



Tabla de contenido

CSS3 Avanzado II	2
Medias Queries (@media)	2
Syntaxis	2
Medios	3
Tamaños de pantalla	4
Orientación	4
Combinar condiciones	4
Características de Medios (media feature)	6
At-rules	10
@import	10
@namespace	10
@supports	11
La palabra clave not	11
La palabra clave or	12
La palabra clave and	12
@keyframes	12
Listas válidas de keyframe	13
Usando validación de formularios	15
Otras restricciones de validación	17
Mensajes de error personalizados	19
Enlaces con las fuentes de la información	20

Tabla de Ejercicios

EJERCICIO 1: @MEDIA	9
EJERCICIO 2: @SUPPORTS	12



CSS3 Avanzado II

Medias Queries (@media)

Las Media Queries son una de las grandes ventajas de CSS3, ya que permiten saber qué sistema se está visualizando una página web y, en función de ello, aplicar unas reglas de estilo u otras. Así, podemos servir un CSS personalizado, acorde las condiciones del navegador o dispositivo que nos visita. Como veremos a continuación, se han convertido en **uno de los mejores recursos con los que cuentan los diseñadores para hacer sitios responsive**.

Utilizar las Media Queries de CSS3 es muy sencillo, como todo el lenguaje CSS en general. De una manera descriptiva podemos indicar una condición y a continuación los estilos que deben aplicarse cuando ésta se cumpla. Para aclarar su sintaxis veremos una serie de ejemplos útiles e ilustrativos.

Podemos usarlas con tranquilidad, puesto que todos las soportan. Solo versiones muy antiguas de Internet Explorer tendrán problemas con ellas, de IE8 hacia atrás.

La última especificación puedes encontrarla en <https://www.w3.org/TR/mediaqueries-4/>

Sintaxis

Podemos ver dos formas distintas de utilizar media queries, utilizando el estilo o bien enlazando un archivo externo.

```
<!-- CSS media query on a link element -->
<link rel="stylesheet" media="(max-width: 800px)" href="example.css"/>

<!-- CSS media query within a style sheet -->
<style>
@media (max-width: 600px) {
  .facet_sidebar {
    display: none;
  }
}
</style>
```

La sintaxis formal es un poco compleja:

```
@media <media-query-list> {
  <group-rule-body>
}
```

donde

<media-query-list> = <media-query>#

donde

<media-query> = <media-condition> | [not | only]? <media-type> | and
<media-condition-without-or>]?

donde

<media-condition> = <media-not> | <media-and> | <media-or> | <media-in-parens>

<media-type> = <ident>

<media-condition-without-or> = <media-not> | <media-and> | <media-in-parens>

donde

```
<media-not> = not <media-in-parens>
<media-and> = <media-in-parens> [ and <media-in-parens> ]+
<media-or> = <media-in-parens> [ or <media-in-parens> ]+
<media-in-parens> = ( <media-condition> ) | <media-feature> | <general-enclosed>
```

donde

```
<media-feature> = ( [ <mf-plain> | <mf-boolean> | <mf-range> ] )
<general-enclosed> = [ <function-token> <any-value> ] | [ <ident> <any-value> ]
```

donde

```
<mf-plain> = <mf-name> : <mf-value>
<mf-boolean> = <mf-name>
<mf-range> = <mf-name> [ '<' | '>' ]? '='? <mf-value> | <mf-value> [ '<' | '>' ]? '='? <mf-name> | <mf-value> '<' '='? <mf-name> '<' '='? <mf-value> | <mf-value> '>' '='? <mf-name> '>' '='? <mf-value>
```

donde

```
<mf-name> = <ident>
<mf-value> = <number> | <dimension> | <ident> | <ratio>
```

Medios

Con Media Queries podemos detectar el medio donde se está consumiendo un sitio web. Los valores que pueden tomar son:

- **all**: cualquier tipo de dispositivo, está definido por defecto.
- **print**: Para aquellas páginas o documentos paginados que se muestren en vista previa de impresión.
- **screen**: Pantallas a color
- **speech**: para sintetizadores de voz

NOTA: CSS2.1 y Media Queries 3 definió varios tipos de media adicionales (tty, tv, projection, handheld, braille, embossed, aural), pero fueron descontinuados en Media Queries 4 y no deben ser usados.

```
@media screen {
  h1 {
    color: red;
  }
}
@media print {
  h1 {
    color: black;
  }
}
```

En estos ejemplos tenemos media queries sencillas, en las que se distingue el modo en el que se está consumiendo la página, en una pantalla o en la impresora. Otro ejemplo podemos verlo así:

```
1 <link rel="stylesheet" type="text/css" media="screen" href="pantalla.css">
2 <link rel="stylesheet" type="text/css" media="print" href="impresora.css">
```

Aquí se expresa que una hoja de estilo “ejemplo.css” se aplica a los dispositivos de un determinado tipo de medio ‘ screen ‘ (pantalla) con una característica (que debe ser una pantalla de color).

```
<link rel="stylesheet" media="screen and (color)" href="ejemplo.css" />
```

Tamaños de pantalla

Lo más común es que usemos las Media Queries para detectar las dimensiones de la pantalla.

```
@media (min-width: 1000px) {  
  body {  
    align: 20px;  
  }  
  div {  
    text-align: center;  
  }  
}
```

Con este ejemplo, estamos aplicando una serie de estilos, que se tendrán en cuenta solamente cuando la anchura de la pantalla sea de 1000px en adelante.

```
@media (max-width: 320px) {  
  body {  
    font-size: .6em;  
  }  
}
```

En este caso, indicamos que la anchura de la pantalla debe de ser 320 píxeles de máximo para que se apliquen los estilos.

Orientación

También es útil para definir estilos en función de la orientación, lo que resulta especialmente adecuado para el diseño para móviles.

```
@media (orientation:landscape){  
  body{  
    background-image:url(archivo.jpg);  
  }  
}  
@media (orientation:portrait){  
  body{  
    background-image:url(archivo-pantalla-en-vertical.jpg);  
  }  
}
```

Landscape es el valor de la orientación cuando la pantalla está en horizontal y **portrait** para la pantalla en vertical.

Combinar condiciones

Se pueden redactar queries utilizando operadores lógicos, incluyendo `not`, `and`, y `only`. Además se puede combinar múltiples queries en una lista separada por comas múltiples; si cualquiera de las queries en la lista es verdadera, la hoja de estilo asociada es aplicada. Esto es equivalente a una operación lógica "or".



and

Si la pantalla esta en formato horizontal, usted puede utilizar el operador `and` y colocar la siguiente cadena:

```
@media (min-width: 700px) and (orientation: landscape) { ... }
```

Esta regla aplicaría en cualquier medio con dimensiones entre 800 y 1200 píxeles.

```
@media (min-width: 800px) and (max-width: 1200px) {  
    .item {  
        display: none;  
    }  
}
```

Lista separada por comas

Las listas separadas por comas se comportan como el operador `or` cuando es usado en media queries. Cuando utilice una lista separada por comas y alguno de los queries retorna verdadero, el estilo o la hoja de estilos será aplicada. Cada query en una lista separada por comas es tratado como una query individual y cualquier operador aplicado a un medio no afectara a los demás. Esto significa que cada query en una lista separada por comas puede tener como objetivo diferentes medios, tipos y estados.

Para un equipo con un ancho mínimo de 700px o si el dispositivo esta colocado en horizontal, usted puede escribir lo siguiente:

```
@media (min-width: 700px), handheld and (orientation: landscape) { ...  
}
```

Por lo tanto, si esta en una `screen` con una ventana de 800px de ancho, la declaración del medio retornara verdadero debido a la primera parte de la lista, si aplicamos esto a un dispositivo `@media all and (min-width: 700px)` podría retornar verdadero a pesar del hecho de que la pantalla falle la verificación tipo de medio del `handheld` en la segunda query. Por otra parte si estuviese en un `handheld` con un ancho de ventana de 500px, la primera parte de la lista fallaría pero la segunda parte retornara verdadero debido a la declaración de medio.

not

El operador `not` aplica a todo el query y retorna verdadero si es posible y sino retorna falso (como por ejemplo `monochrome` en un monitor a color o una ventana con un ancho mínimo de `min-width: 700px` en un monitor de 600px). Un `not` negara un query, si es posible, pero no a todos los query posibles si están ubicados en una lista separada por comas. El operador `not` **no puede ser usado para negar un query individual**, solo para un query completo. Por ejemplo, el `not` en el siguiente query es evaluado al final:

```
@media not all and (monochrome) { ... }
```

Esto significa que el query es evaluado de la siguiente forma:

```
@media not (all and (monochrome)) { ... }
```

... y no de esta forma:

```
@media (not all) and (monochrome) { ... }
```



Otro Ejemplo:

```
@media not screen and (color), print and (color)
```

Es evaluado de la siguiente forma:

```
@media (not (screen and (color))), print and (color)
```

only

El operador `only` previene que navegadores antiguos que no soportan queries apliquen los estilos asignados:

```
<link rel="stylesheet" media="only screen and (color)"  
href="Ejemplo.css" />
```

Si diferentes Media Queries son válidas para un entorno y surgen conflictos, éstos se resolverían por la norma de la cascada, de modo que las últimas reglas de estilo definidas serían las que se tendrían en cuenta.

Características de Medios (media feature)

Cada *característica de medios* verifica una característica específica del navegador o dispositivo.

Nombre	Resumen	Notas
<u>width</u>	Anchura del viewport	
<u>height</u>	Altura del viewport	
<u>aspect-ratio</u>	Relación de aspecto anchura-altura del viewport	
<u>orientation</u>	Orientación del viewport	
<u>resolution</u>	Densidad de píxeles del dispositivo	
<u>scan</u>	Proceso de escaneo del dispositivo	
<u>grid</u>	¿El dispositivo es un grid o un mapa de bits?	
<u>update-frequency</u>	Qué tan rápido (si lo hace) puede el dispositivo modificar la apariencia del contenido	Incluido en Media Queries Nivel 4
<u>overflow-block</u>	Cómo maneja el dispositivo el contenido que excede los límites del viewport a lo largo del eje de bloque	Incluido en Media Queries Nivel 4
<u>overflow-inline</u>	¿Puede desplazarse hacia el contenido que excede los límites del viewport?	Incluido en Media Queries Nivel 4
<u>color</u>	Componente de número de bits por color del dispositivo, o cero si el dispositivo no es a color.	
<u>color-index</u>	Número de entradas en la tabla de búsqueda de color del dispositivo, o cero si el dispositivo no usa una tabla.	
<u>display-mode</u>	Modo de visualización de la aplicación, según se especifique en la <u>propiedad display</u> del manifiesto de la aplicación web.	Definido en la <u>especificación del Manifiesto de Aplicación Web</u> .



<u>monochrome</u>	Bits por pixel en el buffer de marco monocromático del dispositivo, o 0 si el dispositivo no es monocromático.	
<u>inverted-colors</u>	¿El agente usuario o el Sistema Operativo está invirtiendo los colores?	Incluido en Media Queries Nivel 4
<u>pointer</u>	¿El mecanismo de entrada principal es un dispositivo apuntador? y de ser así, ¿qué tan preciso es?	Incluido en Media Queries Nivel 4
<u>hover</u>	¿El mecanismo de entrada principal permite que el usuario posicione el puntero sobre los elementos?	Incluido en Media Queries Nivel 4
<u>any-pointer</u>	¿Hay algún mecanismo de entrada que sea dispositivo apuntador? y de ser así, ¿qué tan preciso es éste?	Incluido en Media Queries Nivel 4
<u>any-hover</u>	¿Algún mecanismo de entrada disponible permite que el usuario posicione el puntero sobre los elementos?	Incluido en Media Queries Nivel 4
<u>light-level</u>	Nivel de luz ambiental actual	Incluido en Media Queries Nivel 4
<u>scripting</u>	¿Se soporta lenguaje de script (p.ej. JavaScript)?	Incluido en Media Queries Nivel 4
<u>device-width</u>	Anchura de la superficie de representación del dispositivo	Descontinuado en Media Queries Nivel 4
<u>device-height</u>	Altura de la superficie de representación del dispositivo	Descontinuado en Media Queries Nivel 4
<u>device-aspect-ratio</u>	Relación de aspecto anchura-altura del dispositivo	Descontinuado en Media Queries Nivel 4
<u>-webkit-device-pixel-ratio</u>	Número de píxeles reales del dispositivo por pixel CSS	No estándar; Específica de WebKit/Blink. De ser posible, use la característica <u>resolution</u> en su lugar.
<u>-webkit-transform-3d</u>	¿Se soportan <u>transformaciones</u> 3D?	No estándar; Específica de WebKit/Blink
<u>-webkit-transform-2d</u>	¿Se soportan <u>transformaciones</u> 2D?	No estándar; Específica de WebKit
<u>-webkit-transition</u>	Se soportan <u>transiciones</u> CSS?	No estándar; Específica de WebKit
<u>-webkit-animation</u>	¿Se soportan <u>animaciones</u> CSS?	No estándar; Específica de WebKit

Veamos algunos ejemplos:

Para indicar que una hoja de estilo debe aplicarse a todos los dispositivos utilizando índices de color, usted puede hacer esto:

```
@media all and (color-index) { ... }
```

Para aplicar una hoja de estilo a un dispositivo con un índice de al menos 256 colores:

```
<link rel="stylesheet" media="all and (min-color-index: 256)"  
href="http://foo.bar.com/stylesheet.css" />
```



Este selecciona el estilo cuando la proporción de aspecto sea 1:1 o superior. En otras palabras este estilo solo sera aplicado cuando el área de visualización sea cuadrada u horizontal.

```
@media screen and (min-aspect-ratio: 1/1) { ... }
```

El siguiente código selecciona una hoja de estilo especial para ser aplicada en pantallas panorámicas ("widescreen"). Este selecciona el estilo cuando la proporción de aspecto sea 16:9 o 16:10.

```
@media screen and (device-aspect-ratio: 16/9), screen and (device-aspect-ratio: 16/10) { ... }
```

Para aplicar una hoja de estilo a un documento cuando este sea mostrado en una pantalla de menos de 800px de ancho, usted puede usar esto:

```
<link rel="stylesheet" media="screen and (max-device-width: 799px)" />
```

Para aplicar una hoja de estilo a dispositivos con al menos 300 dpi de resolución:

```
@media print and (min-resolution: 300dpi) { ... }
```

Para reemplazar la vieja sintaxis (min-device-pixel-ratio: 2):

```
@media screen and (min-resolution: 2dppx) { ... }
```

Esta query especifica una hoja de estilo para ser aplicada a un medio impreso con un ancho mayor a 8.5 pulgadas:

```
<link rel="stylesheet" media="print and (min-width: 8.5in)"  
href="http://foo.com/mystyle.css" />
```

Esta query especifica una hoja de estilo para ser utilizada cuando la ventana tiene un ancho entre 500 y 800 pixeles:

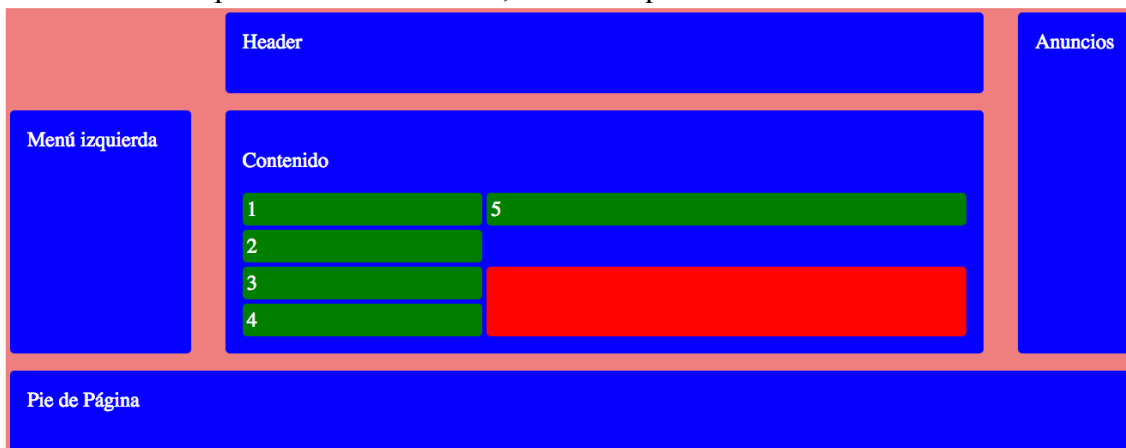
```
@media screen and (min-width: 500px) and (max-width: 800px) { ... }
```

```
1  @media print {  
2    body { font-size: 10pt }  
3  }  
4  @media screen {  
5    body { font-size: 13px }  
6  }  
7  @media screen, print {  
8    body { line-height: 1.2 }  
9  }  
10 @media only screen  
11    and (min-device-width: 320px)  
12    and (max-device-width: 480px)  
13    and (-webkit-min-device-pixel-ratio: 2) {  
14    body { line-height: 1.4 }  
15  }
```

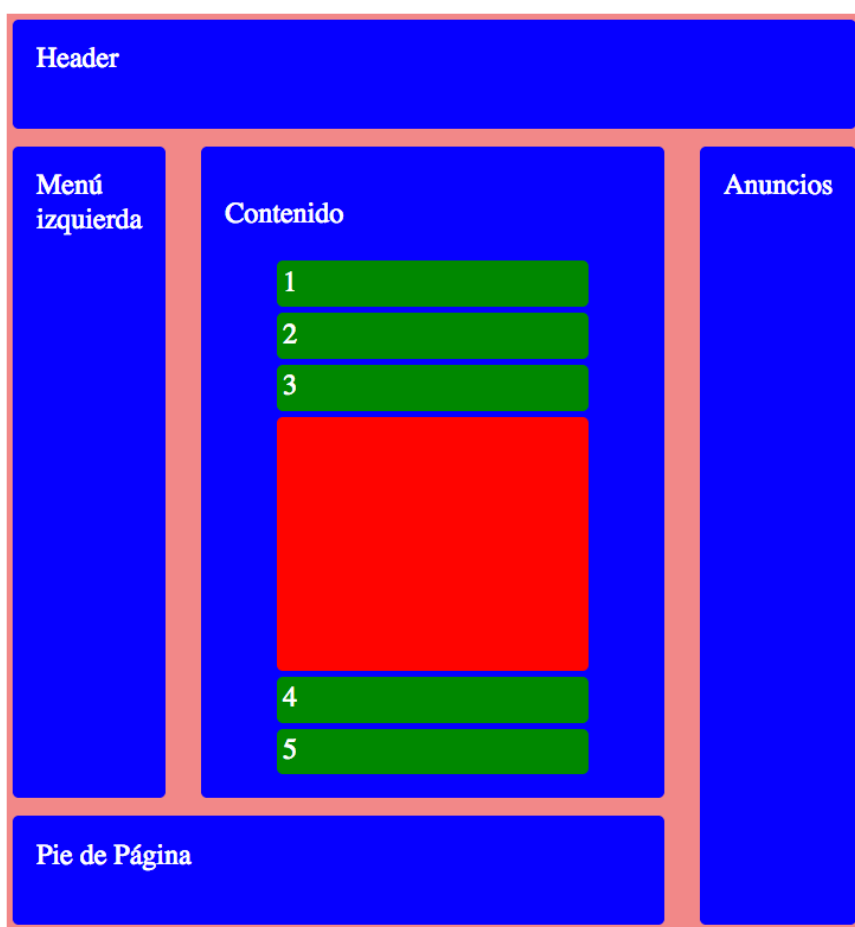



Ejercicio 1: @media

Partiendo del ejercicio realizado en CSS Grid modifica el link del html para que cargue el css cuando la pantalla sea horizontal, recuerda que se mostrará así:



Añade un nuevo enlace donde se cargue otro css (por ejemplo main2.css) si la pantalla es vertical. Utilice CSS Grid para ubicar los elementos de la siguiente manera:



NOTAS:

En wrapper2 se ha aplicado un ancho del 70%, pero no puede utilizar width, tiene que hacerlo con elementos del css grid.

El item6 tiene un tamaño igual que las otras filas, es decir, ocupa lo mismo que la suma de las otras 5 filas

At-rules

@import

@import permite importar reglas desde otras hojas de estilo. Estas reglas deben preceder a todos los otros tipos de reglas, excepto a las reglas @charset; como esto no es una declaración anidada, no puede ser usado dentro de grupos condicionales de reglas-at.

Para que los agentes de usuario puedan evitar recuperar recursos para tipos de medios no soportados, los autores pueden especificar reglas dependientes del tipo de medio @import. Estas condiciones @import se especifican separando por una coma las consultas de medios (media query) después de la url. En la ausencia de cualquier consulta de medios (media query), la importación es incondicional. Especificando all para el medio tiene el mismo efecto.

@import url;

@import url list-of-media-queries;

Donde :

- **url**: Representa la ubicación del recurso a importar. La url puede ser absoluta o relativa.
- **list-of-media-queries**: Es una lista separada por comas de consultas de medios (media query) que condicionan la aplicación de las reglas CSS definidas en el enlace url. Si el navegador no soporta cualquiera de estos consultas de medios (media query), ni siquiera cargará el recurso vinculado.

```
@import url("fineprint.css") print;  
@import url("bluish.css") projection, tv;  
@import 'custom.css';  
@import "common.css" screen, projection;  
@import url('landscape.css') screen and (orientation:landscape);
```

NOTA: algunos de los elementos que podemos ver en estos ejemplos están en desuso, pero permiten ver cómo es la sintaxis.

@namespace

La regla-at @namespace se utiliza para declarar un prefijo de espacio de nombre en CSS para poder crear selectores asociados solo con este espacio de nombre.

Por defecto los selectores CSS son aplicables a todos los elementos del documento, fuera lo que fuera su espacio de nombres. Veamos un ejemplo:



```
@namespace url(http://www.w3.org/1999/xhtml);
@namespace svg url(http://www.w3.org/2000/svg);

/* This matches all XHTML <a> elements, as XHTML is the default unprefix namespace */
a {}

/* This matches all SVG <a> elements */
svg|a {}

/* This matches both XHTML and SVG <a> elements */
*|a {}
```

Los espacios de nombres se definen después de @charset y @import

Espacios de nombres para

XHTML <http://www.w3.org/1999/xhtml>

<svg> <http://www.w3.org/2000/svg>

<math> <http://www.w3.org/1998/Math/MathML>

@supports

La regla-at @supports nos permite detectar si el navegador del usuario soporta o no las nuevas funcionalidades de CSS.

En el siguiente ejemplo el CSS aplica las reglas entre llaves ({ }) solo si el navegador soporta display: flex. De lo contrario las ignora por completo.

```
/* si el navegador soporta display: flex */
@supports (display: flex) {
/* entonces aplica estas reglas: */
    .elemento {
        display: flex;
        /* y las demás reglas */
    }
}
```

NOTA: Es importante dejar un espacio en blanco delante y detrás de las palabras clave.

La palabra clave not

En el siguiente ejemplo el CSS aplica las reglas entre llaves ({ }) solo si el navegador NO soporta display: flex.

```
/* si el navegador NO soporta display: flex */
@supports not (display: flex) {
/* entonces aplica estas reglas: */
    .elemento {
        display: table;
        /* y las demás reglas */
    }
}
```

Podemos verificar múltiples características utilizando las palabras clave and y/o or

La palabra clave or

En el siguiente ejemplo el CSS aplica las reglas entre llaves ({ }) si el navegador cumple al menos una de las condiciones entre paréntesis, o sea si el navegador soporta display: -webkit-flex o display: -moz-flex o display: flex.

```
@supports (display: -webkit-flex) or
  (display: -moz-flex) or
  (display: flex) {
  .elemento {
    display: -webkit-flex;
    display: -moz-flex;
    display: flex;
    /*aquí van las reglas */
  }
}
```

La palabra clave and

En el siguiente ejemplo el CSS aplica las reglas entre llaves ({ }) solo si el navegador soporta tanto la propiedad column-count como la propiedad break-before.

```
@supports (column-count : 3) and (break-before: right) {
  /*aquí van las reglas */
}
```

Utilizando varias palabras clave

Podemos concatenar múltiples condiciones utilizando varias palabras clave. La sintaxis utilizada en este caso se parece mucho con la de otros lenguajes de programación.

```
@supports not (...) and ((...) or (...)) { /* ... */ }
@supports (...) or (...) (not (...)) { /* ... */ }
```

Ejercicio 2: @supports

Modifique el último fichero .css generado para que si el navegador no soporta css grid utilice flexbox. NOTA: recuerde que flexbox no es tan potente como CSS Grid, así que es difícil que algunos elementos queden exactamente igual que cuando se utiliza CSS Grid.

NOTA: Si no encuentras un navegador que NO soporte CSS Grid, lo que puedes hacer es programar directamente sobre flexbox y luego añadir @support.

@keyframes

La regla-at @keyframes se utiliza para definir una secuencia de fotogramas de una animación CSS.

La sintaxis

Como las demás reglas-at, @keyframes es un bloque de código que empieza con la palabra clave @keyframes seguida por el nombre de la animación. A continuación entre llaves {} aparecen todas las reglas CSS que definen una secuencia de fotogramas.

```
@keyframes nombreAnimacion {  
    /*las reglas para nombreAnimacion */  
}
```

Veamos un caso práctico: una animación donde el color de fondo de un elemento HTML cambiará de rojo (#f00) a rojo oscuro (#e00). La animación cuenta con dos fotogramas: al principio (0%) y al final (100%).

```
@keyframes cambiarColor {  
    0%    { background-color: #f00;}  
    100% { background-color: #e00;}  
}
```

¡Ojo!: los selectores de las fotogramas 0% y 100% son porcentajes y tienen que llevar el símbolo %. También podemos utilizar como selectores las palabras clave **from** (desde) en lugar de 0% y **to** (hasta) en lugar de 100%.

```
@keyframes cambiarColor {  
    from { background-color: #f00;}  
    to   { background-color: #e00;}  
}
```

En el caso de que queramos que el color parpadee podemos escribir algo así:

```
@keyframes cambiarColor {  
    0%    { background-color: #f00;}  
    25%   { background-color: #e00;}  
    50%   { background-color: #f00;}  
    75%   { background-color: #e00;}  
    100%  { background-color: #f00;}  
}
```

O para evitar tanta verbosidad podemos abreviar todo este código así:

```
@keyframes cambiarColor {  
    0%, 50% , 100%    { background-color: #f00;}  
    25%, 75%          { background-color: #e00;}  
}
```

También podemos animar varias propiedades a la vez.

```
@keyframes latido {  
    0%, 50% , 100%    {  
        color: #e00;  
        transform:scale(1);  
    }  
    25%, 75%          {  
        color: #f00;  
        transform:scale(1.25);  
    }  
}
```

Listas válidas de keyframe

Para obtener una lista de keyframe que sea válida, debe incluir reglas para al menos los tiempos 0% (o desde) y 100% (o hacia) (o sea, los estados inicial y final de la

animación). Si ambos desplazamientos de tiempo no se especifican, la declaración keyframe es inválida y no se puede utilizar para la animación.

Si se incluyen las propiedades que no se pueden animar en sus reglas de keyframe, serán ignoradas, pero las propiedades admitidas todavía estarán animadas.

Cualquier propiedad que no se especifican en cualquier keyframes son interpoladas (con la excepción de aquellas que no pueden ser interpoladas, las que son eliminadas de la animación completamente). Por ejemplo:

```
1  @keyframes identifier {  
2    0% { top: 0; left: 0; }  
3    30% { top: 50px; }  
4    68%, 72% { left: 50px; }  
5    100% { top: 100px; left: 100%; }  
6  }
```

En este caso, la propiedad top anima con los keyframes de 0%, 30% y 100%, y left anima utilizando los keyframes 0%, 68%, y 100%.

Sólo las propiedades que se especifican tanto en el 0% y 100% de keyframe serán animadas; cualquier propiedad no incluido en ambos de los keyframes conservarán su valor de partida para la duración de la secuencia de animación.

La especificación define que si un keyframe se define varias veces, pero no todas las propiedades afectadas se especifican en cada keyframe, sólo los valores especificados en el último keyframe se consideran. Por ejemplo:

```
@keyframes identifier {  
  0% { top: 0; }  
  50% { top: 30px; left: 20px; }  
  50% { top: 10px; }  
  100% { top: 0; }  
}
```

En este ejemplo, en el 50% del keyframe, el valor utilizado es top: 10px y todos los demás valores en este keyframe se ignoran.

NOTA: los keyframes (fotogramas clave) en cascada son compatibles a partir de Firefox 14. Para el ejemplo anterior, esto significa que en el fotograma clave 50%, el valor left: 20px será considerado. Esto no está definido en la especificación todavía, pero se está discutiendo.



Es importante que las propiedades modificadas en el bloque de código @keyframes puedan ser animadas (animatables). De lo contrario estas reglas serán ignoradas. . . con una excepción.

La propiedad animation-timing-function (función de temporizador de la animación) puede ser utilizada en el bloque de código @keyframes para especificar la curva de velocidad de la animación.

```
@keyframes latido {  
  0%, 50% , 100%  {  
    color: #e00;  
    transform:scale(1);  
    animation-timing-function: ease-in;;  
  }  
  25%, 75%          {  
    color: #f00;  
    transform:scale(1.25);  
    animation-timing-function: ease-out;;  
  }  
}
```

Para poder aplicar esto tenemos las animaciones CSS

https://developer.mozilla.org/es/docs/Web/CSS/CSS_Animations

Puedes encontrar más animaciones:

<https://daneden.github.io/animate.css/>

Usando validación de formularios

Usando validación de formularios proporcionada por el navegador

Una de las características de HTML5 es la habilidad de validar la mayoría de datos del usuario sin depender de scripts. Esto se hace usando atributos de validación en elementos de formulario.

Cuando un elemento es inválido, ocurren dos cosas:

El elemento coincide con la pseudo-clase de CSS **:invalid**; esto te permitirá aplicar estilos específicos a elementos inválidos. De forma similar, los elementos válidos coinciden con la pseudo-clase **:valid**.

Si el usuario intenta enviar los datos, el navegador bloqueará al formulario y mostrará un mensaje de error.

Restricciones de validación en elementos <input>

Todos los elementos <input> pueden ser validados usando el atributo pattern. Este atributo espera como valor una Expresión Regular, sensible a uso de mayúsculas. Si el valor del

elemento no es vacío y no coincide con la expresión regular especificada en el atributo `pattern`, el elemento es considerado inválido.

```
1 <form>
2   <label for="choose">¿Preferirías un plátano o una cereza?</label>
3   <input id="choose" name="i_like" pattern="plátano|cereza">
4   <button>Enviar</button>
5 </form>
```

```
1 input:invalid {
2   border: 1px solid red;
3 }
4
5 input:valid {
6   border: 1px solid green;
7 }
```

En este ejemplo, el elemento `<input>` acepta uno de tres valores posibles: una cadena vacía, el texto "plátano" o el texto "cereza".

El atributo `required`

Si un elemento requiere un valor antes de que el formulario sea enviado, puedes marcar al elemento usando el atributo `required`. Cuando este atributo es verdadero, no se permite que el campo esté vacío.

```
1 <form>
2   <label for="choose">¿Preferirías un plátano o una cereza?</label>
3   <input id="choose" name="i_like" pattern="plátano|cereza" required>
4   <button>Enviar</button>
5 </form>
```

```
1 input:invalid {
2   border: 1px solid red;
3 }
4
5 input:valid {
6   border: 1px solid green;
7 }
```

Los elementos `<input>` que tengan el atributo `type` con valor `email` o `url` no necesitan un atributo `pattern` para ser validados. Si se especifica el tipo `email`, es necesario que el valor del campo sea una dirección bien formada de correo electrónico (o una lista de



direcciones separadas por coma, si tiene el atributo **multiple**). Los campos con tipo url automáticamente necesitan una URL debidamente formada.

Otras restricciones de validación

Todos los tipos de elemento que pueden recibir entrada del usuario (<textarea>, <select>, etc.) soportan el atributo required, pero cabe aclarar que el elemento <textarea> no soporta el atributo pattern.

Todos los campos de texto (<input> or <textarea>) pueden ser restringidos en tamaño usando el atributo maxlength. Un campo es inválido si su valor es mayor al valor del atributo maxlength. La mayoría de las ocasiones, sin embargo, los navegadores no permiten que el usuario escriba un valor más largo del indicado.

Para campos numéricos, los atributos min y max también proveen una restricción de validación. Si el valor del campo es menor que el del atributo min o mayor que el del atributo max, el campo será inválido.

Aquí hay un ejemplo completo:

```
form>
  <p>
    <fieldset>
      <legend>Title<abbr title="This field is
mandatory">*</abbr></legend>
      <input type="radio" required name="title" id="r1" value="Sr"
><label for="r1">Sr. </label>
      <input type="radio" required name="title" id="r2"
value="Sra"><label for="r2">Sra.</label>
    </fieldset>
  </p>
  <p>
    <label for="n1">¿Cuál es tu edad?</label>
    <!-- El atributo pattern puede actuar como fallback para
navegadores que no implementen el tipo number pero sí soporten el
atributo pattern.
Nótese que los navegadores que soportan el atributo pattern
lo harán fallar silenciosamente cuando se use con un campo numérico.
Su uso aquí solo es como fallback -->
    <input type="number" min="12" max="120" step="1" id="n1"
name="age"
      pattern="\d+">
  </p>
  <p>
    <label for="t1">¿Cuál es tu fruta favorita?<abbr title="Este campo
es obligatorio">*</abbr></label>
    <input type="text" id="t1" name="fruit" list="l1" required
```



```

pattern="[Pp]l[áa]tano|[Cc]ereza|[Mm]anzana|[Ff]resa|[Ll]im[oó]n|[Nn]a
ranja">
    <datalist id="l1">
        <option>Plátano</option>
        <option>Cereza</option>
        <option>Manzana</option>
        <option>Fresa</option>
        <option>Limón</option>
        <option>Naranja</option>
    </datalist>
</p>
<p>
    <label for="t2">¿Cuál es tu dirección de correo
electrónico?</label>
    <input type="email" id="t2" name="email">
</p>
<p>
    <label for="t3">Déjanos un mensaje corto</label>
    <textarea id="t3" name="msg" maxlength="140" rows="5"></textarea>
</p>
<p>
    <button>Enviar</button>
</p>
</form>

body {
    font: 1em sans-serif;
    padding: 0;
    margin : 0;
}

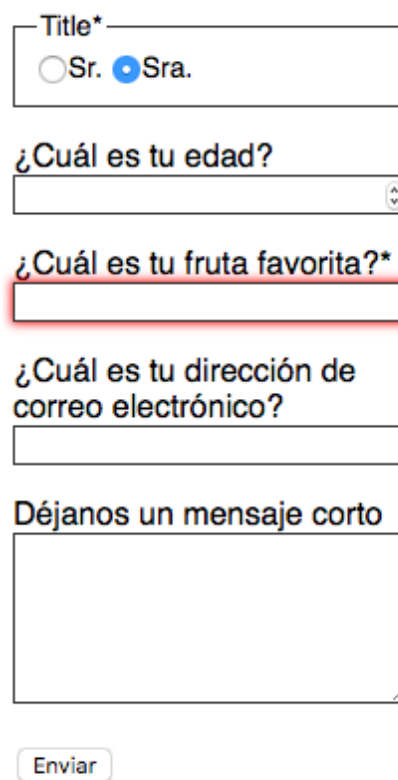
form {
    max-width: 200px;
    margin: 0;
    padding: 0 5px;
}

p > label {
    display: block;
}

input[type=text],
input[type=email],
input[type=number],
textarea,
fieldset {
/* requerido para estilizar adecuadamente elementos

```

```
de formulario en navegadores basados en WebKit */  
-webkit-appearance: none;  
  
width : 100%;  
border: 1px solid #333;  
margin: 0;  
  
font-family: inherit;  
font-size: 90%;  
  
-moz-box-sizing: border-box;  
box-sizing: border-box;  
}  
  
input:invalid {  
    box-shadow: 0 0 5px 1px red;  
}  
  
input:focus:invalid {  
    outline: none;  
}
```



Title*

☐ Sr. ☒ Sra.

¿Cuál es tu edad?

¿Cuál es tu fruta favorita?*

¿Cuál es tu dirección de correo electrónico?

Déjanos un mensaje corto

Enviar

El código completo puedes verlo en

https://developer.mozilla.org/es/docs/Learn/HTML/Forms/Validacion_formulario_datos

Mensajes de error personalizados

Como vimos en los ejemplos anteriores, cada vez que el usuario intenta enviar un formulario inválido, el navegador muestra un mensaje de error. La manera en que estos mensajes son mostrados depende del navegador.

Estos mensajes automatizados tienen dos inconvenientes:

- No hay manera estándar de cambiar su apariencia con CSS.
- Dependen de la localización del navegador, lo que significa que puedes tener una página en un lenguaje específico, pero el mensaje de error se puede en otro lenguaje.



Versión en francés del navegador en una página en inglés

Navegador	Representación
Firefox 17 (Windows 7)	
Chrome 22 (Windows 7)	
Opera 12.10 (Mac OSX)	

Para personalizar la apariencia y texto de estos mensajes, debes usar JavaScript; no hay manera de hacer esto usando solamente HTML y CSS.

Enlaces con las fuentes de la información

- <https://www.arsys.es/blog/programacion/diseno-web/media-queries-css3/>
- https://www.w3schools.com/cssref/css3_pr_mediaquery.asp
- <https://developer.mozilla.org/en-US/docs/Web/CSS/@media>
- <http://w3.unpocodetodo.info/css3/at-keyframes.php>
- <https://developer.mozilla.org/en-US/docs/Web/CSS/At-rule>
-