4. ¿Que son medias queries y cómo implementarlas?

Los Media Queries, son una excelente forma de entregar diferentes estilos para diferentes dispositivos, y proveer la mejor experiencia para cada tipo de usuario. Como parte de la especificación CSS3, los Media Queries expanden el rol del atributo media, que controla como se aplican los estilos. Por ejemplo: se ha vuelto una práctica común por años el uso de una hoja de estilos por separado para imprimir sitios web al especificar media = “print”. Los Media Queries, llevan esta idea al siguiente nivel, al permitir a los diseñadores asignar estilos basados en las propiedades de un dispositivo, como el ancho de pantalla, orientación y así sucesivamente.

Las Figuras 1- 3, muestran los Media Queries en acción. De hecho muestran el mismo sitio web, visto en una computadora de escritorio, en una tableta y en un iPod Touch.

Un ejemplo simple de un Medya Query puede verse así:



|  |  |
| --- | --- |
| 1  2 | &lt;link href="css/phone.css" rel="stylesheet" type="text/css"  media="only screen and (max-width: 400px)" &gt; |

En este ejemplo, el Media Query, ha sido agregado a la etiqueta. Como podrás ver más adelante, puedes usar Media Queries con hojas de estilo. El atributo media es dónde reside el Query. Este otro dice:



|  |  |
| --- | --- |
| 1 | only screen and (max-width: 400px) |

El significado puede ser obvio: aplica esta hoja de estilo sólo a un dispositivo con una ventana de navegador no mayor a 400 pixeles. Puedes observar por el nombre del archivo phone.css que este Query en particular, esta hecho para estilos de un dispositivo móvil. Ahora que has visto un ejemplo, vamos a profundizar en los Media Queries y las características de un dispositivo que se pueden utilizar para controlar como se aplican los estilos.

5. ¿Qué son flexbox en CSS, para que sirve y como se implementan?

La propiedad **Flexible Box**, o **flexbox**, de CSS3 es un modo de diseño que permite colocar los elementos de una página para que se comporten de forma predecible cuando el diseño de la página debe acomodarse a diferentes tamaños de pantalla y diferentes dispositivos. Para muchas aplicaciones, el modelo "caja flexible" produce una mejora sobre el modelo "bloque" porque no utiliza la propiedad float, ni hace que los márgenes del contenedor flexible interfieran con los márgenes de sus contenidos.

Muchos diseñadores verán que el modelo "caja flexible" es más sencillo de utilizar. Los elementos "hijos" de una "caja flexible" pueden colocarse en cualquier dirección y pueden tener dimensiones flexibles para adapterse al espacio visible. Posicionar los elementos "hijos" es por tanto mucho más sencillo, y los diseños complejos pueden hacerse más fácilmente y con código más limpio, ya que el orden de visualización de los elementos es independiente del orden que estos tengan en el código fuente. Esta independencia afecta intencionadamente únicamente a la representación visual, dejando el orden de locución y navegación a lo que diga el código fuente.

existen 6 diferentes modos de maquetación en CSS. Vamos a dar un repaso a todos ellos:

* **Maquetación de bloques**. Es la que todos conocemos; cada elemento es un bloque que, de manera predeterminada, ocupa todo el ancho disponible. Además, los bloques tienen la habilidad de poder flotar. Igualmente podemos organizar el contenido por columnas. Propiedad **display: block**.
* **Maquetación en línea**. Corresponde al modo en que se organizan y comportan los elementos de tipo texto. Propiedad **display: inline**.
* **Maquetación de tablas.** Para el diseño de tablas. Antiguamente se maquetaban páginas web utilizando este modo. Contiene un gran número de modos de display: table, table-cell, table-column, table-row, inline-table, entre otros.
* **Maquetación posicionada**. Para cuando la posición de un elemento no guarda (mucha) relación con los demás. Funciona cuando utilizamos la [propiedad CSS *position*](http://dev.w3.org/csswg/css3-positioning/#position-property) con algún valor de estos: fixed, absolute, center o page.
* **Maquetación flexbox**. Nos permite diseñar páginas complejas que se redimensionan *con elegancia*.

**Maquetación en rejilla**, para colocar los elementos relativos a una rejilla.

Echa un vistazo a su código HTML:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | <!DOCTYPE html>  <html lang="en">    <head></head>    <body>      <header>header</header>      <div id='main'>        <article>article</article>        <nav>nav</nav>        <aside>aside</aside>      </div>      <footer>footer</footer>    </body>  </html> |

6. ¿que son fuentes iconográficas, cuales existen y cómo implementarlas?

Fuentes ICONOGRÁFICAS: Son fuentes primarias e incluyen cualquier tipo de imágenes: la pintura, las fotografías, los grabados, las ilustraciones y los carteles, las caricaturas, los cómics, etc.

Se implementan asi: <img src=”yo.jpg"

**7. ¿Cómo agregar tipografías nuevas a mi proyecto web, y cuales alternativas?**

Históricamente los diseñadores web nos hemos encontrado con el problema de poder incluir únicamente un puñado de aburridas fuentes “seguras” en nuestros diseños web. Afortunadamente desde hace un tiempo este problema fue resuelto mediante nuevas propiedades de CSS (como la @font-face) permitiéndonos embeber las fuentes en nuestro site y que se visualizaran correctamente, con indiferencia de si el usuario final tiene instalada o no la fuente en concreto en su ordenador.

Pero esta propiedad mágica (@font-face) de CSS seguro que os ha dado más de un dolor de cabeza al comprobar que no es tan “mágica” después de todo. Lo mas seguro es que vuestra Web se vea completamente diferente en cada nuevo navegador con la que la probáis…verdad? Y ya no hablemos de versiones del mismo navegador…todo un mundo!

**a)   Para formatos TTF**

@font-face {

font-family: **nombre-De-Mi-Fuente**;

src: url(url/de/mi/fuente/**fuente.ttf**) format(‘truetype’);

}

**b)  Para formato OTF**

@font-face {

font-family: **nombre-De-Mi-Fuente**;

src: url url/de/mi/fuente/**fuente.otf**) format(‘opentype’);

}

**c)  Para formato SVG**

font-face {

font-family: **nombre-De-Mi-Fuente**;

src: url (‘url/**fuente.svg#Identificador-de-fuente’**) format(‘svg’);

}

**d)     Para formatos EOT:**

@font-face {

font-family: **nombre-De-Mi-Fuente**;

src: url url/de/mi/fuente/**fuente.eot**);

}

**e)     Para varios formatos a la vez, por ejemplo OTF + SVG:**

@font-face {

font-family: ‘nombre-de-mi-fuente’;

url(‘fuente.otf’) format(‘opentype’),

url(‘fuente.svg#Fuente-identificador’) format(’svg’);

}

Y una vez incrustada la fuente, ya podemos usarla normalmente como una fuente de las denominadas “seguras”:

.headder\_text{

font-size:19px;

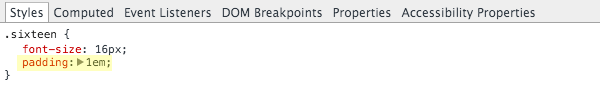
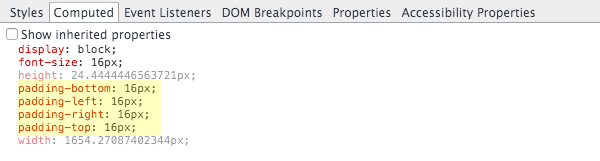
font-family: nombre-de-mi-fuente, Tahoma;

}

**8. ¿qué es em y rem en css y para que se usa?**

Puedes haber entendido el uso de unidades flexibles de medida, pero podrías no haber entendido completamente cuando usar rem y cuando usar em. ¡Éste tutorial te ayudará a descubrirlo!

Tanto em como rem son flexibles, unidades escalables que son traducidas por el navegador en valores de pixeles, dependiendo de los ajustes del tamaño de la fuente en tu diseño. Si usas un valor de 1em o 1rem, pudiera traducirse en el navegador como cualquier cosa desde 16px hasta 160px   o cualquier otro valor.

CSS padding establecido en 1 emCalcula a 16px

Por otro lado valores en px son usados por el navegador como son, así que1px siempre se mostrará exactamente 1px.

Manipula el regulador deslizante en éste ejemplo en CodePen para ver como el valor de las unidades rem y em pueden traducirse en diferentes valores de pixeles, mientras explícitamente estableces las unidades en px fijas en tamaño:

Cuando se usan uniddes rem, el tamaño del pixel al que traducen depende del tamaño de la fuente del elemento raíz de la página, o sea el elemento html. Ese tamaño de fuente raíz es multiplicado por cualquier numero que está usando con tu unidad rem.

Por ejemplo, con un tamaño fuente de elemento raíz de 16px, 10rem debería equivaler a 160px, o sea 10 X 16 = 160.

Cuando se usan unidades em, el valor del pixel con el que terminas es una multiplicación del tamaño de fuente en el elemento al que se le está aplicando estilo.

Por ejemplo, si un div tiene un tamaño de fuente de 18px, 10em equivaldría a180px, o sea 10 X 18 = 180.