Curso HTML, CSS y JavaScript

Junio de 2017

#### Autor: Fran Linde Blázquez

# Tema 9: CSS Avanzado

ÍNDICE

[Tema 9: CSS Avanzado 1](#_Toc485979061)

[1. Introducción 3](#_Toc485979062)

[2. Diseño Responsive 4](#_Toc485979063)

[3. Media Queries 5](#_Toc485979064)

[3.1 Sintaxis 5](#_Toc485979065)

[3.2 Operadores Lógicos 6](#_Toc485979066)

[4. Transiciones y animaciones 7](#_Toc485979067)

[4.1 Transiciones 7](#_Toc485979068)

[4.2 Animaciones 8](#_Toc485979069)

[5. Preprocesadores CSS 11](#_Toc485979070)

[6. SASS 12](#_Toc485979071)

[6.1 Variables 12](#_Toc485979072)

[6.2 Funciones 12](#_Toc485979073)

[6.3 Extend 13](#_Toc485979074)

[6.4 Nesting 13](#_Toc485979075)

[7. LESS 14](#_Toc485979076)

[7.1 Variables 14](#_Toc485979077)

[7.2 Mixins 14](#_Toc485979078)

[7.3 Reglas internas 15](#_Toc485979079)

[7.4 Funciones y Operaciones 16](#_Toc485979080)

## 1. Introducción

Hasta ahora solo hemos visto una introducción a CSS. En este capítulo profundizaremos en CSS, viendo nuevas funcionalidades, así como herramientas para mejorar el estilado de nuestras páginas.

## 2. Diseño Responsive

El diseño responsive es una práctica que consiste en colocar los elementos de la web de forma que se adapten al ancho de cada dispositivo permitiendo una correcta visualización y una mejor experiencia de usuario. Se caracteriza por que el contenido e imágenes son fluidos y se usa código media-queries de CSS3.

Con esto conseguimos que todos los usuarios, independientemente del dispositivo con el que visualicen la web, tengan una experiencia similar.

## 3. Media Queries

Las media queries son una característica nueva de CSS3, con ellas podremos aplicar estilos a diferentes dispositivos según su tamaño.

Para que el contenido de la web fluya correctamente es necesario añadir una meta etiqueta a nuestra cabecera que asigne un tamaño de escala inicial a 1:

<head>

...

<meta name="viewport" content="width=device-width, initial-scale=1">

...

</head>

### 3.1 Sintaxis

Una vez hayamos añadido esta meta etiqueta es hora de crear nuestras propias media-queries. La sintaxis es muy sencilla, solo debemos seguir la siguiente estructura de código:

@media (max-width: 480px){

/\*Estilos que queremos añadir\*/

.clase{

color: red;

}

}

Esta media-query que acabamos de crear afectará a dispositivos con un ancho de pantalla del navegador menor a 480px (inclusive). También podríamos hacerlo a la inversa, que aplicara a dispositivos con un ancho mayor a 480px.

@media (min-width: 480px){

/\*Estilos que queremos añadir\*/

.clase{

color: red;

}

}

### 3.2 Operadores Lógicos

Podemos concatenar tantas condiciones queramos con operadores lógicos como and, not y only. Los estilos que añadamos dentro de las media-queries solo tendrán efecto cuando se cumpla correctamente la sentencia.

@media only screen and (min-width: 481px) and (max-device-width: 1024px) and (orientation:portrait) {

/\*Estilos que queremos añadir\*/

.clase{

color: red;

}

}

## 4. Transiciones y animaciones

Otra de las potencias de CSS3 es la posibilidad de realizar efectos de animación que ofrezcan a nuestra web una mayor vistosidad.

Para ello haremos uso de:

### 4.1 Transiciones

Una transición es el estado por el que pasa cualquier elemento de html para adquirir unos estilos diferentes a los que tenía, ya sea para una transformación, un cambio de color, de tamaño...

Estos cambios pueden producirse con **:hover**, un **:focus**, un cambio de clases...No importa el motivo, con unas propiedades especificas nosotros podremos controlar como se realizará esta transición.

Para llevar a cabo este control debemos agregar la propiedad transition a nuestro elemento CSS y agregarle la duración de la transición (en segundos), las propiedades a las que afecta (color, width,opacity, all...) y el intervalo de transición (mediante puntos de una curva Bézier o las palabras clave admitidas). Un ejemplo sería el siguiente:

.link {

color: red;

transition: .5s color ease-in-out;

}

.link:hover {

color: blue;

}

En el anterior ejemplo cuando hagamos **hover** sobre un elemento con la clase **.link**, este cambiará de color rojo a azul en medio segundo haciendo una transición suave tanto al comenzar como al terminar. Este mismo código con la palabra clave **ease-in-out** es igual que si escribiéramos **transition: .5s color cubic-bezier(0.42, 0, 0.58, 1.0)**.

### 4.2 Animaciones

Las animaciones son similares a las transiciones, pero a diferencia de ellas no necesitamos un cambio de estado para que se produzcan. Una vez que apliquemos a un elemento una animación esta se ejecutara haciendo lo que hayamos especificado que haga.

Las animaciones pueden ser tan simples como un cambio de color, o tan complejas como podría ser la animación de un logotipo. Veamos un ejemplo:

@keyframes nombre\_animacion {

from {

color: red;

}

to {

color: blue;

}

}

En este ejemplo hemos creado una animación con el nombre “nombre\_animacion” que hará que un cambio de color de red a blue. También podemos modificar los estilos agregando estados a la animación:

@keyframes nombre\_animacion {

0% {

color: red;

}

50% {

color: yellow;

}

100% {

color: blue;

}

}

Como ves, hemos hecho que la propiedad color pase a amarillo a la mitad de la animación antes de terminar siendo azul. Podemos añadir tantos porcentajes como queramos para modificar la animación a nuestro gusto.

Una vez tenemos definida la animación pasamos a asignársela a un elemento. Para ello tenemos una serie de sub-propiedades de animation.

Las más utilizadas son:

* **Animation-delay:** Tiempo de retardo entre el momento en que el elemento se carga y el comienzo de la secuencia de la animación.
* **Animation-direction:** Indica si la animación debe retroceder hasta el fotograma de inicio al finalizar la secuencia o si debe comenzar desde el principio al llegar al final.
* **Animation-duration:** Indica la cantidad de tiempo que la animación consume en completar su ciclo (duración).
* **Animation-iteration-count:** El número de veces que se repite. Podemos indicar infinite para repetir la animación indefinidamente.
* **Animation-name:** Especifica el nombre de la regla @keyframes que describe los fotogramas de la animación.
* **Animation-play-state:** Permite pausar y reanudar la secuencia de la animación
* **Animation-timing-function:** Indica el ritmo de la animación, es decir, como se muestran los fotogramas de la animación, estableciendo curvas de aceleración.
* **Animation-fill-mode:** Especifica qué valores tendrán las propiedades después de finalizar la animación (los de antes de ejecutarla, los del último fotograma de la animación o ambos).

Un ejemplo práctico de como asignar una animación a un elemento sería:

.link {

animation-name: nombre\_animacion;

animation-duration: .5s;

animation-delay: .5s;

animation-direction: normal;

}

Como podemos ver en el ejemplo anterior, estamos asignando la animación “nombre\_animacion” a la clase .link con una duración y espera de 0.5 segundos cada uno en una dirección normal.

## 5. Preprocesadores CSS

Debido a la dinamización de las webs, CSS se convirtió en algo difícil de mantener. Los ficheros se hacían infinitos, no teníamos variables, ni funciones...Para solucionar esto aparecieron los preprocesadores, que añaden funcionalidad al estándar CSS.

Un preprocesador de CSS es una herramienta que nos permite escribir pseudo-código CSS que luego será convertido a CSS real. Ese pseudo-código se conforma de variables, condiciones, bucles o funciones. Podríamos decir que tenemos un lenguaje de programación que genera CSS.

El objetivo de estos preprocesadores es tener un código más sencillo de mantener y editar. Los preprocesadores incluyen características tales como variables, funciones, mixins, anidación o modularidad. Explicaremos todo ello más adelante.

Para hacer uso de preprocesadores necesitaremos montar un entorno que realice al menos el compilado de nuestros ficheros. Es común encontrar entornos con **Grunt**, **Gulp**, **NPM** o **Webpack** como motores que arranquen nuestras tareas de compilado.

## 6. SASS

Sass dispone de dos tipos de formato. El primero en salir fue. sass y se caracteriza, al igual que Stylus, en no hacer uso de llaves ni punto y coma final, en busca de la simplicidad y la eliminación de ruido. Más tarde, con la versión 3, salió. scss permitiendo la utilización de llaves y punto y coma final como el CSS clásico.

### 6.1 Variables

Permite reutilizar valores que podemos manejar desde un solo sitio de forma sencilla y centralizada.

$brand: #F98355;

body {

color: $brand;

}

### 6.2 Funciones

Es un elemento esencial cuando se trabaja bajo la premisa DRY **(Don’t Repeat Yourself).**

Un mixin permite aprovechar un trozo de nuestro código al que podemos llamar repetidamente y que encapsula en una o más líneas de código.

Un ejemplo podría ser:

@mixin circle($width, $color) {

width: $width;

height: $width;

border-radius: 100%;

background: $color;

}

.new-circle {

@include circle(50px, #000);

}

### 6.3 Extend

Hay ciertos momentos en los que deseamos aprovechar una clase o variarla ligeramente. SASS nos brinda la posibilidad de poder reutilizar este código anterior y añadir algún elemento nuevo, si así lo deseamos. Veámoslo en un ejemplo.

.circle {

width: 50px;

height: 50px;

border-radius: 100%

}

.black-circle {

@extend .circle;

color: #000;

}

### 6.4 Nesting

Usando preprocesadores deberíamos evitar códigos como el siguiente:

div a:hover {

color: green;

}

**SASS** nos ayuda a simplificar haciendo uso del nesting o anidación. Conseguimos escribir menos, con la indentación del código CSS podemos reflejar mejor la jerarquía:

div {

a {

&:hover {

color: green;

}

}

}

## 7. LESS

### 7.1 Variables

Las variables permiten especificar valores que van a ser utilizados en un único lugar, y además poder reusarlas en otras partes de la hoja de estilo, pudiendo realizar cambios globales modificando una única línea de código.

/\* LESS \*/

@color: #4D926F;

#header {

color: @color;

}

h2 {

color: @color;

}

/\* CSS compilado \*/

#header {

color: #4D926F;

}

h2 {

color: #4D926F;

}

### 7.2 Mixins

Los mixins permiten incluir todas las propiedades de una clase en otra diferente añadiendo simplemente el nombre de la clase como una propiedad más. El comportamiento es similar al de las variables, pero con clases enteras. Los mixins pueden comportarse también como funciones, y tomar argumentos, como se ve en el siguiente ejemplo.

/\* LESS \*/

.rounded-corners (@radius: 5px) {

-webkit-border-radius: @radius;

-moz-border-radius: @radius;

-ms-border-radius: @radius;

-o-border-radius: @radius;

border-radius: @radius;

}

#header {

.rounded-corners;

}

#footer {

.rounded-corners(10px);

}

/\* CSS compilado \*/

#header {

-webkit-border-radius: 5px;

-moz-border-radius: 5px;

-ms-border-radius: 5px;

-o-border-radius: 5px;

border-radius: 5px;

}

#footer {

-webkit-border-radius: 10px;

-moz-border-radius: 10px;

-ms-border-radius: 10px;

-o-border-radius: 10px;

border-radius: 10px;

}

### 7.3 Reglas internas

En vez de escribir selectores con nombres largos para especificar una relación de herencia, en Less puedes anidar selectores dentro de otros. Esto hace la herencia más clara y reducir el tamaño de la hoja de estilo.

/\* LESS \*/

#header {

h1 {

font-size: 26px;

font-weight: bold;

}

p {

font-size: 12px;

a {

text-decoration: none;

&:hover {

border-width: 1px

}

}

}

}

/\* CSS compilado \*/

#header h1 {

font-size: 26px;

font-weight: bold;

}

#header p {

font-size: 12px;

}

#header p a {

text-decoration: none;

}

#header p a:hover {

border-width: 1px;

}

### 7.4 Funciones y Operaciones

¿Existe dependencia entre elementos de tu hoja de estilo? Las operaciones te permiten sumar, restar, multiplicar y dividir los valores de las propiedades y colores, dándote el poder de crear relaciones complejas entre propiedades. Las operaciones deberían estar definidas entre paréntesis para asegurar la compatibilidad con CSS. Las funciones se corresponden a funciones JavaScript, permitiéndote modificar valores como necesitemos.

/\* LESS \*/

@the-border: 1px;

@base-color: #111;

@red: #842210;

#header {

color: (@base-color \* 3);

border-left: @the-border;

border-right: (@the-border \* 2);

}

#footer {

color: (@base-color + #003300);

border-color: desaturate(@red, 10%);

}

/\* CSS compilado \*/

#header {

color: #333;

border-left: 1px;

border-right: 2px;

}

#footer {

color: #114411;

border-color: #7d2717;

}