Programador Web – Nivel 1

Unidad 6: Desarrollo web multiplataforma

Contacto: consultas@elearning-total.com
Web: www.elearning-total.com

Indice

Unidad 6: Desarrollo web multiplataforma

Objetivos

Que el alumno logre:

- Manejar los nuevos elementos incorporados a CSS3 para crear sitios multiplataforma.

Propiedades de Texto

En el proceso de desarrollo del diseño web, una de las grandes limitaciones con las que nos cruzamos tienen que ver con las posibilidades de trabajar con bloques de texto.

CSS3 hace incapié en este problema e implementa algunas propiedades específicas para el manejo de texto, como ser la posibilidad de utilizar cualquier familia tipográfica, de separar el texto en columnas o de cortar largas palabras de texto.

PROPIEDAD @FONT-FACE

@font-face aparece en el nivel 2 de CSS, pero se asienta como estándar en el nivel 3 de CSS.

Esta propiedad básicamente nos permite utilizar en nuestra web cualquier familia tipográfica, sin importar si el usuario la tiene o no instalada en su pc.

Esto se debe a que a través de este estilo, lo que hacemos es indicarle al navegador que no busque la tipografía en la máquina del usuario, sino que lo haga en nuestro servidor.

La implementación de *@font-face* es la siguiente:

```
@font-face{
  font-family:<nombre_fuente>; /*El nombre con el que nos referiremos a la fuente*/
  src: <source>[<ruta de acceso a la fuente>],
/*La ruta de donde cargamos el archivo del tipo*/
  [format:<formato>];
/*Formato de la fuente*/
}
```

Veamos un ejemplo de implementación:

```
@font-face{
        font-family: 'dospuntocero';
        src:url('fuente/duepuntozero_regular.ttf') format('truetype');
    }
    #ejemplo{
```

```
    font-family: 'dospuntocero';
}
```

Podemos visualizar este ejemplo en fuente.html

Una cuestión que provoca algunos problemas con el uso de esta característica es que aún no existe un estándar definitivo de fuentes aceptado por todos los navegadores y plataformas. Po esta razón, y para lograr mayor compatibilidad, hay que incluir las fuentes en diferentes formatos, para que los usuarios puedan verlas correctamente.

Por ejemplo:

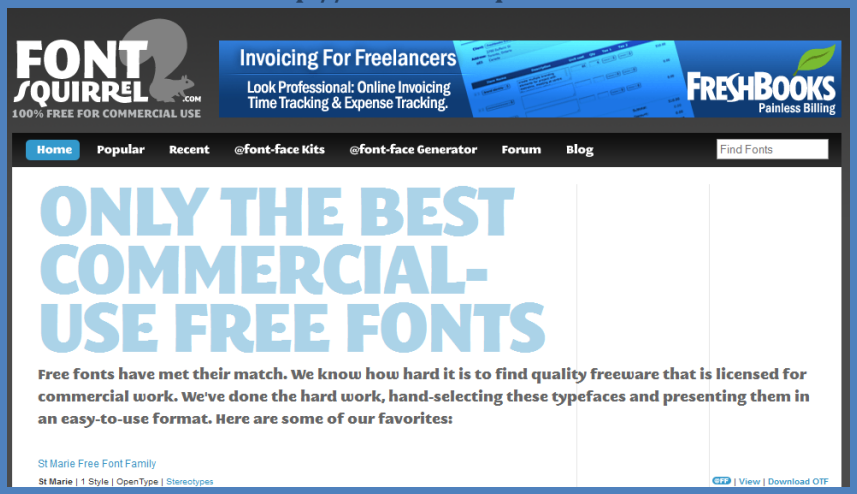
```
src:  url('Delicious-Roman.woff') format('woff'), url('Delicious-
Roman.ttf') format('truetype'), url('Delicious-Roman.svg') format('svg');
```

Los formatos de fuente que pueden utilizarse con esta característica son:

- Truetype,
- OpenType
- Embedded OpenType
- WebOpen
- FontFormat (WOFF)

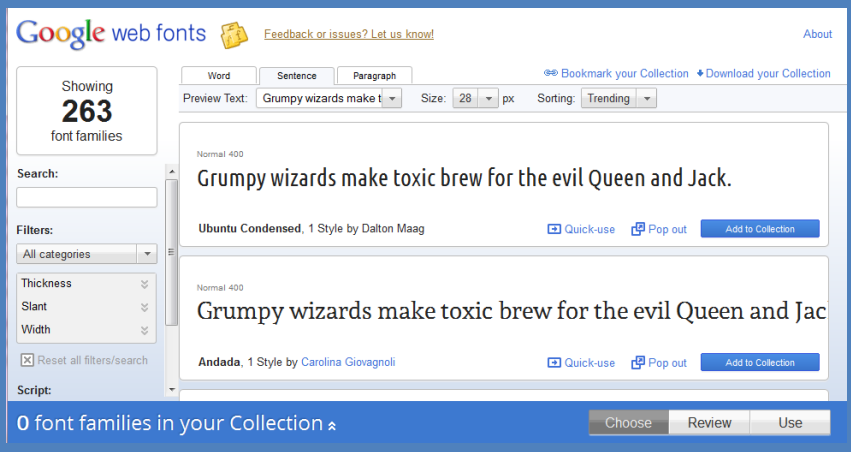
Por sus características de compresión y por estar pensaba especialmente para la web, se espera que WOFF se convierta en el estándar de tipografías para internet. Este formato se encuentra apoyado por el W3C (www.w3.org/TR/WOOF).

NOTA: un buen sitio en donde se pueden bajar Fuentes de uso gratuito en todos los formatos es: **http://www.fontsquirrel.com**



Otra opción es trabajar con el **API de Google**, esto nos evitará cargar la tipografía en nuestro servidor y nos dará la certeza que la tipografía elegida está en todos los formatos necesarios para lograr la mayor compatibilidad posible.

Podemos encontrarlo en **WWW.GOOGLE.COM/WEBFONTS**



PROPIEDAD WORD-WRAP

La propiedad ***word-wrap*** nos sirve para romper las palabras que son demasiado largas y no caben enteras por la anchura de una caja.

En el modelo de caja planteado desde html para los navegadores, las palabras se van ubicando en líneas para para cubrir el ancho disponible de la caja. Cuando tenemos palabras muy largas que no caben en nuestro contenedor, esto puede ser un problema. Dependiendo las propiedades de la caja contenedora del texto, la palabra saldrá fuera de la caja, o la caja se adaptará a la palabra, haciendo que se amplíe el ancho de la caja, esto en algunos casos puede alterar el formato de nuestro diseño (si teníamos div con posicionamiento flotante, probablemente caigan hacia abajo al redimensionar la caja de texto). En cualquier caso, el efecto resultante suele ser poco agradable, porque muchas veces nos descuadra nuestro diseño y hace que las páginas queden mal maquetadas.

Para evitar este efecto, en CSS 3 se ha incluido un atributo llamado ***word-wrap*** que sirve para especificar que las palabras que sean demasiado largas se deben cortar, de manera que quepan en el ancho disponible de la caja. Una propiedad muy útil cuando tenemos grandes bloques de texto dentro de nuestro contenido.

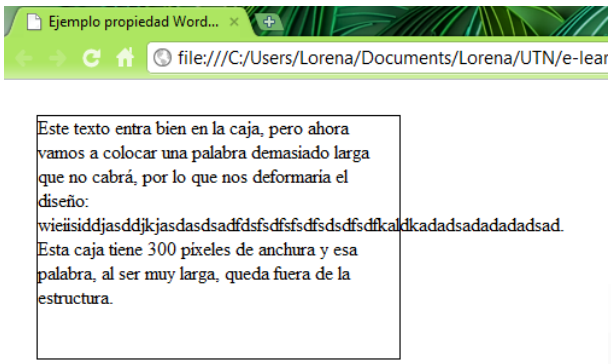
El atributo word-wrap tiene dos posibles valores: ***normal*** o ***break-word***.

- **word-wrap: normal;**

Hace que las palabras no se corten, lo que sería el comportamiento normal que conocíamos hasta ahora, ocurriendo que las palabras largas nos puedan descuadrar nuestro diseño.

Ahora podemos ver una caja que tenía una anchura de 300 px y que por culpa de una palabra muy larga se deforma la caja o el texto aparece por fuera.

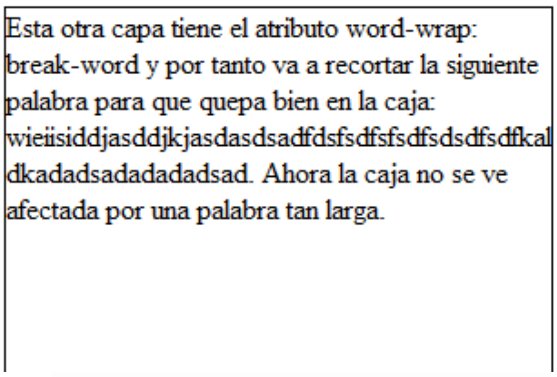
- **word-wrap: break-word;**



Vista previa en Google Chrome

Con este otro valor de ***word-wrap: break-word***, lo que conseguimos es que la palabra se recorte, para caber en el ancho que habíamos definido.

Este atributo por ahora es soportado por la mayoría de los navegadores (Internet Explorer, Safari, Firefox, Opera y Google Chrome), por lo tanto es una propiedad que ya puede comenzar a implementarse.



Vista previa en Firefox

TEXTOS MULTICOLUMNA

Numerosas publicaciones maquetan el texto en diversas columnas, como criterio de diseño y para facilitar la lectura. Podríamos imaginarnos el texto de los periódicos si no se encontrase dividido en diversas columnas ¿no sería casi imposible seguir las líneas para hacer una lectura cómoda de la información si todo estuviera en una única columna?

Este problema no lo encontramos generalmente en las páginas web, porque el texto que cabe en el cuerpo de la página no es tan grande como el que cabría en una hoja de un diario. Además, generalmente la propia página ya se encuentra dividida en diversas columnas. No obstante, aunque quisiéramos, con las características de CSS 2 no sería muy fácil hacer un texto fluído que se adaptase automáticamente en columnas, de modo que éstas crecieran homogéneamente a medida que el texto crece o se reduce. CSS 3 soluciona este problema, o más bien ofrecerá una solución al mismo, sencilla y automática.

Para crear una ***estructura multi-columna*** utilizaremos varios atributos, que servirán para modelizar las columnas:

colum-count: permite indicar en cuantas columnas se dividirá el contenido de un elemento contenedor.

colum-width: servirá para definir la anchura de las distintas columnas a crear.

column-gap: nos permitirá definir el espacio en blanco entre columnas.

column-rule: servirá para crear un filete o línea divisoria entre las columnas.

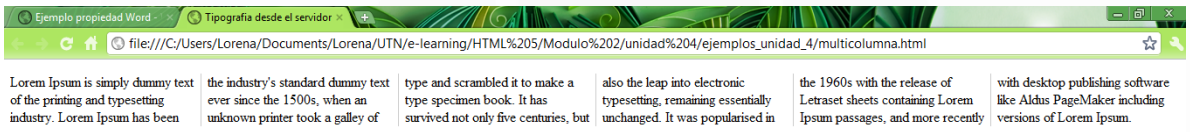
Estas etiquetas por el momento son soportadas solo por Opera. Sin embargo Safari, Google Chrome y Firefox ya implementan el multicolumna, de manera experimental y hasta que las especificaciones de CSS 3 estén dispuestas para incorporar en los navegadores. Para ello, podemos utilizarlas si colocamos los prefijos para cada navegador ("-moz-" en el caso de Firefox y "-webkit-" para el navegador Safari o Chrome.)

Un ejemplo de multicolumna, para que funcione en estos navegadores sería:

```
.multicolumna{
   -moz-column-width: 200px;
```

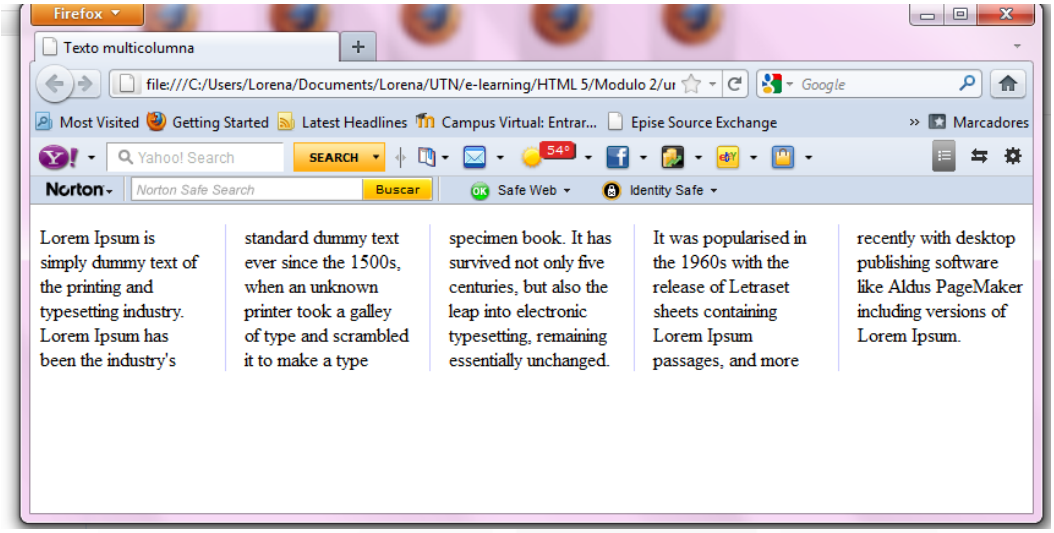
```
    -moz-column-gap: 15px;
    -moz-column-rule: 1px solid #ccc;
    -webkit-column-width: 200px;
    -webkit-column-gap: 15px;
    -webkit-column-rule: 1px solid #ccc;
    column-width: 200px;
    column-gap: 15px;
    column-rule: 1px solid #ccc;

  }
```

Visualización del ejemplo (multicolumna.html) desde Google Chrome:



Otra posibilidad para crear un multicolumna será definir simplemente el número de columnas que querríamos incorporar en el texto, por medio del atributo column-count, de esta manera:

```
  .multicolumna5columnas{
    -moz-column-count: 5;
    -moz-column-gap: 2em;
    -moz-column-rule: 1px solid #ccf;
    -webkit-column-count: 5;
    -webkit-column-gap: 2em;
    -webkit-column-rule: 1px solid #ccf;
    column-count: 5;
    column-gap: 2em;
    column-rule: 1px solid #ccf;
  }
```



Especificar el número de columnas es quizás más cómodo, pero en páginas fluidas puede funcionar peor, porque no se sepa con certeza qué anchura va a tener el lugar donde se muestran los textos y por tanto unas veces queden columnas muy anchas y otras muy estrechas.

El sistema de múltiples columnas se está encuentra en estado de borrador y forma parte de un módulo aparte dentro de las especificaciones de CSS 3. Podemos encontrar más información sobre el tema en la W3C desde la URL: http://www.w3.org/TR/css3-multicol/

TEXTOS CON SOMBRA

El atributo ***text-shadow*** se incluyó inicialmente en el CSS2 pero fué eliminado en CSS2.1. Renovado y con mayor compatibilidad con los navegadores, CSS3 vuelve a incluirlo. Este atributo es similar al atributo box-shadow que vimos en la unidad anterior. La diferencia es que, mientras que box-shadow crea sombras a en forma de caja, text-shadow realiza una sombra ajustada a los propios caracteres de un texto.

La sintaxis de *text-shadow* es igual a la de *box-shadow*:

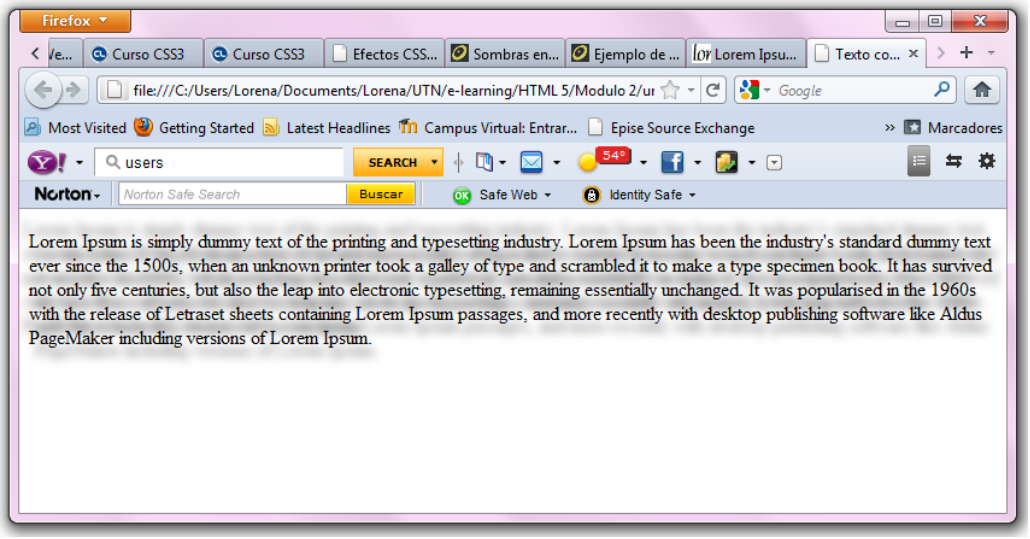
- **text-shadow: distanciaX distanciaY difuminado color;**

Por ejemplo:

```
text-shadow: 5px 10px 7px rgba(0,0,0,0.5);
```

Para asignar varias sombras usamos la coma como separador. Por ejemplo:

```
text-shadow: 5px 10px 7px rgba(0,0,0,0.5), -5px -10px 7px rgba(0,0,0,0.5);
```



TEXTOS EN 3D CON TEXT-SHADOW

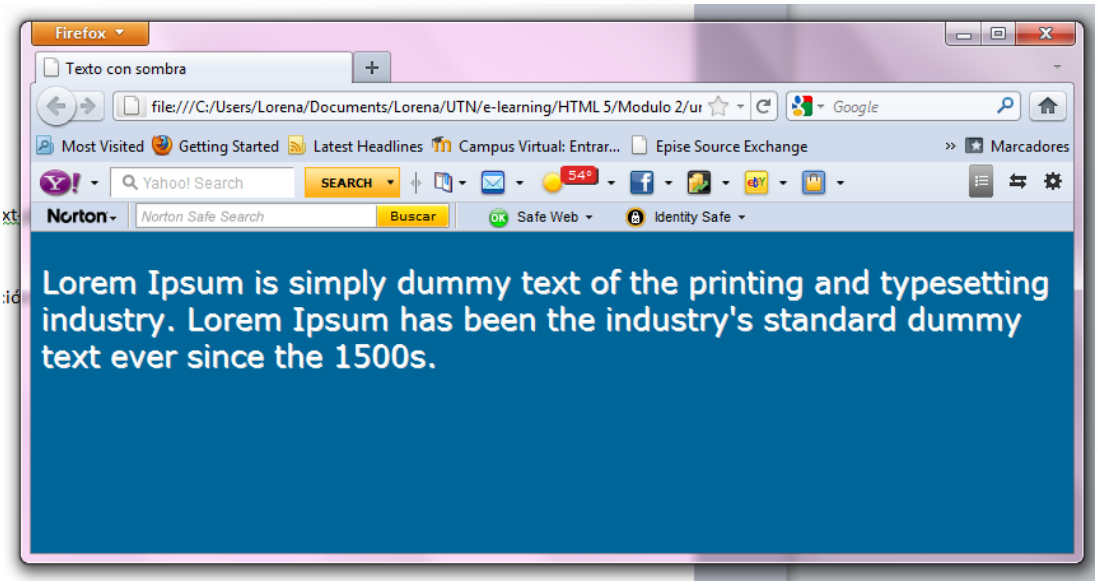
Veamos una aplicación práctica de *text-shadow* que dará como resultado un efecto distinto a una simple sombra paralela.

Sabemos que podemos aplicar diferentes sombras a un texto, y que podemos decidir que el difuminado de la sombra sea 0. Partiendo de esta base, podemos crear un efecto de pseudo 3D a cualquier texto.

Primero tenemos que aplicar el efecto sombra a nuestro texto. Creamos un texto blanco y una sombra a una distancia *x* e *y* de un color gris claro y con difuminado 0:

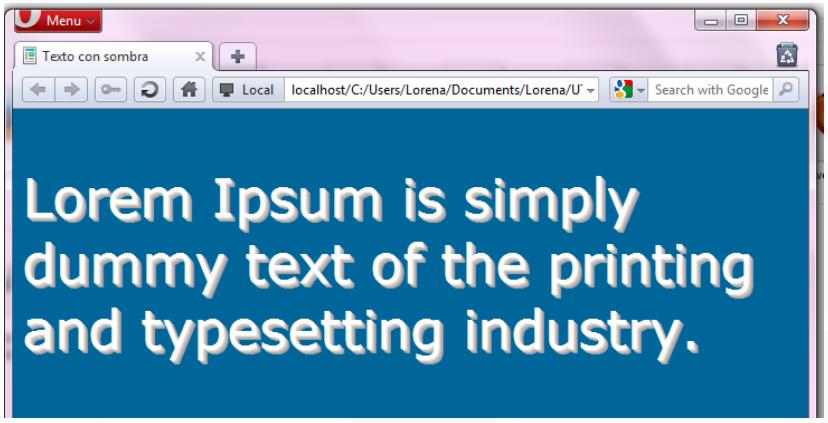
Por ejemplo:

```
text-shadow: 1px 1px 0px rgba(230,230,230,1);
```



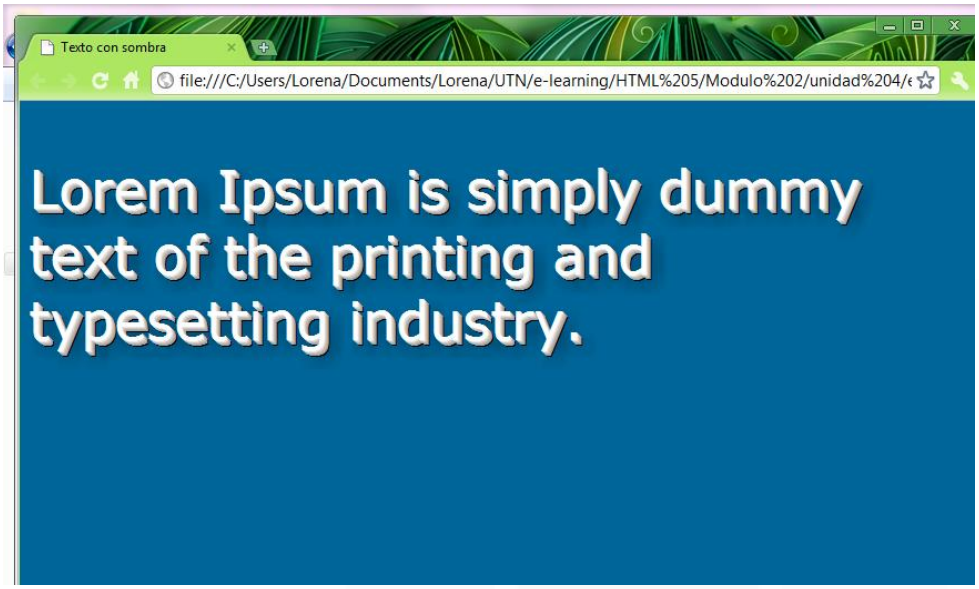
Podemos intensificar el efecto, agregando algunas sombras más:

```
text-shadow:
1px 1px 0px rgba(230,230,230,1),
2px 2px 0px rgba(200,200,200,1),
3px 3px 0px rgba(180,180,180,1),
4px 4px 0px rgba(160,160,160,1);
```



Si queremos hacerlo aún más realista, podemos aplicar una nueva sombra siguiendo el método anterior, con la diferencia que esta vez le daremos color negro. Por último añadimos una sombra mayor que todas las demás, de color gris claro y con difuminado:

```
text-shadow:
1px 1px 0px rgba(230,230,230,1),
2px 2px 0px rgba(200,200,200,1),
3px 3px 0px rgba(180,180,180,1),
4px 4px 0px rgba(160,160,160,1),
/*Añadimos*/
5px 5px 0px rgba(0,0,0,1),
8px 8px 20px rgba(0,0,0,0.5);
```


Texto con sombra
file:///C:/Users/Lorena/Documents/Lorena/UTN/e-learning/HTML%205/Modulo%202/unidad%204/
Lorem Ipsum is simply dummy text of the printing and typesetting industry.

Animaciones CSS

Una de las características más impresionantes de CSS 3 es la posibilidad de crear animaciones de los elementos de la página sin utilizar programas o plugins.

Todo ello nos abre infinitas posibilidades que antes solo estaban disponibles para los programadores Javascript o diseñadores con Flash.

Como todos sabemos, hasta el momento, las animaciones en las páginas web se realizaban utilizando diversas tecnologías accesorias, más allá del simple HTML o CSS.

El primer sistema que alcanzó gran popularidad para realizar una animación de elementos bastante fluida fue la tecnología Flash y luego lo acompañaron algunos otros sistemas como Silverlight, de características similares. Sin embargo, todo esto son tecnologías propietarias, que requieren la instalación de un plugin para funcionar en el navegador, lo que impide que sean universales, por mucha aceptación que hayan llegado a tener.

Paralelamente existen varios otros soportes para animación que sí forman parte de las tecnologías de creación de páginas web universales, pero que no llegan ni de lejos a las posibilidades de animación que podríamos desear. Nos referimos a los GIF animados, que tanto se utilizaron en los comienzos del desarrollo web, así como a Javascript que también permite hacer animaciones a base de cambiar atributos CSS de manera progresiva a lo largo de un tiempo.

Con CSS 3 viene una nueva forma de realizar animaciones totalmente novedosas y que resultará mucho más sencilla que el uso que podemos conocer con Javascript. Pero lo que es más importante, que soporta muchos más tipos de animación que hasta ahora estaban reservados a tecnologías como Flash, como pueden ser rotaciones, ampliaciones y reducciones del tamaño vectoriales, etc.

Esto no se queda ahí, ya que además se han implementado una ciertas interacciones con el usuario y que se consiguen únicamente con CSS 3. Además, todo ello sin tener que utilizar ningún lenguaje de programación, lo que puede resultar mucho más agradable y al alcance de los desarrolladores menos técnicos.

Ventajas de las animaciones CSS 3

Las animaciones CSS permiten hacer muchas de las cosas que antes teníamos reservadas sólo al uso de tecnologías supletorias, que no hacían más que incrementar la dificultad del desarrollo, limitar su compatibilidad entre distintos tipos de usuarios y plataformas, así como los requisitos de conocimientos del desarrollador para poder incorporarlas.

Por tanto, una de las ventajas es que nos podemos olvidar de Flash si queremos hacer elementos con dinamismo en nuestra web. Dejar a Flash de lado además implica que no tenemos que preocuparnos por el posicionamiento de la página que tantos dolores de cabeza provoca cuando nuestra web esta creada enteramente en Flash.

Sin embargo, las ventajas más importantes serían la compatibilidad y la facilidad de implementación, al usar un lenguaje que ya resulta familiar para el desarrollador. La compatibilidad viene dada por el uso de un sistema abierto y regulado por el W3C, al que todos los navegadores tarde o temprano se adaptarán. Y la facilidad de desarrollo porque sólo trabajaremos en nuestros sitios con el lenguaje CSS y no existirá la necesidad de dominar otros lenguajes de programación como ocurría con Flash.

Inconvenientes de las animaciones CSS

También existen algunas desventajas al trabajar con animaciones en CSS. Lo cierto es que la mayor que se podría destacar es sólo circunstancial, debido al poco soporte que existe actualmente a esta utilidad. Tenemos dos principales inconvenientes.

Las animaciones CSS no son admitidas por los muchos navegadores (Ninguna utilidad para animación con CSS 3 se puede utilizar en Internet Explorer y en Firefox algunas cosas ya podemos ver que funcionan a medias, pero aún le queda largo camino por recorrer).
Consume bastantes recursos de máquina para producir las animaciones.

También podremos encontrar que existe alguna dificultad a la hora de la programación, pero no más de la que encontraríamos si tuviésemos que utilizar otros lenguajes o tecnologías distintos de CSS.

Finalmente, volvemos a remarcar que, debido a la imposibilidad de ver los resultados en todos los clientes web, al menos por el momento, deberemos utilizar navegadores basados en Webkit, como son Safari o Google Chrome (siempre en su versión más actualizada).

Algunos conceptos necesarios para comprender las animaciones en CSS.

Fotograma clave

Los ***fotogramas claves*** son valores iniciales y finales que debe tener la animación CSS. Estas localizaciones, en teoría, las sabemos a ciencia cierta, es decir, siempre conocemos en qué punto vamos a empezar y en cual vamos a terminar la animación, así como su duración. Pero podemos crear otros fotogramas clave, no solamente los de inicio y fin, que correspondan con puntos intermedios del movimiento. Las reglas que determinan estos valores es lo que llamamos fotogramas clave dentro de CSS.

Su sintaxis sería algo así:

```
@keyframes 'nombre_fotograma_clave' {

 puntodelKeyframe {
    atributos iniciales;
  }

  puntodelKeyframe {
    nuevos atributos;
  }

  puntodelKeyframe {
    últimos atributos;
  }

}
```

En el código real se traduciría así:

```
@keyframes 'animacion' {

  0% {
    left: 100px;
  }
  40% {
    left: 150px;
  }
  60% {
    left: 75px;
  }
  100% {
    left: 100px;
  }
}
```

Esta animación estaría compuesta de 4 fotogramas clave, el porcentaje es en el momento de la animación en el que va a producirse ese fotograma y los px son la longitud y la alineación donde se colocaría el fotograma dentro del DIV en que se encaje.

El código de dicho DIV sería el siguiente:

```
div {
  animation-name: 'nombre-fotograma-clave';
  animation-duration: 45s;
  animation-iteration-count: 10;
}
```

Los atributos de estilo para esta capa que se ven en el código anterior son los siguientes:

animation-name: el nombre del fotograma clave.

animation-duration: la duración de la animación.

animation-iteration-count: la veces que se repite.

Propiedades sobre la animación aplicables en el DIV

Además de las propiedades que vimos en el párrafo anterior, tenemos otra serie de atributos que se pueden aplicar a la animación y que se colocan en el DIV.

Esta sería una lista de las propiedades adicionales, aplicables para definir las animaciones que especificamos en el DIV:

animation-timing-function: se aplica entre los fotogramas clave, no sobre toda la animación y describe como progresa la animación a lo largo de un ciclo.

animation-direction: esta propiedad define el sentido de la animación. Si especificamos “alternate” y los ciclos de interacción son impares, la animación irá en la dirección normal, si no, se realizará en la dirección inversa

animation-delay: propiedad que nos indica el momento en el que comenzará la animación. Si el valor es 0 se ejecuta en cuanta se carga la página.

animation: esta propiedad combina las anteriores de una forma resumida.

Código completo para una animación CSS

A continuación veremos un código CSS donde estamos definiendo una animación:

```
div {
  animation-name: 'movimiento-diagonal';
  animation-duration: 5s;
  animation-iteration-count: 10;
}

@keyframes 'movimiento-diagonal' {
  from {
    left: 0;
    top: 0;
  }
```

```
  to {
    left: 100px;
    top: 100px;
  }
}
```

Este ejemplo lo que nos mostraría sería una animación en la que se mueve un elemento de la esquina inferior izquierda a la esquina superior derecha, ese movimiento va a tardar 5 segundos y se va a repetir 10 veces.

> *(NOTA: **por el momento este atributo solo funciona en Chrome y Safari, por lo tanto a las propiedades debemos agregarle -webkit-)***

Animación de texto

Lo primero que tenemos que hacer es crear nuestros fotogramas clave, para ello utilizamos el siguiente código en nuestra hoja de estilos:

```
@-webkit-keyframes movimiento-diagonal {
  from {
    left: 0px;
  }

  to {
    left: 100px;
  }
}
```

A través de este código le indicamos el inicio y el fin de nuestra animación.

Ahora crearemos un estilo con selector de id, para darle forma a nuetro div:

```
#anim {
  -webkit-animation-name: movimiento-diagonal;
  -webkit-animation-duration: 3s;
```

```
    -webkit-animation-iteration-count: infinite;
    -webkit-animation-direction: alternate;/*para que vuelva a su posicion inicial */
    width: 100px;
    background-color: Teal;
    color: #fff;
    position: relative;
    padding: 2px;
}
```

En la primera línea le damos el ***nombre a la animación***, que tiene que ser el mismo que el del fotograma clave.

En la segunda le damos una ***duración*** de 3 segundos, es decir, la animación tardará en hacer el recorrido sólo 3 segundos.

En la tercera le decimos que lo ***repita*** infinitas veces.

La propiedad -***webkit-animation-direction: alternate*** hace que el texto, una vez que haga el recorrido, vuelva a su posición inicial realizando el camino inverso.

Y por último le damos un ***ancho***, y ***color de fondo*** y de ***texto***, así como una ***posición relativa***, ya que de lo contrario no funcionaría nuestra animación.

(Recuerden que por el momento estos atributos son válidos solamente en Safari y Chrome)

Atributo Transform

Un atributo que podemos combinar con las animaciones en CSS, es el atributo ***transform***.

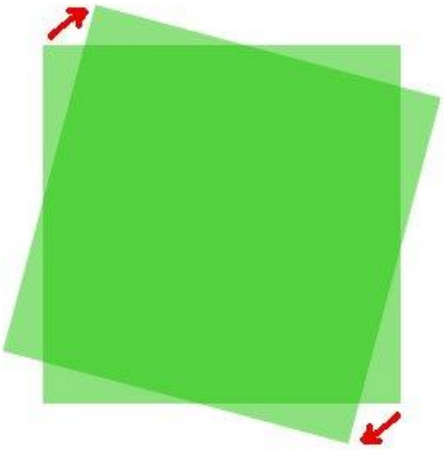
El atributo ***transform*** nos permite, como su propio nombre indica, transformar un elemento. Su sintaxis es la siguiente:

```
transform: tipo(cantidad);
```

El valor tipo puede tomar cuatro valores, y cada uno de ellos realiza una función diferente:

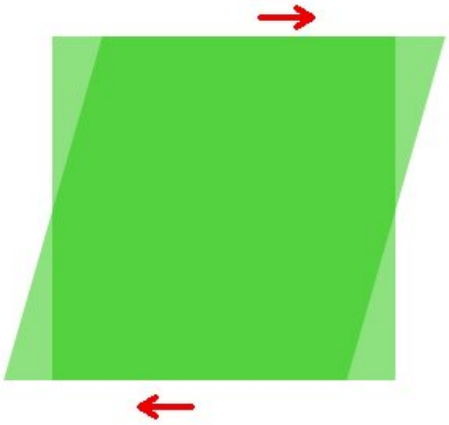
Rotate. Nos permite girar los elementos un número de grados. La sintaxis es:

```
transform: rotate(25deg);
```



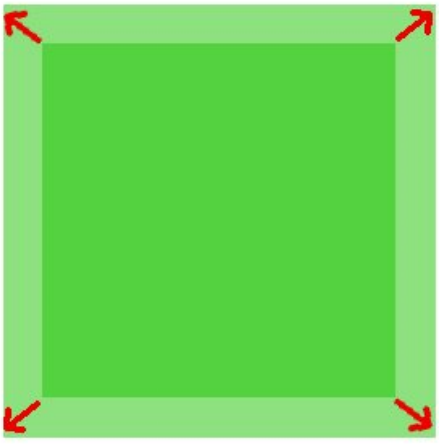
Skew. Podemos inclinar un elemento tanto en coordenadas X como Y. El valor se expresa en grados y la sintaxis es la siguiente:

```
/*transform: skew(gradosX, gradosY);*/
transform: skew(15deg, 3deg);
```



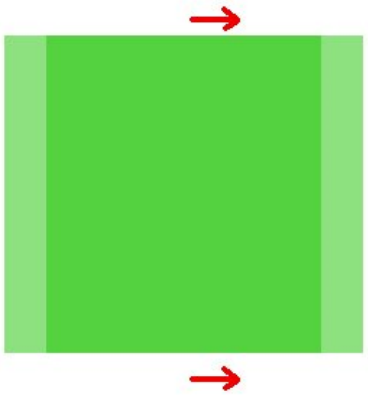
Scale. Con este tipo podremos escalar nuestro elemento tanto en X como en Y en una cantidad expresada en tantos por uno:

```
/*transform: scale(escalaX,escalaY);*/
transform: scale(1.5,0.6);*/
```



Translate. Podemos desplazar el elemento tanto en X como en Y.

```
/*transform: translate(desplazamientoX, desplazamientoY);*/
transform:translate(12px, 19px);
```



Se pueden aplicar diferentes transformaciones a un mismo elemento simplemente escribiéndolas de manera consecutiva:

```
transform: scale(1.6) skew(10deg) translate(5px) rotate(12deg);
```

Responsive Design

Hoy en día ha aumentado de forma exponencial el uso dispositivos móviles, además de la demanda de contenido para estos, por lo que cada vez es más necesario una web adaptable a los distintos dispositivos.

¿Qué es el Responsive Design?

Esta técnica de diseño web consiste en crear una estructura de una página web que según el tamaño de la pantalla (o ventana) en la que se visualice variará su contenido para que siempre sea visible y cómodo de usar desde computadoras de escritorio, tablets y smartphones, y se puede poner en práctica esta forma de adaptar el contenido a todo tipo de resoluciones con hojas de estilo CSS o con JavaScript (tenemos que tener en cuenta también en que puede haber personas o bots que tengan desactivado JavaScript).

Beneficios

La web se visualizará en todos los dispositivos que usemos correctamente, sin necesidad de hacer zoom y se adaptará a los giros en dispositivos móviles.

Google tiene en cuenta que páginas usan diseños que se adaptan a dispositivos móviles.

Ayudamos a las personas que tienen discapacidades visuales a que puedan usar más fácilmente la web.

¿Cómo implementarlo?

Podemos ver dos opciones, tener una web existente o crearla de cero, si existe una web, dependiendo de su complejidad puede ser relativamente sencillo adaptarla o puede ser una tarea tediosa (sobre todo si es un CMS con una plantilla desastrosa, a nivel de estructura).

Tipos de Responsive Designs

Podemos determinar los siguientes tipos de responsive design:

Adaptándose al ancho

Este tipo de diseño es uno de los más comunes, que se distingue por no mover en exceso los elementos de la web e intentar que se adapten al ancho de pantalla reduciendo el menú (a veces cambiando la disposición), redimensionado las imágenes y poco más, como ejemplos pongo los siguientes:

- **Mejorando.la** (**http://mejorando.la/**)

Cascada de columnas

En este caso tenemos una web con varias columnas que al visualizarse en pantallas estrechas se pondría una debajo de la otra seguidamente, por ejemplo:

- **Modernizr** (**http://modernizr.com/**)
- **Wee Nudge** (**http://weenudge.com/**).

Reestructuración

Como el propio título indica en esta variante cambiamos la estructura de los elementos disponiéndolos de una forma distinta según el tamaño de ventana y se puede hacer de muchas formas (hay solo depende del diseño que queramos) por ejemplo:

- **CSS-Tricks** (**http://css-tricks.com/**)
- **Food Sense** (**http://foodsense.is/**)

Cambio del diseño visual

Esta se podría decir que no es una forma de estructurar el contenido para hacerlo más fácil de manejar fuera del navegador de escritorio, sino que según el tamaño de la ventana adapta el diseño de una forma u otra por motivos estéticos, y suele usarse en las típicas web que dan la información sobre algo de forma rápida y concisa, como en las siguientes:

- Neovada (http://www.neovada.com/)
- Design made in Germany (http://www.designmadeingermany.de/magazin/5/)

@media de CSS3

La regla @media nos permite especificar que cierto conjunto de reglas CSS solo deben aplicarse a cierto tipo de dispositivo.

Así las definiciones dentro del bloque de la regla @media screen { ... } solo serían interpretadas por dispositivos conectados a monitores de computadoras y los de la regla@media projection { ... } solo se aplicaría a proyectores.

CSS3 añade importantes y nuevas capacidades que nos permiten definir conjuntos de estilos dependiendo de propiedades comunes de los dispositivos que acceden a nuestros sitios. Propiedades como el alto y el ancho o la relación de aspecto o el número de colores disponible. Las reglas @media pueden ser utilizadas para adaptar nuestras páginas, no solo para dispositivos comunes, sino para todo tipo de dispositivos que nuestros usuarios utilicen para visitar nuestros sitios.

MEDIA QUERIES

Hasta ahora, si necesitábamos conocer el tamaño actual de la ventana del navegador, debíamos usar JavaScript para recolectar datos de ese tipo desde el navegador y después darle un uso a esos datos a través de la modificación del DOM a través de métodos programados en JavaScript. Aunque dicho método es válido, no es realmente óptimo ni intuitivo.

CSS3 nos aporta las *media queries* que nos proveen de una forma de conocer algunas de las propiedades comunes de los dispositivos que nos visitan que podemos utilizar en nuestros archivos de estilo para construir entornos dependiendo de los mismos **sin ayuda** de JavaScript.

Aunque *media queries* dispone de muchas propiedades diferentes, podemos identificar 5 principales:

- *Aspect-ratio:* Mira las dimensiones relativas del dispositivo expresadas como una relación de aspecto: 16:9 por ejemplo.
- *Width y height:* Mira las dimensiones del área de visualización. Además pueden ser expresadas en valores mínimos y máximos.
- *Orientation:* Mira si el layout es panorámico (el ancho es mayor que el alto) o vertical (el alto es mayor que el ancho). Esto nos permite ajustar los diseños para dispositivos con propiedades de giro de la pantalla como el iPhone, y otros smartphones y los tablets.
- *Resolution:* Mira la densidad de los pixeles en el dispositivo de salida. Esto es especialmente útil cuando queremos aprovecharnos de las ventajas de los dispositivos que tiene una resolución mayor a 72 dpi.
- *Color, Color-index y monochrome*: Encuentran el número de color o bits por color. Esto nos permite crear diseños específicos para dispositivos monocromáticos.

VIEWPORT

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

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

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

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

- *Width:* ancho de la página (se puede establecer en píxeles o como device-width y usará el ancho del que dispone).
- *Height:* alto de la página, actúa igual que el width.

- *Initial-scale:* escala o zoom inicial de la página (este y los demás tipos de escala se establecen con valores como 1.0 para no tener zoom o 2.5 para tener un zoom del 2,5 de aumento, por ejemplo).
- *Minimum-scale:* zoom mínimo que podemos hacer en la página.
- *Maximum-scale:* zoom máximo que podemos hacer en la página.
- *User-scalable:* establece si está permitido o no hacer zoom (yes/no).

USANDO MEDIA QUERIES PARA ESPECIFICAR ESTILOS

Antes de nada debemos crear nuestra hoja de estilos. Creamos un estilo por defecto que incorporará todas las necesidades **universales** para nuestro diseño y lo guardamos. Si nos sentimos especialmente originales, podemos llamar al archivo **default.css**:

```
/**
 * Estilo por defecto
 */

body {
   background: white url(../media/fondo.jpg) no-repeat 0 0;
   margin: 0;
   padding: 100px 0 0 75px;
}

h1 {
   color: black;
   font-style: italic;
}

h2 {
   color: #717171;
}

p {
   font: normal 100%/1.5 Helvetica, Arial, Sans-serif;
   color: #313131;
```

```
}
```

Creamos archivos de estilo para cada tipo de dispositivo o media para el que queramos diseñar un estilo específico. Por ejemplo, podemos incluir un estilo específico para impresoras, si nos seguimos sintiendo igual de originales a este podemos llamarle, no se, ***print.css*** por ejemplo:

```
/**
 * Estilo para impresoras
 */
body {
   background: white url(../media/fondo_printer.jpg) no-repeat 0 0;
   margin: 0;
   padding: 100px 0 0 75px;
}

h1 {
   color: #818181;
}

p {
   font: normal 12pt/2 Constantia, palatino times, "times new roman", serif;
   color: #000000;
}
```

Ya tenemos un estilo por defecto y otro para cuando la página va a ser impresa. Vamos a añadir otro para el iPhone, y para no perder la originalidad que nos caracteriza, vamos a llamarle ***iphone.css***:

```
/**
 * Estilo para dispositivos iPhone
 */
body {
   -webkit-text-size-adjust: none;
   background: rgb(102, 102, 102) url(../media/fondo_iphone.jpg) no-repeat center 0;
   padding: 20px 5px 5px 5px;
}

h1 {
```

```
    color: rgb(140, 120, 120);
    text-shadow: 0 0 5px rgb(0, 0, 0);
}

p {
    font: normal 1em/1.25em "helvetia neue", Helvetia, Arial, Sans-serif;
    color: rgb(255, 255, 255);
}
```

Ahora vamos a añadir la etiqueta meta de la vista en el head del documento HTML:

Eso **previene** de que dispositivos con una pantalla más pequeña —como el iPhone— redimensionen la página impidiendo que los estilos se interpreten. Enlazamos nuestro estilo por defecto en el documento HTML a través de la etiqueta **link** y utilizando como regla **media all**.

Hacemos lo propio con la hoja de estilo **print.css** pero en esta ocasión utilizamos la regla **print**.

Lo mismo con la hoja de estilo para el iPhone, pero esta vez utilizaremos la regla **screen** y añadimos nuestros primer *media queries* en paréntesis conectando múltiples consultas con la palabra **and**.

El código completo del documento HTML sería como el que sigue:

```
<!DOCTYPE html>
<html lang="es">
    <head>
        <meta http-equiv="Content-Type" content="text/html" charset="UTF-8">
        <meta name="viewport" content="width=device-width; initial-scale=1.0; maximum-scale=1.0; user-scalable=0;">
        <title>Ejemplo de Media Queries</title>
        <link rel="stylesheet" media="all" href="/css/default.css">
        <link rel="stylesheet" media="print" href="/css/print.css">
        <link rel="stylesheet" media="screen and (max-device-width: 480px) and (min-device-width: 320px;)" href="iphone.css">
    </head>
    <body>
        <h1>La Comunidad del Anillo</h1>
        <article>
```

```
        <header>
          <h1><strong>Capítulo I.</strong> Gandalf el Mago</h1>
        </header>
        <p>Cuando Frodo Bolson de Pipas vio acercarse por el camino el carruaje de
        Gandalf el Mago no pudo evitar soltar un gritito de alegría como
        <em>¡Vaaamono preeeehmoh!</em></p>
      </article>
      <footer>
        <p class="author">by J.R.R. Tolkien </p>
      </footer>
    </body>
  </html>
```

La propiedad @media

La propiedad **`@media`** permite que incluyamos *media queries* directamente en nuestras hojas de estilo. De esta manera **mejoramos el rendimiento** de la página al no tener que cargar archivos CSS adicionales para los diferentes dispositivos.

Para utilizar las reglas `@media` lo único que tenemos que hacer es crear nuestra hoja de estilos y cuando queramos definir un conjunto de reglas de estilo específicas para un dispositivo determinado utilizar la sintaxis:

```
@media screen and (parámetros) and (parámetros) { ... }
```

El efecto es el mismo pero minimizamos la carga de archivos a un único archivo que incluya todas nuestras reglas CSS.

Ejemplo:

Vamos a poner un ejemplo que podría ser funcional, que se basa en un menú que hay en una cabecera y consta de cinco links que mandan a las distintas partes de la web y actuará de la siguiente forma los elementos `<a>`:

- En principio tendrá el siguiente código, que mantendrá a los cinco elementos en línea.

```
a.menu {
        display:inline-block;
        padding:0px 12px;
        margin:0px 8px;
}
```

- Cuando el ancho de pantalla sea inferior a 1200 píxeles (lo que podría ser el ancho máximo de la página) los elementos del título se juntan para que no sobresalgan de la cabecera.

```
@media all and (max-width: 1200px) {
        a.menu {
                display: inline-block;
                padding:0px 6px;
                margin:0px 4px;
        }
}
```

- Cuando el ancho de ventana sea inferior a 840 píxeles se reestructurarán los elementos y pasaran de estar en línea a estar en cascada uno debajo del otro y tomarán un pequeño margen a su izquierda.

```
@media all and (max-width: 840px) {
        a.menu{
                margin:0px;
                padding:0px;
                padding-left:5%;
                display:block;
                float:none;
                text-align:left;
        }
}
```

- Y por último a los 520 píxeles imaginamos que para que se adapte a anchos estrechos el resto de la página ha cambiado de estructura, por lo que ahora dispone el menú de todo el ancho y lo añadiremos un margen izquierdo superior.

```
@media all and (max-width: 520px) {
```

```
        a.menu{
                padding-left:20%;
        }
}
```

Resumen

En esta Unidad…

En la presente unidad desarrollamos los conceptos necesarios para incorporar los atributos gráficos a nuestras estructuras de HTML utilizando el lenguaje CSS3

Con las propiedades propuestas podemos utilizar cualquier familia tipográfica para nuestra web. También trabajamos con la posibilidad de desarrollar sitios adaptables a los diferentes dispositivos disponibles hoy en el mercado.

En la próxima Unidad…

En la próxima unidad vamos comenzar a trabajar con los conceptos de programación, para prepararnos para incorporar contenido dinámico a nuestros sitios.