia donde indican los parámetros.
  + El primer “**número**” es la translación que se produce en horizontal hacia la derecha. (Eje X). Si la cantidad indicada es negativa la translación es horizontal hacia la izquierda.
  + El segundo “**número**” indica la translación del elemento verticalmente, hacia abajo. (Eje Y). Si la cantidad indicada es negativa la translación es vertical hacia arriba.
  + El valor de “**número**” puede definirse en px, %, em, in, mm o cm.
  + Cuando se define un único valor en “**número**”, éste se aplica a los ejes X e Y. Con dos valores, se aplican respectivamente al desplazamiento horizontal y al vertical.
  + **translateX(número)**, desplaza el elemento a lo largo del eje X (horizontal).
  + **translateY(número)**, desplaza el elemento a lo largo del eje Y (vertical).

***¡Veámoslo en la práctica 14.4!!***

* **El valor matrix(número ...)** Combina todos los valores anteriores. Utiliza 6 parámetros que se obtienen transformando las coordenadas de los vértices en una matriz matemática. La fórmula es algo complicada por eso solo nombraremos que existe y no profundizaremos más en él puesto que tampoco se utiliza masivamente.

La propiedad “**transform-origin**” indica cual será el punto de origen desde el que se transforma el elemento. Piense en los valores que hemos estudiado en la propiedad “**transform**”, todos, las rotaciones, los escalamientos, los sesgos y los traslados se han realizado tomando como punto de referencia el centro del elemento que hemos transformado. Esta propiedad, pues, permite cambiar ese punto. Su sintaxis es:

**transform-origin: coordenadaX coordenadaY;**

A tener en cuenta:

* **CoordenadaX**, indica la coordenada X (horizontal) del origen. Puede expresarse en % o una medida o las palabras claves “**left**”, “**center**” y “**right**”.
* **CoordenadaY**, indica la coordenada Y (vertical) del origen. Puede expresarse en % o una medida o las palabras claves “**top**”, “**center**” y “**bottom**”.

***Para terminar de dominar bien esta parte realice la 1ª ejercitación del capítulo 14***

TRANSFORM EN 3D

En las propiedades anteriores nos manejábamos en 2D, en el plano de la pantalla. Recientemente **CSS3** ha dado el salto a la 3ª dimensión con las siguientes propiedades. Como aún no son definitivas solamente las enunciaremos para conocer su existencia.

* **Transform-style:** Especifica cómo se representan los elementos anidados en el espacio 3D. Admite los valores: “**flat**” y “**preserve**”.
  + **Flat**: Los hijos del elemento al que se le declara no conservan su posición en la 3D. Desaparecen de la vista.
  + **Preserve-3D**: Los hijos del elemento al que se le declara conservan sus propiedades y se mantienen visibles.
* **Perspective.** Aplica la 3ª dimensión o profundidad. Ajusta el punto de vista con el que se muestra el elemento. Cuanto más bajo es el valor más profundidad otorga. Con un valor de 0 el resultado es que no hay efecto 3D.
* **Perspective-origin.** Establece la posición X e Y desde la cual el espectador parece estar mirando a los hijos del elemento.
* **Backface-visibility.** Controla si se muestra o no la cara posterior del elemento al transformarlo. Tiene dos valores “**visible**” y “**hidden**”.

¿Se atreve a probar su funcionamiento?

TRANSICIONES con CSS3

En **CSS3** una transición es un cambio progresivo de un valor de una propiedad a otro valor de la misma propiedad. Por ejemplo para la propiedad “color” pasar del valor “verde” al valor “amarillo”. O para la propiedad “width” pasar del valor 20px., al valor 100px. Al hablar de cambio “progresivo” significamos que el cambio de un valor a otro se realiza pasando por sus valores intermedios provocando un efecto de animación.

Las transiciones no se pueden aplicar a todas las propiedades CSS, solo a aquellas cuyos valores tengan un componente numérico. Al final de este apartado incluiremos un listado de las propiedades a las que se puede aplicar.

A continuación veremos las 4 propiedades CSS que nos permiten realizar transiciones.

La propiedad “**transition-property**” indica las propiedades a las que se se aplicarán las transiciones, por ejemplo, como ya hemos dicho, el color o la anchura de un elemento. Su sintaxis es:

**transition-property: none / all / propiedad, propiedad, … ;**

A tener en cuenta:

* **None**. Indica que no se aplican las transiciones a ninguna propiedad. Es el valor por defecto.
* **All**. Indica que las transiciones se aplicarán a todas las propiedades que puedan admitirlas.
* **Propiedad**. Se trata del nombre de la propiedad o propiedades a las que queremos aplicarles las transiciones. Si hay más de una deben ir serparadas por coma.

***¡Vayamos a la práctica 14.5!***

La propiedad “**transition-duration**” indica el tiempo que tardará en realizarse la transición. Su sintaxis es:

**transition-duration: tiempo;**

A tener en cuenta:

* **Tiempo**. Indica el tiempo de duración de la transición en segundos “**s**” o milisegundos “**ms**”.
* Si en “**tiempo**” indicamos un valor negativo se computa como 0 y por lo tanto no se produce la transición
* Si en la propiedad “**transition-property**” se indican varias propiedades a cambiar se puede declarar un tiempo de transición para cada una.

***¡Veámoslo en la práctica 14.6!***

Con las dos propiedades anteriores bastaría para que la transición funcionara pero tenemos otras propiedades…

La propiedad “**transition-timing-function**” determina los cambios de velocidad mientras se ejecuta la transición en el tiempo estipulado. Su sintaxis es:

**transition-timing-function: linear / ease / ease-in / ease-out / ease-in-out / steps (num) / steps (num, start) / steps (num, end) / cubic-bezier (num, num, num, num);**

A tener en cuenta:

* **linear**. La velocidad de la transición se mantiene constante durante toda su duración.
* **Ease.** Es el valor por defecto. La transición se efectúa rápida al principio y se ralentiza al terminar.
* **Ease-in.** La transición se efectúa lenta al principio y coge su máxima velocidad al ir llegando a su fin.
* **Ease-out.** La transición es rápida al principio y va disminuyendo la velocidad al finalizar.
* **Ease-in-out.** La trasición va lenta al principio, se vuelve rápida a la mitad y acaba lenta.
* **Steps (num).** La transición no es líneal sino a saltos. “**num**” son los saltos que se producen en la transición.
* **Steps(num, start).** Indica el número de saltos que se realiza al inicio de cada fracción de tiempo.
* **Steps(num, end).** Indica el número de saltos que se realiza al final de cada fracción de tiempo.
* **Cubic-bezier.** Definimos la velocidad de la transición en base a una curva cúbica de Bezier. Para ello incorporamos cuatro valores “**num**” que serán números decimales entre el 0 y el 1.

***¡Vemos con la práctica 14.7 cómo se efectúan las transiciones utilizando timing-function!***

La propiedad “**transition-delay**” indica el retraso de tiempo que transcurre entre la acción que desencadena la transición y su comienzo. Su sintaxis es:

**transition-delay: tiempo;**

A tener en cuenta:

* **Tiempo**. Indica el tiempo en que se retrasará el inicio de la transición en segundos “**s**” o milisegundos “**ms**”.
* **Tiempo,** admite valores de tiempo negativos.

***¡Veamos cómo afecta esta propiedad realizando la práctica 14.8!***

Las propiedades de transición se pueden indicar de forma acortada o resumida con la siguiente sintáxis:

**transition: property duration timing-function delay;**

A tener en cuenta:

* Se puede utilizar la forma acortada para declarar varias propiedades a cambiar con igual o distintos parámetros en cada una de las transiciones. En este caso se declaran las distintas propiedades en el orden indicado separadas por comas de la siguiente forma: “**transition: color 3s linear 0s, width 1s ease 1s;**”.

***Para terminar de dominar bien esta parte realice la 2ª ejercitación del capítulo 14***

ANIMACIONES EN CSS

Hasta ahora, en este capítulo, hemos visto las propiedades de transformación y transición. Estas inician sus cambios después de la intervención del usuario, casi siempre con la pseudoclase “**:hover**”. Pero **CSS3** provee de una serie de propiedades que permiten que se realicen animaciones sin la intervención del usuario es decir que se activan automáticamente al cargar la página web en el navegador.

Otra diferencia entre las animaciones y las propiedades “**transform**” y “**transition**” es que estas tienen un estado inicial y uno final, mientras que en aquellas permiten alterar la línea de tiempo y desarrollo de los cambios en función de las necesidades o gustos del desarrollador.

Las animaciones en **CSS3** se crean utilizando la técnica del “**Fotograma Clave**”. Los fotogramas clave son los que marcan el estado del elemento animado al principio y al final de la animación y en las posiciones intermedias en las que el elemento tenga que variar de forma o propiedad.

Además de los fotogramas clave disponemos de una serie de propiedades, semejantes a las de transiciones, que dan nombre a la animación, marcan la duración, los retrasos, etc…

Los fotogramas claves se crean con la regla “**@Keyframes**”. Su sintaxis es:

**@keyframes “nombre\_animación” {**

**0% {propiedad1: valor; propiedad2: valor; …}**

**…**

**100% {propiedad1: valor; propiedad2: valor; …}**

**}**

A tener en cuenta:

* **Nombre\_animación**, es el nombre que asignaremos a la animación mediante la propiedad “**animation-name**”. Este nombre es necesario para poder aplicar el resto de propiedades de la animación.
* Dentro de la declaración, entre llaves, pondremos los diversos fotogramas clave los cuales son unas “subreglas” **CSS**, (en realidad cada fotograma clave es en si una regla **CSS**), compuestas por:
  + **Selector**. Indicados en tantos por ciento, posicionan el elemento en un punto concreto de la animación exponiendo las propiedades que el elemento debe tener en ese punto de la animación. El primero debe ser siempre el 0% que indica el principio de la animación, y el último el 100% que indica el final. Entre medias se pueden colocar los fotogramas claves que se desee indicando su tanto por ciento.
  + **Declaraciones**. Dentro de las llaves de los fotogramas pondremos las propiedades **CSS** que tenga el elemento en ese punto concreto de la animación. Lo normal es poner las mismas propiedades en varios fotogramas, pero con diferentes valores.
* Es obligatorio poner al menos dos fotogramas clave, el inicio y el final de la animación.
* El paso de un fotograma clave al siguiente se realiza de forma progresiva, siempre que esto sea posible. Sólo las propiedades cuyo valor esté basado en un número, (números, medidas, porcentajes, colores), pueden variar progresivamente. Las propiedades cuyos valores no son números, (palabras clave, nombres, urls, etc…), son ignoradas y no se pude hacer ningún tipo de cambio en ellas.
* Para mover un elemento dentro de la página, éste debe estar posicionado, es decir, tiene que tener la propiedad “**position: absolute;**” o “**position: relative;**” ya que sino no se le pueden aplicar las propiedades de posicionamiento (left, right, top, bottom), que son las que cambian al elemento de sitio.

Veamos ahora las propiedades de animación imprescindibles para realizar una animación simple:

La propiedad “**animation-name**” indica a qué elemento debe aplicarse la animación y asigna un nombre a la animación. Su sintaxis es:

**animation-name: “nombre\_animación”;**

A tener en cuenta:

* **Nombre\_animación**, es el nombre que usaremos en la regla @keyframes.

La propiedad “**animation-duration**” indica el tiempo que va a durar la animación. Su sintaxis es:

**animation-duration: “tiempo”;**

A tener en cuenta:

* **Tiempo**, Indica el tiempo de duración de la animación en segundos “**s**” o milisegundos “**ms**”.
* Esta propiedad es obligatoria ya que toda animación tiene siempre una duración en el tiempo.

***¡Y ahora vayamos a la práctica 14.9!***

La propiedad “**animation-iteration-count**” indica las veces que se repetirá la animación. Su sintaxis es:

**animation-iteration-count: “num” / infinite;**

A tener en cuenta:

* **Num**, es un número entero positivo que Indica el número de veces que se repetirá la animación. Su valor por defecto es 1 por lo que si no ponemos esta propiedad la animación se ejecutará una vez.
* **Infinite,** indica que la animación se repetirá indefinidamente.

***¡Veámoslo en la práctica 14.10!***

La propiedad “**animation-direction**” indica si la animación al llegar al final acaba o si debe volver en sentido inverso. Su sintaxis es:

**animation-direction: normal / alternate;**

A tener en cuenta:

* **Normal**, es el valor por defecto, la animación, una vez concluido el tiempo que se le ha marcado finaliza.
* **alternate,** indica que una vez terminada la animación se repitá en sentido inverso, volviendo a reproducirse la secuencia de atrás a adelante. (Con lo cual la duración de la animación será el doble).

***¡Veámoslo en la práctica 14.11!***

La propiedad “**animation-timing-function**” determina los cambios de velocidad mientras se ejecuta la animación en el tiempo estipulado. Es semejante a la propiedad “**transition-timing-function**” que ya conocemos. Su sintaxis es:

**animation-timing-function: linear / ease / ease-in / ease-out / ease-in-out /**

**cubic-bezier (num, num, num, num);**

A tener en cuenta:

* **linear**. La velocidad de la animación se mantiene constante durante toda su duración.
* **Ease.** Es el valor por defecto. La animación va lenta al principio para volverse rápida en el medio y terminar mucho más lenta.
* **Ease-in.** La animación se efectúa lenta al principio y coge su máxima velocidad al ir llegando a su fin.
* **Ease-out.** La animación es rápida al principio y va disminuyendo la velocidad al finalizar.
* **Ease-in-out.** La animación va lenta al principio, se vuelve rápida a la mitad y acaba lenta.
* **Cubic-bezier.** Definimos la velocidad de la animación en base a una curva cúbica de Bezier. Para ello incorporamos cuatro valores “**num**” que serán números decimales entre el 0 y el 1.

La propiedad “**animation-delay**” indica el tiempo que tarda la animación en empezar después de cargarse la página. Su sintaxis es:

**animation-delay: tiempo;**

A tener en cuenta:

* **Tiempo**. Indica el tiempo en que se retrasará el inicio de la animación en segundos “**s**” o milisegundos “**ms**”.
* **Tiempo,** admite valores de tiempo negativos.

La propiedad “**animation-play-state**” indica si la animación está ejecutándose o en pausa. (Se utiliza junto con JavaScript para que el usuario, mediante un botón, pueda pausar o poner en marcha la animación). Su sintaxis es:

**animation-play-state: paused / running;**

A tener en cuenta:

* **Paused**. Indica que la animación está en pausa.
* **Running.** Indica que la animación se está ejecutando. Este es el valor por defecto.

Al igual que en otros casos, las propiedades de animación se pueden indicar de forma acortada o resumida con la siguiente sintáxis:

**animation: animation-name animation-duration animation-timing-function animation-delay animation-iteration-count animation-direction;**

A tener en cuenta:

* Las propiedades “**animation-name**” y “**animation-duration**”, que son obligatorias, deben declararse en primer lugar y por ese orden. Las propiedades que no son imprescindibles u obligatorias pueden cambiar de orden e incluso omitirse.

***Para terminar de dominar bien esta parte realice la 3ª ejercitación del capítulo 14***

INSERTAR ARCHIVOS DE AUDIO CON HTML

Insertar un archivo de audio en una web se realiza de forma muy sencilla mediante la etiqueta <**audio**>. Su sintaxis básica es:

**<audio src=”archivo\_de\_sonido”> … </audio>**

Esta etiqueta tiene los siguientes atributos:

* **Src**. Define la ruta del archivo de sonido. Lógicamente es obligatorio.
* **Controls.** Muestra los controles del reproductor de audio. Incluye las funcionalidades de reproducción, pausa, avance y volumen. Si no incluimos este atributo no veremos ningún efecto en pantalla, ni podremos reproducir el sonido a menos que indiquemos el atributo “**autoplay**”.
* **Autoplay.** Indica que el archivo de sonido se reproduzca automáticamente tras la carga de la página.
* **Loop.** Especifica que el archivo de sonido se repita constantemente.
* **Preload.** Indica al navegador que debe descargar el archivo de audio durante la carga de la página de modo que esté disponible para una reproducción inmediata una vez la solicite el usuario. Puede tomar los siguientes valores:
  + **Preload=”none”.** Sin carga previa.
  + **Preload=”metadata”.** Carga previa de los metadatos asociados al archivo de sonido.
  + **Preload=”auto”.** Carga previa automática.
  + El atributo “**preload**” se ignorará si el atributo “**autoplay**” está activado.

**El Problema de los formatos de sonido**

El problema que nos encontramos a la hora de añadir audio a la web es que no todos los navegadores soportan, debido a problemas de patentes, todos los formatos de sonido que existen en la actualidad. Los más habituales son: Ogg, Mp3, Acc y Wav.

A la hora de redactar este manual Google Chrome y Firefox soportan todos los formatos. Safari soporta mp3, wav y acc e Internet Explorer soporta mp3 y aac.

Para asegurarnos que nuestro sonido se puede oir en todos los navegadores la solución más sencilla es utilizar un programa de conversión de audio como por ejemplo “**Audacity**”.

La etiqueta <**source**> permite resolver la problemática de los distintos formatos de sonido. Se utiliza para especificar varios tipos de archivos de audio. Cada navegador escogerá el formato que mejor convenga o el que pueda reproducir. Su formato es:

**<audio controls>**

**<source src=”música.ogg”>**

**<source src=”música.mp3”>**

**<source src=”música.acc”>**

**Su navegador no soporta la etiqueta audio.**

**</audio>**

El navegador reproducidará el primer archivo que entienda. Los otros son ignorados.

La etiqueta <**source**> tiene dos atributos:

* **Src**. Define la ruta del archivo de sonido. Lógicamente es obligatorio.
* **Type**. Define el tipo MIME del contenido. Sirve para acelerar el proceso de carga. Puede valer:
  + Type=”audio/oog”.
  + Type=”audio/mpeg”.
  + Type=”audio/acc”.

***Para terminar de dominar bien esta parte realice la 4ª ejercitación del capítulo 14***

INSERTAR ARCHIVOS DE VIDEO CON HTML

Al igual que el sonido, insertar un archivo de vídeo en una web se realiza de forma muy sencilla mediante la etiqueta <**video**>. Su sintaxis básica es:

**<video src=”archivo\_de\_video”> … </video>**

Esta etiqueta tiene los siguientes atributos:

* **Src**. Define la ruta del archivo de vídeo. Lógicamente es obligatorio.
* **Width**. Indica la anchura del vídeo.
* **Height**. Determina la altura del vídeo. Al igual que con la etiqueta <**img**> se puede definir de forma explícita las dimensiones del vídeo. En otro caso, el elemento se muestra por defecto con la altura y la anchura propias del vídeo. Si especifica una dimensión, pero no la otra, el navegador ajustará automáticamente el tamaño de la dimensión que no se haya especificado para preservar la proporción de aspecto del vídeo.
* **Poster**. Permite especificar una imagen que el navegador usará mientras está descargando el vídeo o hasta que el usuario inicie su reproducción. Si no se especifica este atributo, se inserta la primera imagen del vídeo en su lugar.
* **Controls.** Muestra los controles del reproductor de vídeo. Incluye las funcionalidades de reproducción, pausa, avance y volumen. . Si no incluimos este atributo veremos el primer fotograma del video pero no podremos reproducirlo a menos que indiquemos el atributo “**autoplay**”.
* **Autoplay.** Indica que el archivo de vídeo se reproduzca automáticamente tras la carga de la página.
* **Loop.** Especifica que el archivo de vídeo se repita constantemente.
* **Preload.** Indica al navegador que debe descargar el archivo de vídeo durante la carga de la página de modo que esté disponible para una reproducción inmediata una vez la solicite el usuario. Puede tomar los siguientes valores:
  + **Preload=”none”.** Sin carga previa.
  + **Preload=”metadata”.** Carga previa de los metadatos asociados al archivo de vídeo.
  + **Preload=”auto”.** Carga previa automática.
  + El atributo “**preload**” se ignorará si el atributo “**autoplay**” está activado.

**El Problema de los formatos de vídeo**

Al igual que en el caso del sonido con el vídeo también tenemos el problema de los distintos formatos que soportan los navegadores. En la actualidad podemos encontrar los siguientes: Ogv, H.264, y WepM.

A la hora de redactar este manual Google Chrome y Firefox soportan todos los formatos. Safari e Internet Explorer soporta todos aunque instalando los plugin adecuados.

Aunque prácticamente las últimas versiones de los navegadores soportan todos los formatos de audio y vídeos debemos tener en cuenta que muchos usuarios no tienen actualizados sus navegadores y que muchas personas utilizan tecnologías de navegación móviles. Para estar completamente al día de los codecs de sonido y audio soportados por los diferentes navegadores de todos los tipos recomendamos visitar las webs “HTML5\_audio” y “HTML5\_video” de la Wikipedia en su versión inglesa.

También podemos utilizar un programa de conversión de video como “Miro Video Converter”, “Prism Video Converter” o la extensión de Firefox “Firefogg”, para asegurarnos que nuestro video se puede ver en todos los navegadores.

Además, como en el caso del audio, podemos utilizar la etiqueta <**source**>.

***Para terminar de dominar bien esta parte realice la 5ª ejercitación del capítulo 14***

CAPÍTULO 15.- OTRAS TECNOLOGÍAS EN HTML5. CANVAS, GEOLOCALIZACIÓN…

A lo largo de todo este curso hemos ido aprendiendo el lenguaje **HTML5** y su compañero inseparable **CSS3**, pero el mundo **HTML5** es una plataforma abierta que no se queda aquí. El **W3C** define 8 tecnologías que lo componen aunque, a la hora de elaborar este manual, aún se estén desarrollando:



* **Semántica.** Es el conjunto de todas las etiquetas que hemos ido estudiando a lo largo de este curso. Es la parte central y frontal del **HTML5**.
* **Offline y Almacenamiento local.** Se trata de servicios para que las aplicaciones **HTML5** sigan funcionando incluso aunque no se disponga de conexión a internet. Se trata del Web storage, local storage, Bases de Datos indexadas y Apis de ficheros.



* **Acceso al dispositivo.** Se trata de aplicaciones que permiten el acceso al dispositivo donde está navegando el usuario. Acceso a la cámara, al micrófono. El más avanzado es la geolocalización.
* **Conectividad** Se trata de aplicaciones que permiten una conectividad más eficiente para disponer de comunicaciones en tiempo real, chats, juegos más rápidos y en general mejor comunicación con el servidor. Son los llamados Web sockets.



* **Multimedia.** El Audio y Vídeo que ya conocemos de forma nativa en el navegador.
* **3d, Gráficos y Efectos:** Se trata de tecnologías como: Canvas, WebGL, SVG y las propiedades para 3D de CSS3, que permiten realizar webs impresionantes efectos visuales representados de forma nativa en el navegador. Por ejemplo para el desarrollo de juegos en 3D.



* **Optimización e integración.** Tecnologías como Web Workers y XMLHttpRequest que se encargan de que el navegador se integre con el dispositivo lo que permitirá navegar muchísimo más rápido.



* **CSS3.** Los estilos y efectos que ya conocemos y que mejoran las webs sin sacrificar su rendimiento y estructura semántica.

En este tema vamos a tratar brevemente algunas de estas tecnologías para tener un conocimiento global de la especificación **HTML5**. No profundizaremos en ellas puesto que en todas se utiliza **JavaScript** para su funcionamiento y este es objeto de otro curso de **Fray Luis Centro Académico**.

CANVAS

“**Canvas**” se podría traducir por “lienzo”. Se trata de un espacio de dibujo, un lienzo en la web, para dibujar. En dicho espacio podemos dibujar mediante scripts de **JavaScript**.

La etiqueta <**canvas**> tiene la siguiente sintáxis:

**<canvas width=”200px” height=”200px”> … </canvas>**

A tener en cuenta:

* Los atributos “**width**” y “**height**” definen las dimensiones del lienzo de dibujo. Si no se especifican el área de dibujo o lienzo tendrá una dimensión de 300px de largo y 150px de alto.
* Es posible incluir varias etiquetas <**canvas**> en la misma página. El identificador “**id**” puede resultar útil en este caso.
* Es posible aplicarle estilos de la misma forma que a cualquier otra imagen de la web.
* Por defecto la zona de dibujo que crea esta etiqueta es transparente por lo que, al menos en la fase de diseño, es recomendable agregar un estilo de borde simple de 1px.
* Las especificaciones de **HTML5** preveen en un futuro diseños en 3D.
* Esta etiqueta es reconocida por todos los navegadores.

Esto es todo en lo que respecta a canvas en **HTML**. A partir de aquí el control pasa a a **JavaScript** para realizar los dibujos y diseños. Se pueden realizar líneas y figuras geométricas con color, distintos grosores, añadir imágenes, texto, sobreados y transformaciones.

***Pero… ¡Veamos un ejemplo sencillo realizando la práctica 15.1!***

A modo de apunte, otras funciones de dibujo en **JavaScrip** son:

* **lineTo(x, y)**. Dibuja una línea recta. Los argumentos “**x**” e “**y**” son las coordenadas del extremo final de la recta. El punto de partida depende de los caminos diseñados anteriormente, dado que el último punto de un camino es el punto de partida para el siguiente. No obstante el punto de partida puede modificarse con la función “**moveTo**”.
* **moveTo(x, y)**. Indica un nuevo punto de partida para el siguiente dibujo a realizar.
* **clearRect(x, y, width, height)**. Borra la zona especificada haciéndola completamente transparente.
* **stroke()**. Como hemos comprobado con los rectángulos se aplican directamente el diseño, color, etc. Sin embargo, para las demás formas, líneas, etc, etc es como si estuviéramos usando un puntero sin tinta. Para que se vea la línea, es necesario ejecutar después de la función correspondiente, la función “stroke()”.
* **lineWidth=valor**. Define el grosor de la línea. El valor por defecto es 1px.
* **lineCap=valor**. Determina como se diseñan los extremos de cada línea. Existen tres valores:
  + **butt.** La línea termina exactamente en el lugar definido por el código. Es el valor por defecto.
  + **round.** Agrega medio círculo a cada extremo de la línea.
  + **square.** Agrega un borde cuadrado a cada extremo de la línea.
* **arc(x, y, radio, anguloInicial, anguloFinal, sentidoInverso).** “**x**” e “**y**” son las coordenadas del centro del círculo. “**radio**” es el radio del mismo. Los parámetros “**anguloInicial**” y “**anguloFinal**” definen los puntos de partida y de llegada del arco en radianes, medidos a partir del eje X. El parámetro “**sentidoInverso**” puede ser “**false**”, dibuja el arco en el sentido de las agujas del reloj, o “**true**” que dibuja el arco en el sentido inverso a las agujas del reloj.
* **quadraticCurveTo.** Dibuja formas complejas a través de fórmulas matemáticas cuadráticas.
* **bezierCurveTo.** Dibuja formas complejas a través de fórmulas matemáticas de curvas de Bézier

***Para terminar de dominar bien esta parte realice la 1ª ejercitación del capítulo 15***

SVG

Además de **Canvas**, **SVG**, (Scalable Vector Graphics), es otra posibilidad para crear gráficos web interactivos y que **HTML5** la reconoce de manera nativa. **SVG** está basado en el lenguaje **XML** que es muy parecido al **HTML** por lo que las instrucciones también se indican con etiquetas.

La etiqueta <**svg**> tiene la siguiente sintáxis:

**<svg width=”200px” height=”200px”> … </svg>**

A tener en cuenta:

* Los atributos “**width**” y “**height**” definen las dimensiones del dibujo.

***¡Pero veamos un ejemplo realizando la práctica 15.2!***

Las principales diferencias entre **Canvas** y **SVG** son:

|  |  |
| --- | --- |
| **CANVAS** | **SVG** |
| Orientado a Pixel. Manipula píxeles. | Basado en el modelo de objetos. Manipula elementos como las mismas etiquetas **HTML**. |
| En una web, es un elemento individual, similar en su comportamiento a la etiqueta <**img**>. | En una web, cada elemento gráfico **SVG**, se puede tratar individualmente como si fuera una etiqueta independiente. |
| Para modificar un gráfico tenemos que utilizar **JavaScript**. | Cada elemento gráfico se puede modificar con etiquetas y/o estilos **CSS**. Se permite a los lenguajes de script el acceso a todos los elementos, propiedades y atributos que componen un documento **SVG**. |

|  |  |
| --- | --- |
| El modelo evento/interacción con el usuario es rudimentario exclusivamente a nivel del elemento **canvas**; las interacciones se deben programar manualmente a partir de las coordenadas del ratón. | Existe la posibilidad de asignar eventos a los distintos elementos de un documento **SVG** tales como al posicionar el ratón o al hacer clic, etc, etc… |
| Modo inmediato. **Canvas** no tiene consciencia de las formas que se han dibujado en el lienzo. Lo único que permanece es el mapa de bits resultante. | Modo retenido. Los elementos del dibujo **SVG** permanecen dentro de un modelo en memoria. |
|  | No pierde calidad si se hace zoom o si se redimensiona. |
|  | Se puede realizar el dibujo con un programa de diseño gráfico como **Illustrator** y pasarlo al código **HTML** bien, copiando y pegando, bien mediante llamadas al archivo externo **.SVG** utilizando las etiquetas <**object**>, <**embed**> e <**img**>. |
|  | Cuando el elemento <**svg**> aparece en un documento **HTML5**, funciona igual que un bloque “**inline**”. |

WEB STORAGE

Cuando navegamos por internet hay muchas páginas que “recuerdan” que ya les hemos visitado. Esta memoria se consigue a través de las tan famosas “cookies”, pequeños pedacitos de información que viajan en cada petición entre el cliente y el servidor web. La utilización de cookies ya no es considerada una buena solución ya que:

* Aumentan el peso de cada petición al servidor.
* Tienen una limitación de 4Kb de espacio disponible.
* No todos los visitantes de nuestra web pueden tener las cookies habilitadas ya que muchos usuarios desconfían de la utilidad que muchas empresas dan a dichas cookies.

**Web Storage**, especificación incluida de forma nativa en **HTML5**, permite utilizar **JavaScript** para guardar información en los navegadores de nuestros usuarios que se elimine al finalizar la sesión, “**SessionStorage**”, o que se guarde de manera permanente en su disco duro, “**LocalStorage**”.

Otras tecnologías de **HTML5** que permiten que las aplicaciones almacenen datos en los dispositivos del cliente son: **Web SQL Database**, **Indexed Database** y **File Access**.

Hay varias razones por las que puede ser recomendable utilizar el almacenamiento en el cliente:

* Permite que una aplicación web funcione cuando el usuario no está conectado, sincronizando los datos cuando vuelve a establecer conexión.
* Aumenta el rendimiento ya que puede mostrar una gran cantidad de datos en la web en el momento en que el usuario hace clic en el sitio, sin tener que esperar a que vuelvan a descargarse.
* Se programa de manera más sencilla con **JavaScript**.

**Web Storage**, al utilizar **JavaScript**, supera ampliamente el alcance de este curso por lo que sólo la incluimos a modo informativo.

WEB WORKERS

Uno de los problemas del lenguaje **JavaScript** es que no permite la ejecución de script en paralelo. El usuario lo nota porque en procesos largos y costosos el navegador se queda “congelado” durante unos instantes.

**Web Workers** es una especificación **HTML5** que permite ejecutar tareas de fondo en paralelo al programa **JavaScript** principal sin afectar al rendimiento de la página web.

WEBSOCKETS

**WebSockets** es una tecnología avanzada que hace posible abrir una sesión de comunicación interactiva para actualizaciones de información entre el navegador del usuario y un servidor. Con esta  API, puede enviar mensajes a un servidor y  recibir  respuestas controladas por eventos sin tener que consultar al servidor para una respuesta. Muy útiles para aplicaciones web de indicadores bursátiles, plataformas web orientadas al trabajo colaborativo, sistemas de notificaciones, juegos on-line, o los conocidos chats.

DRAG AND DROP – ARRASTRAR Y SOLTAR

Esta es una de las nuevas características añadidas en **HTML5** que sirve para crear la capacidad de arrastrar y soltar un elemento de un lugar a otro de igual forma que se hace con los iconos de **Windows**. Hasta ahora, esta funcionalidad se intentaba realizar usando **JavaScript** y **Flash** pero en **HTML5** se implementa de forma nativa por lo que, sin necesidad de recurrir a complementos externos y con un poquito de JavaScript, podemos utilizarlo en nuestras webs.

La tarea de arrastrar y soltar en un navegador, está compuesta por dos partes básicas: los atributos **HTML** y la función **JavaScript**.

En cuanto a los elementos **HTML** se necesitan dos tipos: un elemento arrastrable y otro en el que se pueda soltar. Cada uno de ellos se especifica con una serie de atributos:

Atributos para el elemento arrastrable:

* **draggable**: Si es “**true**” el elemento se puede arrastrar, si es “**false**” o se omite, no se puede mover el elemento.
* “**ondragstart**”: Aquí se indicará la acción que se llevará acabo cuando se empiece a arrastar el elemento.
* “**ondrag**”: Aquí se indicará la acción que se produce mientras se arrastra el elemento.
* “**ondragend**”: Aquí se indicará la acción que se llevará acabo cuando se termine de arrastar el elemento, cuando se suelte.

Y para el elemento al que se pueden arrastrar:

* “**ondragenter**”: Aquí se indicará la acción que se llevará acabo cuando un elemento arrastrable entre dentro del elemento de destino.
* “**ondragover**”: Aquí se indicará la acción que se llevará acabo cuando un elemento arrastrable esté sobre el elemento de destino. En esta función es donde se indica qué elementos arrastrables se pueden soltar en ese lugar.
* “**ondragleave**”: Aquí se indicará la acción que se llevará acabo cuando un elemento arrastrable deje de estar encima del elemento donde se puede soltar.
* “**ondrop**”: Aquí se indicará la acción que se llevará acabo cuando se suelte un elemento arrastrable dentro del elemento de destino.

***¡Pero dejémonos de teoría y veamos un ejemplo práctico realizando la práctica 15.3!*** Comprobaremos que no es tan complicado como parece.

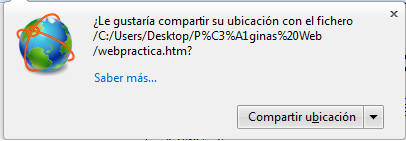
GEOLOCALIZACIÓN

La geolocalización es un procedimiento que permite conocer las coordenadas geográficas, longitud y latitud, de un usuario y posicionarlo sobre un mapa o un plano.

Como otros muchos de los servicios que estamos viendo en este tema, la geolocalización no es nueva en la red, la novedad reside en que esta funcionalidad esté incluida de forma nativa en **HTML5**.

Las aplicaciones de la geolocalización son muchas, una web comercial pude proporcionar los distribuidores más cercanos al usuario o las tiendas y/o hoteles, una red social puede indicar los amigos que se encuentran en los alrededores, etc, etc…

Para geolocalizar al usuario se utiliza su dirección IP, la de su red Wifi o el GPS de su teléfono móvil.

Puesto que obtener esta información del usuario puede generar problemas de protección de la privacidad este servicio de geolocalización debe suponer una opción voluntaria del usuario por lo que al utilizar este servicio aparece una ventana de aviso en el navegador del visitante pidiéndole autorización y la información obtenida se envía a través de internet de forma cifrada. Además los datos obtenidos deben expirar en dos semanas.

Como ya es habitual esta utilidad de geolocalización utiliza en gran medida **JavaScript**.

***Pero… ¡Veámoslo en la práctica 15.4!***

La característica **JavaScript** “**position.coords**” devuelve las coordenadas como hemos visto pero también permite recuperar otros valores como:

* **position.coords.longitude** envía la longitud de la posición actual.
* **position.coords.latitude** envía la latitud de la posición actual.
* **position.coords.altitude** devuelve la altitud de la posición actual.
* **position.coords.accuracy** indica la precisión de las coordenas.
* **position.coords.altitudeAccuracy** proporciona la precisión de la altitud.
* **position.coords.heading** proporciona la posición en grados respecto al norte.
* **position.coords.speed** corresponde con la velocidad del usuario respecto a su última posición.

***Para terminar de dominar bien esta parte realice la 2ª ejercitación del capítulo 15***

CAPÍTULO 16.- RESPONSIVE WEB DESIGN:

LOS MEDIA QUERIES Y

OTRAS TÉCNICAS DE DISEÑO WEB.

En el capítulo 9 de este curso hablamos de páginas web de anchuras fijas o variables. Hasta aquí hemos optado por diseñar con una anchura fija. En este capítulo abordaremos el diseño de webs con una anchura variable que se adapta al tamaño de la ventana que el usuario está utilizando. Esta técnica también se conoce como “**diseño líquido**” o más recientemente como “**Responsive Web Design**” en inglés o “**Diseño Web Adaptativo**” en español.

Con la masiva implantación de móviles y tabletas, cada una con una forma y tamaño de pantalla distinto, cada vez es mayor el número de usuarios que utilizan estos medios para navegar sin que se visualicen las páginas de forma correcta. Se hace pues necesario diseñar para los distintos tamaños de pantalla.

Así, las ventajas de utilizar el **Diseño Web Adaptativo** son:

* Mejora la experiencia de usuario viendo la web de manera más óptima en cada dispositivo para todos los usuarios. Por ejemplo, se evita que el usuario tenga hacer zoom y scroll en su pantalla del móvil para ver bien la web.
* Costes más bajos: ya no es necesario hacer un diseño para cada tipo de pantalla.
* Actualizaciones más eficientes.
* Búsquedas: sólo es necesaria una url, lo que evitará errores o redirecciones.
* Mejora en el posicionamiento de los buscadores.
* Continuidad. Como las webs se ven de forma parecida y correcta en cada dispositivo con el que el usuario se conecte, conseguiremos ofrecerle una experiencia ininterrumpida allá donde esté.

Para diseñar siguiendo un estilo “**Responsive Web Design**” utilizamos:

* Las meta tags: la etiqueta <**meta**>.
* Un diseño fluido o de web flexible abandonando las medidas fijas como píxeles o puntos y adaptando unidades relativas como porcentajes, rem o em.
* **CSS3 Media Queries** que permiten utilizar distintos estilos **CSS** basándose en las medidas del dispositivo donde se va a visualizar nuestra web.

¡Veamos cada uno de ellos!

LA ETIQUETA <META>

Ya aprendimos que en el área del <**head**> de una página web se contiene la información necesaria para la correcta visualización de la página por parte de los navegadores. Y que el contenido de dicha área no se muestra en la página web.

También vimos que en esta área se encuentran los metadatos que es una de las categorías de contenido en **HTML**. Estos metadatos, también conocidos como “**meta tags**”, se indican con las etiquetas <**meta**>, de la cuales ya hemos utilizado alguna como <**meta charset=”utf-8” /**>.

Los metadatos aportan información de varios tipos para diferentes aplicaciones que acceden a nuestra web, como buscadores y utilidades de diversa índole. Cada uno de estos sistemas procesa únicamente aquellas etiquetas <**meta**> que entiende e ignora el resto.

La etiqueta <**meta**> no tiene su correspondiente de cierre y su sintaxis habitual es:

**<meta name=”….” content=”….” />**

Los atributos de la etiqueta <**meta**> son:

* **name.** Asigna un nombre al metadato.
* **content**. Se utiliza para dar valores al metadato creado con “**name**”.

“**name**” y “**content**” se utilizan para indicar propiedades del documento. Cada una de esas propiedades es un metadato. Los valores de estas propiedades o metadatos no son una lista cerrada de palabras clave, aunque el atributo “**name**” suele utilizarse con una serie de valores preestablecidos. Algunos de los metadatos más utilizados son:

* + **application-name**: asigna un nombre al documento web.

Ej: <**meta name=”aplication-name” content=”Página de turismo en España” /**>.

* + **author**. Indica el nombre del creador de la página.

Ej: <**meta=”author” content=”Nombre\_del\_Alumno” /**>

* + **description**. Pequeña descripción de lo que contiene la página web. No debe contener más de 155 caracteres. Utilizado por los buscadores. Se puede acompañar con el atributo “**lang**” para poder ofrecer la descripción de la página para diferentes lenguajes.
  + **generator**. Indica con qué software se ha creado la página. Si la página ha sido creada directamente con **HTML**, como nuestro caso, no es necesario indicarlo.
  + **keywords**. Conjunto de palabras relevantes de la web. Cada palabra o grupo de palabras va separada de las demás por comas. En la actualidad los grandes buscadores han dejado de tenerla en cuenta, aunque muchos diseñadores la siguen implementando para buscadores minoritarios. También se puede utilizar conjuntamente con el atributo “**lang**” para ofrecer “**keywords**” en otros idiomas.
  + **copyright**. Indica el autor de la propiedad intelectual de los contenidos de la web.
  + **organization**. Indica la empresa a la que pertenece la página web.
  + **language**. Especifica el idioma de la web. (El nombre del idioma se indica en inglés). Ej: <**meta name=”language” content=”Spanish” /**>
  + **revisit-after**. Indica a los buscadores cada cuanto tiempo se actualiza la página para que vuelvan a visitarla e indexar los cambios. Lo normal es escribirlo en inglés: “15 days”, “1 month”, “2 months”, etc. Los buscadores más importantes ignoran este metadato. Ej: <**meta name=”revisit-after” content=”1 month” /**>
  + **robots**. Define el comportamiento que los robots de búsqueda deben tener con la página. Podemos incluir todos los valores que queramos separados por comas. Los posibles valores son:
    - **index**. Permite al robot indexar la página y por ello aparecerá en las hojas de resultados de los buscadores. Es el valor por defecto.
    - **noindex**. Indica al robot que no indexe la página.
    - **follow**. Permite al robot seguir los enlaces que hay en nuestra página. También es el valor por defecto junto con “**index**”. Por lo tanto no añadir este metadato es lo mismo que poner:

<**meta name=”robots” content=”index,follow” /**>

* + - **nofollow**. Indica al robot que no puede seguir los links de la página.
    - **noarchive**. Indica a los motores de búsqueda que no utilicen el enlace “En caché” de la página.
    - **nosnippet**. Indica que no se muestre la descripción de la página en los resultados de una búsqueda. (Esa pequeña descripción q ue sale en los resultados de búsqueda, por ejemplo de google, se llama “**snippet**”.)
    - **noimageindex**. Indica que la página no aparezca como la página de referencia de una imagen que se muestra en los resultados de la búsqueda de Google.
    - **noydir**. Indica que no se use la descripción Yahoo Directory, si la hay, como descripción de la página en los resultados de una búsqueda (Sólo para Yahoo).
  + **googlebot**. Es igual que “**robots**” pero sólo para Google.
* **charset**. Se utiliza, como ya sabemos, para especificar la codificación usada en nuestra página. Este elemento debe estar dentro de los primeros 512 bytes de la página.
* **http-equiv**. Este atributo de la etiqueta <**meta**> se utiliza para indicarle al servidor de la página el comportamiento o algunas de sus propiedades. Se trata de metadatos de control. Puede tener los siguientes valores:
  + **http-equiv=“content-type”.** Se utiliza como alternativa al atributo “**charset**” en versiones anteriores a **HTML5**.
  + **http-equiv=”refresh”**. Indica el tiempo para que se refresque la página. Hasta ahora era muy utilizado por ejemplo en las páginas de noticias. También sirve para redireccionar pasados unos segundos a otra página diferente. No es bien vista por los buscadores y el **W3C** recomienda no usarla, así que mejor no emplearla.

Ej: <**meta http-equiv=”refresh” content=”10;url=t-rural.htm” /**>

* + **http-equiv=”Expires”**. Indica cuando deja de ser válido el contenido de la web y obliga al navegador a volver a cachear la página. El valor de la fecha ha de estar en formato GMT. Ej: <**meta http-equiv=”expires” content=”Mon, 23 Sep 2013 12:00:00 GMT” /**>

También es necesario comentar que existen etiquetas meta para informar a los motores de búsqueda de Facebook y otras redes sociales semejantes a: <**meta property=”og:title” content=”Página de turismo” /**>

Por otro lado comentar aquí una curiosa iniciativa que tiene relación con el autor de la página web. Se trata de “**humanstxt.org**”, una web que promueve una web “más humana”. Visite su web y comprobará que en realidad se trata de una alternativa a la etiqueta <**meta name=”autor”/**>.

***Pero… ¡Dejémonos de teoría y realicemos la práctica 16.1!***

***Para asegurar estos conocimientos realice la primera ejercitación del capítulo 16***

EL METADATO VIEWPORT

Por su importancia para el Responsive Web Design y por lo tanto por la relevancia que tomará en los próximos años, nos interesa profundizar y trabajar un poco más con el metadato “**viewport**”.

**Viewport** es un metadato creado por Apple para facilitar el desarrollo de webs para sus dispositivos móviles. Actualmente se puede considerar un estándar utilizado para cualquier dispositivo móvil.

**Viewport** ajusta el contenido de la página al tamaño de dispositivos móviles. Hay que tener muy claro que el **viewport** no se corresponde con el tamaño real de la pantalla en píxeles sino al espacio útil donde se mostrará una página web. Por ejemplo, el primer iphone tenía 230px de ancho, pero su **viewport** simulaba un área de 980px. Lo que quiere decir que encajaba los 980px en los 230px. con lo que el contenido de la web se veía completo pero muy pequeño, obligando al usuario a hacer zoom para simplemente poder leer el texto. Y lo mismo sucede con el resto de dispositivos móviles cambiando las cantidades de **viewport** entre unos y otros.

Para solucionar esto podemos utilizar los siguientes valores en la etiqueta **meta name=”viewport”**:

* + **width**. Define el ancho, en píxeles, del **viewport**. Puede ser un entero positivo o la cadena de texto “**device-width**”. En este último caso se toma el ancho del dispositivo.

Ej: <**meta name=”viewport” content=”width=320”**>, o

<**meta name=”viewport” content=”width=device-width”**>

Para diseñar webs móviles, esta última opción es la mejor pues automáticamente nos va a arrancar la web con el ancho máximo que tenga el dispositivo, sea el que sea.

* + **height**. Define la altura, en píxeles, del **viewport**. Puede ser un entero positivo o la cadena de texto “**device-height**”.
  + **initial-scale**. Define el ratio entre el ancho del dispositivo y el tamaño del **viewport**. Puede ser un número positivo entre 0.0 y 10.0.
  + **maximum-scale**. Define el valor máximo del zoom; este debe ser mayor o igual a la **minimun-scale** o el comportamiento será indeterminado. Puede ser un número positivo entre 0.0 y 10.0.
  + **minimun-scale**. Define el valor mínimo del zoom; este debe ser menor o igual a la **maximun-scale** o el comportamiento será indeterminado. Puede ser un número positivo entre 0.0 y 10.0.

Ej: <**meta name=”viewport” content=”width=device-width, maximun-scale=1, minimun-scale=1 /**>.

Con esta declaración anularíamos el zoom del dispositivo móvil.

* + **user-scalable**. Si tiene el valor “**no**”, el usuario no podrá hacer zoom en la página. El valor por defecto es “**yes**”.

Ej: <**meta name=”viewport” content=”width=device-width, user-scalable=no” /**>.

Produce el mismo efecto que el ejemplo anterior.

***Pero… ¡Veámoslo en la práctica 16.2!***

Para diseñar nuestras web según Responsive Web Design habitualmente utilizaremos este metadato.

También debemos conocer que a la hora de escribir este manual el **W3C** está trabajando en la directiva **@viewport** que anulará la meta viewport transfiriéndola a **CSS**.

***Asegure estos conocimientos realizando la segunda ejercitación del capítulo 16***

DISEÑO WEB FLEXIBLE O FLUIDO

La segunda pata del **Responsive Web Design** es la web flexible o fluida. Para ello el secreto está en diseñar con medidas flexibles, utilizando por tanto unidades en porcentaje, rem o em y olvidando las medidas en px y cm. Siempre trabajando sobre **CSS**.

Esto supone un cambio de mentalidad en el diseñador. Hasta ahora hemos aprendido a diseñar webs para pantallas de ordenador, con unas medidas fijas. A partir de ahora tendremos que pensar en móvil, en varios tamaños de pantalla y en porcentajes. Es más, lo lógico, sería diseñar primero la página web para la pantalla del móvil para luego ir adaptando el diseño a pantallas de sobremesa.

Para diseñar una web flexible debemos de tener en mente estas 10 normas básicas:

* Para cambiar una web con medidas fijas a porcentajes debemos aplicar la fórmula básica del Responsive Web Design que es “**target / contexto**”, es decir **dividir lo que queremos modificar entre su contenedor**.
* Tenemos que sumar y restar para completar el 100% en las diferentes medidas.
* Aplicar también la flexibilidad a las imágenes, videos... para ello utilizaremos, entre otras, la propiedad “**max-width: 100%**”.
* Para la tipografía usar 100% en el body y “em” para el resto de la página. Aunque están surgiendo nuevas unidades como “rem” o “vw” que también pueden ser útiles.
* No olvidar la fórmula básica de las fuentes flexibles: **100% = 1em = 12pt. = 16px**.
* Tener en cuenta que no solamente hay pantallas pequeñas de dispositivos móviles. También hay muchos usuarios que navegan con grandes pantallas o incluso empiezan a proliferar las grandes televisiones con conexión a internet. Para ellos utilizar la propiedad **CSS** “**max-width**” para no pasarse de tamaño.
* Y al contrario, se puede utilizar la propiedad **CSS** “**min-width**” para que un elemento no se haga tan pequeño que no se vea bien.
* No utilizar posiciones absolutas. Si posicionamos un objeto por ejemplo 600px a la derecha, en una pantalla de 400px, dicho objeto quedaría fuera de ella.
* Para posicionar los elementos en la web utilizar la propiedad **CSS** “**display: inline-block**” junto con “**float**”.
* Solo utilizaremos las medidas fijas, por ejemplo, px, para establecer los límites, el tamaño mínimo o máximo, de un elemento.

***Pero… ¡Veámoslo realizando la práctica 16.3!***

LA PROPIEDAD BOX-SIZING

Incluimos esta propiedad **CSS** a estas alturas del curso por su importancia para el **Responsive Web Design**. **Box-sizing** indica al navegador cómo queremos que calcule el espacio total ocupado por un elemento incluyendo o no en el ancho total el valor del padding y del border. Su sintaxis es:

**box-sizing: content-box / border-box;**

A tener en cuenta:

* **content-box:** Es el valor por defecto. La anchura de caja se calcula como ya conocemos: width + padding + marging + border.
* **border-box**: Fuerza al navegador a incluir en el ancho de la caja los valores de las propiedades padding y border. Es decir el padding y el border pasan a formar parte de la anchura de la caja. La anchura de la caja se calcula: width + marging.
* A la hora de escribir este manual hay que utilizar los prefijos privativos con esta propiedad.

Decimos que esta propiedad es importante para el **Diseño Web Adaptativo** porque si establecemos un ancho de 100% para un elemento para que ocupe todo el ancho de su contenedor, al añadir su padding y border, necesariamente se desbordará. Y calcular la anchura del padding y border en porcentaje puede ser bastante complicado. Utilizar el valor “**border-box**” de la propiedad “**box-sizing**” nos simplifica enormemente los cálculos.

***Asegure estos conocimientos realizando la tercera ejercitación del capítulo 16***

LOS MEDIA QUERIES

La tercera pata del **Responsive Web Design** son los **Media Queries**.

Como hemos podido comprobar el diseño fluido a base de porcentajes está muy bien hasta que nos encontramos con anchos de pantalla realmente pequeños como por ejemplo la de un móvil. Llega un momento en que los elementos se descolocan porque no caben y puede acabar todo en un caos. En este caso es probable que tengamos que prescindir de ciertos elementos de la web o situarlos en un lugar diferente. Para ello utilizaremos los **Media Queries**.

Las hojas de estilo **CSS** permiten tener representaciones diferentes según el medio de salida: pantalla, impresora, lector braille, smartphone, etc, etc… Para ello disponemos del atributo “**media**” que se agrega en la declaración del estilo y nos permite diseñar hojas de estilo diferentes según los medios de salida.

La manera de indicar los estilos para cada medio de salida depende de la declaración de estilos:

* **1.- Declaración de estilos interna**. Las propiedades **CSS** se indican en la cabecera del documento **HTML** como en el siguiente ejemplo:

**<head>**

**<style type=”text/css” media=”print”>**

**/\* Reglas de estilo que se aplican al documento impreso \*/**

**</style>**

**<style type=”text/css” media=”screen”>**

**/\* Reglas de estilo que se aplican en la visualización en pantalla \*/**

**</style>**

**</head>**

* **2.- Declaración de estilos externa**. Las propiedades **CSS** se indican en un archivo diferente al del **HTML**, como en el siguiente ejemplo:

**<link rel=”stylesheet” type=”text/css” href=”impresora.css” media=”print”>**, o

**<link rel=”stylesheet” type=”text/css” href=”pantalla.css” media=”screen”>**.

Los diferentes medios de salida que entiende el parámetro “**media**” son:

|  |  |
| --- | --- |
| **media=”all”** | Los estilos definidos se aplican a todos los tipos de medio. Es el valor por defecto. |
| **media=”screen”** | Los estilos se aplican a la visualización en las pantallas. |
| **media=”print”** | Los estilos se aplican a la impresión en papel. |
| **media=”projection”** | Los estilos se aplican a webs vistas mediante proyectores. |
| **media=”handheld”** | Los estilos se aplican a la visualización en ordenadores de bolsillo o smartphones. |
| **media=”speech”** | Los estilos se aplican a sintetizadores de voz. |
| **media=”braille”** | Los estilos se aplican a los lectores braille. |
| **media=”embossed”** | Los estilos se aplican a las impresoras de perforación braille. |
| **Media=”tty”** | Los estilos se aplican cuando se utiliza un dispositivo con un tamaño fijo de carácter, como un teletipo. |
| **Media=”tv”** | Los estilos se aplican cuando se accede a una web desde una televisión. |

En un futuro, necesariamente, se incrementará esta lista, a medida que aparezcan nuevos dispositivos o medios en los que se puedan visualizar páginas webs. Por ejemplo: “**3d-glasses**”.

Llegados a este punto es preciso mencionar que debemos prestar una especial atención al **media=”print”**. Todos hemos comprobado en alguna ocasión que al imprimir una web, lo que hemos obtenido en papel, era bastante decepcionante: fondos negros que nos gastan mucha tinta, menús de navegación inútiles en papel, tipos de letra ilegibles, texto cortado porque no cabe en la página o papel echado a perder…

Por ello consideramos importante crear reglas de estilo solo para la versión impresa de nuestras webs que tengan en cuenta entre otras cosas:

* Eliminar los elementos inútiles o molestos a la hora de imprimir la página.
* Repasar el formato del contenido de texto pensando en su lectura en papel.
* Agregar información, habitualmente reservada a la visualización en pantalla.
* Controlar los fondos de colores o de imágenes.
* Controlar el color, el tipo y tamaño de la letra.
* Controlar los enlaces.
* Controlar las imágenes.
* Comprobar si es necesario el menú de navegación.
* Quitar las animaciones.
* Controlar si merece la pena que se impriman los formularios.
* Controlar los márgenes
* Etc, etc…

Debemos tener en cuenta que una página web en pantalla debe resultar atractiva al ojo mientras que una información en papel debe ser atractiva al espíritu. En una página impresa se busca riqueza de información o contenido.

A modo de enumeración, algunas propiedades **CSS** relacionadas con la impresión son:

|  |  |  |
| --- | --- | --- |
| **Propiedad** | **Utilidad** | **Valores** |
| Page-break-before | Fuerza un salto de página antes de algún elemento de la página. | Always / avoid / auto |
| Page-break-after | Fuerza un salto de página después de algún elemento de la página. | Always / avoid /auto |
| Page-break-inside | Evita el salto de página en el elemento designado | Always/ avoid /auto |
| Selector: @page {  /\* propiedades para imprimir \*/  }  @page :first {  /\* propiedades para la primera página \*/  } | Permite especificar diferentes propiedades para un medio impreso. Se debe definir al principio del **CSS**. Además de las propiedades habituales son propias de este selector:   * Size. * Size: portrait * Size: landscape * Y las pseudoclases: :first, :left, y :right para indicar primera página, páginas izquierdas o páginas derechas. |  |

* **3.- Declaración de estilos con la regla @media.**

Las reglas **@media**, también conocidas como **Media Queries**, implementadas en **CSS3**, nos permiten especificar el conjunto de reglas **CSS** que deben aplicarse a cada medio de salida pero condicionados a unos determinados criterios.

Las reglas **@media** son de verdadera utilidad para el **Responsive Web Design** ya que podemos especificar que el medio de salida sea la pantalla pero además de un determinado tamaño.

La sintaxis de la regla **@media** es:

**@media medio\_de\_salida and (criterio) {**

**/\* Reglas CSS que se desean aplicar cuando**

**se cumpla ese medio\_de\_salida y el (criterio) \*/**

**}**

A tener en cuenta:

* **Medio\_de\_salida**, puede ser cualquiera de los que hemos indicado más arriba: all, print, screen, etc, etc…
* **Criterio**, indica una propiedad del medio de salida que lo especifica. Estas propiedades pueden ser de los siguientes tipos:
  + **Aspect-ratio**, especifica el dispositivo por la relación de aspecto. Ej: 16:9. Las propiedades de este tipo son:
    - Aspect-ratio
    - Max-aspect-ratio
    - Min-aspect-ratio
    - Device-aspect-ratio
    - Max-device-aspect-ratio
    - Min-device-aspect-ratio
  + **Width** y **Height**, especifica el dispositivo por el ancho o el alto de su área de visualización. Las propiedades de este tipo son:
    - Device-height
    - Max-device-height.
    - Min-device-height.
    - Device-width.
    - Max-device-width.
    - Min-device-width.
    - Height.
    - Max-height.
    - Min-height.
    - Width.
    - Max-wdith.
    - Min-width.
  + **Orientation**, especifica el dispositivo si está en vertical u horizontal. Las propiedades de este tipo son:
    - Orientation portrait.
    - Orientation landscape.
  + **Resolution**, especifica el dispositivo por su densidad de píxeles. Las propiedades de este tipo son:
    - Resolution.
    - Max-resolution.
    - Min-resolution.
  + **Color, color-index** y **monochrome**, especifica el dispositivo de salida por la cantidad de colores que admite. Las propiedades de este tipo son:
    - Color
    - Max-color.
    - Min-color.
    - Color-index.
    - Max-color-index.
    - Min-color-index.
    - Monochrome
    - Max-monochrome.
    - Min-monochrome.

Aunque a primera vista pueda parecer lo mismo, hemos de distinguir entre la resolución de pantalla y el ancho de ventana del navegador. Para entenderlo piense que en un ordenador podemos tener una resolución de pantalla de 1980px de ancho pero si reducimos la ventana, el ancho de ventana del navegador es menos que la resolución de pantalla.

Los media queries actúan por defecto sobre la vista del navegador con la excepción de aquellos que especifican en su nombre la palabra “**device**” que actúan sobre toda la ventana del dispositivo.

Algunos ejemplos de sintaxis de la regla **@media** son:

* Para una pantalla de móvil de 320px:

**@media screen and (max-width: 320px) {**

**/\* Reglas CSS que se desean aplicar para este tamaño de pantalla \*/**

**}**

* Para una pantalla de ancho superior a 1.200px, la regla podía ser:

**@media screen and (min-width: 1200px) {**

**/\* Reglas CSS que se desean aplicar para este tamaño de pantalla \*/**

**}**

* Un poco más complicado… para maquetar para una famosa tablet en su posición horizontal podría ser:

**@media only screen and (min-device-width: 768px) and (max-device-width: 1024px) and (orientation: landscape) {**

**/\* Reglas CSS que se desean aplicar para este tamaño de pantalla \*/**

**}**

***Pero… ¡Pasemos a la práctica realizando la número 16.4!***

El problema que se nos plantea es la infinidad de tamaños de pantalla que hay en la actualidad y más teniendo en cuenta que pueden utilizarse en vertical u horizontal. Nosotros hemos tenido en cuenta una sola media querie en nuestra práctica. Intentar abarcar todos los dispositivos es una locura. Algunos autores recomiendan optimizar las webs para estos anchos de pantalla:

* 320px
* 480px
* 600px
* 768px
* 900px
* 1200px

Pero lo habitual es maquetar para tres tipos de pantalla:

* Móviles en vertical: donde la pantalla solo permite lectura y navegación muy simplificada y podemos incluirla entre 320px y 400px.
* Móviles en horizontal y tablets: donde ya se permite una visualización horizontal clásica con pantallas de 400px, 600px y hasta 800px.
* Dispositivos de sobremesa: donde ya contamos con webs normales mayores de 800px.

Es preferible utilizar la regla **@media** a la declaración de estilos externa ya que así mejoramos el rendimiento de la página al no tener que cargar archivos **CSS** adicionales para los diferentes dispositivos.

FONT-SIZE EN FUNCIÓN DEL TAMAÑO DE LA VENTANA.

Recientemente el **W3C** ha añadido algunas unidades nuevas para poder dar tamaño a los textos en función de las medidas de su caja contenedora inicial o del tamaño de la ventana. Con ello conseguimos que las fuentes vayan cambiando de tamaño en función del tamaño de la ventana. Estas unidades, conocidas como unidades viewport, son:

* **vw**: Igual al 1% de la anchura de la pantalla.
* **vh**: igual al 1% de la altura de la pantalla.
* **vmin**: Igual a la más pequeña de “vw” o “vh”.
* **vmax**: Igual a la más grande de “vw” o “vh”.

***¡Para ver el resultado realice la práctica 16.4!***

De estas tres unidades “**vw**” es la más factible para ser utilizada en el **Responsive Web Design**.

También debemos comentar que a la hora de realizar este manual, el **W3C** está trabajando un una nueva etiqueta **HTML5** llamada <**picture**> que permita mostrar imágenes diferentes en función del tipo de pantalla en que se visualice. Esto permitirá incluir imágenes con mayor o menor peso adaptándose a los diferentes dispositivos móviles y sacar un mayor partido de las nuevas pantallas de alta resolución “retina display”.

***Asegure estos conocimientos realizando la cuarta ejercitación del capítulo 16***

CAPITULO 17.- PUBLICACIÓN DE UNA WEB

Y OTROS TEMAS

CONTADORES DE VISITAS

Insertar un contador de vistas en nuestra web es muy sencillo. Contamos con dos opciones:

1) Utilizar el sistema de contadores del proveedor de alojamiento de la página web. Generalmente los proveedores de alojamiento ofrecen alguna posibilidad para insertarlos, en este caso, lo lógico es seguir sus instrucciones.

2) Utilizar un servicio de contadores gratuitos. Por ejemplo en <http://www.websmultimedia.com/contador-de-visitas-gratis> encontrará contadores gratis.

En los dos casos lo que hay que hacer es colocar el código **HTML** que proporciona el programa del contador. Habitualmente se trata de una etiqueta <**img**> cuyo **SRC** se enlaza con el programa contador.

***¡Realice ahora la práctica 17.1!***

AÑADIR UN FAVICON

Un “**favicon**”, también conocido como “icono de página” es una pequeña imagen que representa una página o un sitio web. Es como el logotipo de cada web. Se suele mostrar en la barra de direcciones del navegador y a la izquierda del nombre de la web en la barra de título o pestaña de la ventana del navegador.

El **favicon** también se utiliza para identificar las páginas en las listas de marcadores o favoritos y en el historial de páginas visitadas.

Para añadir un **favicon** se necesita una imagen de 16x16 ó 32x32 píxeles con formato .png, .gif o .ico y añadir en el <**head**> de la página el siguiente código:

**<link rel="icon" type="image/png" href=”n\_del\_icono.png" >**

***¡Veámoslo en la práctica 17.2!***

ATRIBUTOS GLOBALES DE HTML5

A lo largo de este curso, cuando hemos ido estudiando cada elemento **HTML**, hemos ido indicando sus atributos, los que le son propios. Hay una serie de atributos, llamados atributos globales, que están disponibles para todos los elementos **HTML** y que, aunque algunos ya los hemos utilizado, debería conocer. Son los siguientes:

|  |  |
| --- | --- |
| **Atributo** | **Descripción** |
| **accesskey** | Especifica una tecla de acceso directo para activar/hacer focus en un elemento. |
| **class** | Especifica el nombre de la clase de un elemento para aplicar estilos css. |
| **contenteditable** | Especifica si el contenido de un elemento es editable o no. |
| **contextmenu** | Muestra  un menú contextual al hacer clic con el botón derecho del ratón. |
| **dir** | Especifica la dirección del texto contenido en un elemento. |
| **draggable** | Indica si un elemento es arrastrable o no. |
| **dropzone** | Especifica si los datos arrastrados son copiados, movidos o vinculados, cuando se dejan caer sobre un elemento. |
| **hidden** | Indica si el elemento es visible o no. |
| **id** | Identificador único del elemento en la página. |
| **lang** | Especifica el idioma del contenido de un elemento. |
| **spellcheck** | Especifica si el elemento debe tener su ortografía y gramática comprobada o no. |
| **style** | Aplica un estilo css al elemento. |
| **tabindex** | Especifica el orden de tabulación de un elemento. |
| **title** | Especifica información adicional acerca de un elemento. |
| **translate** | Indica si un elemento se traduce o no. (Es para evitar que código de programación o similares sean traducidos automáticamente con el resto de la página). |
| **data-\*** | Permite añadir atributos definidos por el usuario.  Por ejemplo: **<p data-calorias=”50”>kiwi</p>** |

ATRIBUTOS DE EVENTOS

Los eventos de **HTML** permiten a los desarrolladores web agregar interactividad entre el sitio web y el visitante ejecutando un programa, generalmente en **JavaScript**, cuando el visitante de la web realiza una acción como por ejemplo hacer doble clic sobre un elemento o pasar con el puntero del ratón sobre un elemento. Esto último lo hemos realizado con la pseudoclase “**:hover**” de **CSS** pero lo interesante de los atributos de eventos es que son mucho más numerosos y que podemos aplicarles verdaderos programas.

Hemos utilizado atributos de eventos como, “**ondragstart**”, “**ondragover**” u “**ondrop**”, cuando creamos el puzle en la web “**t-rural.htm**”. Como pudimos ver entonces, los atributos de eventos se indican como el resto de atributos de los elementos **HTML** siguiendo el formato:

**<etiqueta atributo\_de\_evento="código JavaScript">**

A modo de ejemplo en el siguiente código, definimos un párrafo que cambia el color de su texto a rojo cuando pasamos por encima con el puntero del ratón, y lo devuelve a negro cuando lo retiramos:

**<p onmouseover="this.style.color='red'" onmouseout="this.style.color='black'">**

**Este es el texto que cambiaría de color al pasar el ratón.**

**</p>**

Sacaremos el máximo partido a los atributos de eventos de **HTML** cuando dominemos **JavaScript** pero para que nos vayan sonando, estos son los atributos de eventos en **HTML5**:

|  |  |
| --- | --- |
| **Atributos de Eventos globales y de ventana.** | |
| **Atributo de Evento** | **Descripción** |
| **onafterprint** | Se ejecutará un script después de imprimirse el documento. |
| **onbeforeprint** | Se ejecutará un script antes de imprimir el documento. |
| **onbeforeunload** | Se ejecutará un script antes de que se cargue el documento. |
| **onerror** | Se ejecutará un script cuando se produzca un error. |
| **onhaschange** | Se ejecutará un script cuando el documento haya cambiado. |
| **onload** | Se ejecutará un script después de que la página termine de cargarse. |
| **onmessage** | Se ejecutará un script cuando se active el mensaje. |
| **onoffline** | Se ejecutará un script cuando el documento está desconectado. |
| **ononline** | Se ejecutará un script cuando el documento se ponga en línea. |
| **onpagehide** | Se ejecutará un script cuando la ventana está oculta. |
| **onpageshow** | Se ejecutará un script cuando la ventana se haga visible. |
| **onpopstate** | Se ejecutará un script cada vez que la entrada del historial cambie. |
| **onredo** | Se ejecutará un script cuando el documento realiza un “rehacer”. |
| **onresize** | Se ejecutará un script cuando se redimensiona la ventana del navegador. |
| **onstorage** | Se ejecutará un script cuando se actualiza el almacenamiento web. |
| **onundo** | Se ejecutará un script cuando el documento realiza un “deshacer”. |
| **onunload** | Se ejecutará un script cuando el usuario abandone el documento o la ventana del navegador se ha cerrado. |

|  |  |
| --- | --- |
| **Eventos de Formulario** | |
| **Atributo de Evento** | **Descripción** |
| **onblur** | Se ejecutará un script en el momento en que el elemento pierde el foco. |
| **onchange** | Se ejecutará un script cuando se cambia el valor del elemento. |
| **oncontextmenu** | Se ejecutará un script cuando se activa el menú contextual. |
| **onfocus** | Se ejecutará un script cuando el elemento obtiene el foco. |
| **onformchange** | Se ejecutará un script cuando un formulario cambie. |
| **onforminput** | Se ejecutará un script cuando el usuario escribe en un formulario. |
| **oninput** | Se ejecutará un script cuando un usuario escribe en un elemento. |
| **oninvalid** | Se ejecutará un script cuando un elemento no es válido. |
| **onselect** | Se ejecutará un script después de que un fragmento de texto haya sido seleccionado en un elemento. |
| **onsubmit** | Se ejecutará un script cuando se envía un formulario. |

|  |  |
| --- | --- |
| **Eventos de teclado** | |
| **Atributo de Evento** | **Descripción** |
| **onkeydown** | Se ejecutará un script cuando el usuario está presionando una tecla. |
| **onkeypress** | Se ejecutará un script cuando el usuario pulsa una tecla. |
| **onkeyup** | Se ejecutará un script cuando el usuario suelte una tecla. |
| **Eventos del mouse** | |
| **Atributo de Evento** | **Descripción** |
| **onclick** | Se ejecutará un script cuando se haga clic con el ratón en el elemento. |
| **ondblclick** | Se ejecutará un script cuando el ratón haga doble clic en el elemento. |
| **ondrag** | Se ejecutará un script cuando se arrastre un elemento. |
| **ondragend** | Se ejecutará un script al final de una operación de arrastre. |
| **ondragenter** | Se ejecutará un script cuando un elemento ha sido arrastrado a un destino válido. |
| **ondragleave** | Se ejecutará un script cuando un elemento deja un destino válido. |
| **ondragover** | Se ejecutará un script cuando un elemento está siendo arrastrado por un destino válido. |
| **ondragstart** | Se ejecutará un script en el inicio de una operación de arrastre. |
| **ondrop** | Se ejecutará un script cuando se suelte el elemento arrastrado. |
| **onmousedown** | Se ejecutará un script cuando se pulse un botón del ratón sobre un elemento. |
| **onmousemove** | Se ejecutará un script cuando el usuario mueve el ratón sobre un elemento. |
| **onmouseout** | Se ejecutará un script cuando el puntero del ratón se mueva fuera de un elemento. |
| **onmouseover** | Se ejecutará un script cuando el puntero del ratón se sitúa sobre un elemento. |
| **onmouseup** | Se ejecutará un script cuando se suelta el botón del ratón sobre un elemento. |
| **onmousewheel** | Se ejecutará un script cuando se gira la rueda del ratón. |
| **onscroll** | Se ejecutará un script cuando se desplaza la barra de desplazamiento de un elemento. |

|  |  |
| --- | --- |
| **Eventos con elementos Multimedia** | |
| **Atributo de Evento** | **Descripción** |
| **OnAbort** | Se ejecutará un script al abortar la ejecución multimedia. |
| **oncanplay** | Se ejecutará un script cuando el archivo multimedia este listo para comenzar a reproducirse. |
| **oncanplaythrough** | Se ejecutará un script cuando un archivo multimedia se reproduzca entero hasta el final sin detenerse para cargar partes del archivo. |
| **ondurationchange** | Se ejecutará un script cuando se cambie la longitud del multimedia. |
| **onemptiedv** | Se ejecutará un script cuando de repente el archivo multimedia no es accesible por errores de red o de carga… |
| **onended** | Se ejecutará un script cuando la reproducción llega al final. (Un evento útil para los mensajes como "gracias por escucharme"). |
| **onerror** | Se ejecutará un script cuando se produzca un error al cargar el archivo. |
| **onloadeddata** | Se ejecutará un script cuando se carguen los datos de archivos multimedia. |
| **onloadedmetadata** | Se ejecutará un script cuando se cargan los metadatos de un archivo multimedia como dimensiones, duración, estilo, autor... |
| **onloadstart** | Se ejecutará un script cuando el navegador empiece a cargar los datos del archivo multimedia. |
| **onpause** | Se ejecutará un script cuando la reproducción multimedia este en pausa ya sea por el usuario o mediante programación. |
| **onplay** | Se ejecutará un script cuando la reproducción multimedia esté lista para comenzar. |
| **onplaying** | Se ejecutará un script cuando realmente haya empezado la reproducción del archivo multimedia. |
| **onprogress** | Se ejecutará un script cuando el navegador este obteniendo los datos del archivo multimedia. |
| **onratechange** | Se ejecutará un script cada vez que se cambie la velocidad de reproducción como cuando un usuario cambia a un modo de cámara lenta o avance rápido. |
| **onreadystatechange** | Se ejecutará un script cada vez que el archivo multimedia cambia al estado “listo”. |
| **onseeked** | Se ejecutará un script cuando el atributo de búsqueda se establece en “false”. Indica que la búsqueda ha terminado |
| **onseeking** | Se ejecutará un script cuando el atributo de búsqueda se establece en “true”. Indica que la búsqueda está activa. |
| **onstalled** | Se ejecutará un script cuando el navegador no sea capaz de recuperar los datos de los archivos multimedia por cualquier motivo. |
| **onsuspend** | Se ejecutará un script cuando se detiene el recuperar los datos de los archivos multimedia por cualquier razón antes de que estén completamente cargados. |
| **ontimeupdate** | Se ejecutará un script cuando la posición de reproducción haya cambiado como cuando el usuario avanza rápidamente a un punto diferente del archivo multimedia. |
| **Onvolumechangev** | Se ejecutará un script cada vez que se cambie el volumen. Incluido poner el volumen en “silencio”. |
| **onwaiting** | Se ejecutará un script cuando la reproducción del multimedia se ha detenido pero se espera que reanude como cuando la reproducción hace una pausa para cargar más datos. |

SPRITES CSS

 **CSS Sprites** es una técnica de diseño que consiste en utilizar un solo archivo de imagen compuesto por todas las imágenes que utilizamos en la web. Esta única imagen se va posicionando haciendo uso de las propiedades “**background-position**” de **CSS** para mostrar sólo la imagen que nos interesa en cada parte de la web.

A la izquierda mostramos uno de los **Sprite CSS** utilizado por Google que lo ha ido cambiando, según sus necesidades, a lo largo de los años.

La principal ventaja de la técnica de los **CSS Sprites** es la mejora en la velocidad de descarga de la web pues no es lo mismo bajar 15 archivos de imágenes que una sola imagen que vamos posicionando. Además sólo se hace una petición al servidor en vez de 15.

Ultimamente, con la llegada del Responsive Web Desing y su tendencia hacia el “diseño flat”, el uso de los **Sprites CSS** se está cuestionando y los diseñadores web optan por mostrar iconos mediante fuentes tipográficas como las que puede ver en <http://css-tricks.com/flat-icons-icon-fonts/>

PUBLICAR NUESTRA PÁGINA WEB

Para publicar nuestra web en Internet necesitamos:

1.- Un dominio que puede ser de pago como [www.academiafrayluis.es](http://www.academiafrayluis.es) o gratuito, en este caso, lo normal es que sea un subdominio tal como [www.alumno1.academiafrayluis.es](http://www.alumno1.academiafrayluis.es). Si nuestra web es profesional lo lógico es adquirir un dominio propio que normalmente se paga mediante anualidades. Para practicar nosotros emplearemos un subdominio gratuito.

2.- Un proveedor de hosting donde alojar los archivos que componen nuestra web. Al igual que en el caso anterior, podemos utilizar un hosting de pago o uno gratuito. Lógicamente un hosting de pago tendrá más servicios y sobretodo más capacidad de almacenamiento. Para practicar nosotros utilizaremos el hosting gratuito de la empresa **HOSTINGER**.

3.- Un programa FTP que nos permite trasladar todos los archivos de la web desde nuestro disco duro local al del proveedor del hosting. Nosotros utilizaremos “**Filezilla**” que es uno de los más habituales en todos los proveedores hosting. Esta transferencia de archivos se realiza en **modo autentificado**, es decir, introduciendo un nombre de usuario y contraseña para evitar que otras personas puedan publicar en nuestro espacio web.

***¡Vayamos a la práctica realizando la número 17.3!***

RECOPILACIÓN DE BUENAS PRÁCTICAS CON HTML Y CSS

A modo de enumeración esta es una recopilación de algunas buenas prácticas:

Para **HTML**:

* Recordar cerrar las etiquetas.
* Nombrar las etiquetas siempre en minúsculas.
* Utilizar ficheros externos para **CSS** y **JavaScript**.
* Enlazar los **CSS** externos al principio del documento en su <**head**>.
* Enlazar los **JavaScript** externos al final del documento justo antes de la etiqueta <**/body**>.
* Ser cuidadoso con la compatibilidad de los elementos que usemos. Usar **www.caniuse.com**.
* Usar el atributo “**alt**” en todas las imágenes.
* Validar el código.
* Usar una “**chuletilla**” con las etiquetas **HTML5**.

Para **CSS**:

* Primero definir los estilos para etiquetas **HTML** como **header**, **footer**, **div**, **a**, **p**, etc.
* Después definir los estilos para indentificadores “**id**”.
* Por último definir los estilos para las clases “**class**”.
* Ordenar las reglas de estilos conforme van apareciendo en el **HTML**.
* Ordenar las propiedades en cada regla de estilo por orden alfabético.
* Utilizar tabulaciones en las propiedades **CSS** para facilitar su lectura.
* Especificar el alto y ancho de las imágnes.
* Definir un tamaño de fuente global en el body.
* Evitar el uso de muchos selectores descendentes como: **section div ul li a { … }**.
* Reducir el uso de selectores **CSS3** como **:first-child**, facilitan mucho la vida pero consumen muchos recursos.
* Aprovechar las ventajas en cascada de **CSS**. No es necesario escribir: **nav ul li a {Font-weight: 600;}**. Nos vale lo mismo con: **nav {Font-weight: 600;}**.
* Recordar poner las reglas de estilo de reset antes de cualquier otra regla de estilo.
* Comprobar el diseño en varios navegadores.
* Utilizar comentarios para separar las secciones de la hoja de estilos.
* Reduccir el número de archivos **CSS** de la página. Si es posible, combinar todos los archivos **CSS** individuales en un único archivo.
* Una vez que hemos asegurado el acceso completo y correcto a toda la información de la página independientemente del navegador utilizado por el usuario, utilizar las características más modernas de **CSS** para que se puedan disfrutar en los navegadores más modernos.
* Usar herramientas como **Firebug** en Firefox para el depurado de errores **CSS**.

***Asegure estos conocimientos realizando la ejercitación del capítulo 17***

***Demuestre todo lo que ha aprendido realizando la ejercitación final del curso.***