UNIDAD 3

DISEÑO DE INTERFACES WEB: SASS.

EXTENDS

MEDIA QUERIES CON SASS

GRADIENTES, FILTROS Y TRANSFORMACIONES

Profesora: **Rosa María Medina Gómez**
Cefire Específico de FP de Cheste
Generalitat Valenciana
Curso 2019 - 2020

TABLA DE CONTENIDO

UNIDAD 3

Introducción

En esta unidad vamos a ver qué son y cómo usar un **extend**, **media query**, **sprite**, y las **funciones matemáticas y colores** (gradientes, filtros y transformaciones).

Extends

En la unidad anterior, vimos que este CSS:

application.css

```
.btn-a{
     background: #777;
     border: 1px solid #ccc;
     font-size: 1em;
     text-transform: uppercase;
}
.btn-b{
     background: #ff0;
     border: 1px solid #ccc;
     font-size: 1em;
     text-transform: uppercase;
}
```

Lo podíamos simplificar usando:

application.css

```
.btn-a, .btn-b{
     background: #777;
     border: 1px solid #ccc;
     font-size: 1em;
     text-transform: uppercase;
}
.btn-b{
     background: #ff0;
}
```

Mediante **Sass** podemos combinar estos selectores usando la directiva `extend`:

_buttons.scss

```
.btn-a {
     background: #777;
     border: 1px solid #ccc;
     font-size: 1em;
     text-transform: uppercase;
}
```

```
.btn-b{
     @extend .btn-a;
     background: #ff0;
}
```

Una vez utilizado el `extend` al compilar te genera:

application.css

```
.btn-a,
.btn-b{
     background: #777;
     border: 1px solid #ccc;
     font-size: 1em;
     text-transform: uppercase;
}
.btn-b{
     background: #ff0;
}
```

¿Qué es lo que ocurriría si usamos selectores **nested** y usamos **extend**?

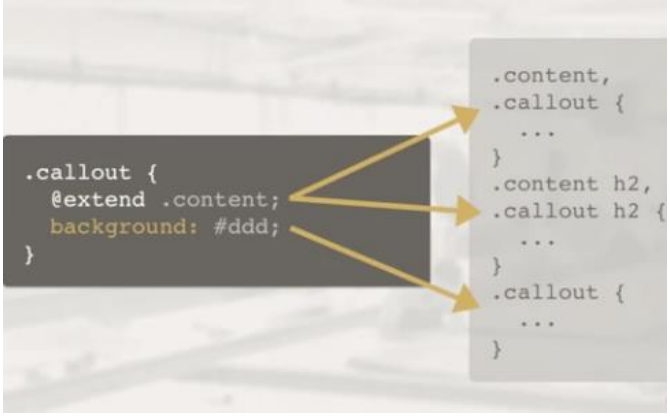
application.scss

```
.content{
     border: 1px solid #ccc;
     padding: 20px;
     h2{
          font-size: 3em;
          margin: 20px 0;
     }
}
.callout{
     @extend .content;
     background: #ddd;
}
```

En este ejemplo, estamos extendiendo `.content` en nuestra declaración, pero `.content` tiene un bloque para los `h2` dentro. Cuando compilamos el código anterior, no solo se añade lo que contiene `.content` a `.callout`, sino que también hereda el `h2`.

application.css

```
.content,
.callout{
     border: 1px solid #ccc;
     padding: 20px;
}
.content h2,
.callout h2{
     font-size: 3em;
     margin: 20px 0;
}
.callout{
     background: #ddd;
}
```



`extend` es bastante útil ya que podemos combinar selectores en nuestra hoja de estilos. Pero tiene algunos problemas. Por ejemplo:

_buttons.scss

```
.btn-a{
     background: #777;
     border: 1px solid #ccc;
     font-size: 1em;
     text-transform: uppercase;
}
.btn-b{
     @extend .btn-a;
     background: #ff0;
}
.sidebar .btn-a{
     text-transform: lowercase;
}
```

Al heredar . `btn-b` las propiedades de . `btn-a`, las hereda tanto del selector `.btn-a` como también del selector `.sidebar .btn-b` (cuando `.btn-a` está dentro de `.sidebar`), por lo que también se creará una regla idéntica para cuando `btn-b` esté dentro de `sidebar`.

application.css

```
.btn-a,
.btn-b{
     background: #777;
     border: 1px solid #ccc;
     font-size: 1em;
     text-transform: uppercase;
}
```

```
.btn-b{
     background: #ff0;
}
.sidebar .btn-a,
.sidebar .btn-b {
     text-transform: lowercase;
}
```

Por tanto, desde el momento en el que el `.btn-b` extiende de `.btn-a`, cualquier cambio que hagas desde `.btn-a` usando otros selectores afectará a `.btn-b`. Esto lo podemos solventar usando selectores "*placeholder*" como veremos a continuación.

Los selectores *placeholder* en Sass están precedidos por el símbolo del porcentaje (**%**), además, pueden ser extendidos, pero nunca serán un selector en sí mismos. Si volvemos al ejemplo anterior de los botones y usamos un *placeholder* de forma que las partes comunes las agrupamos y hacemos que `.btn-a` y `.btn-b` extiendan del *placeholder*.

_buttons.scss

```
%btn{
     background: #777;
     border: 1px solid #ccc;
     font-size: 1em;
     text-transform: uppercase;
}
.btn-a{
     @extend %btn;
}
.btn-b{
     @extend %btn;
     background: #ff0;
}

.sidebar .btn-a{
     text-transform: lowercase;
}
```

Con esto al compilar tenemos:

Los selectores placeholder son bastante útiles cuando queremos reusar estos estilos CSS en nuestra hoja de estilos. Aquí tenemos otro ejemplo:

_buttons.scss

```
%ir{
     border: 0;
     font: 0/0 a; // anulamos la fuente
     text-shadow: none;
     color: transparent;
     background-color: transparent;
}
.logo{
     @extend %ir;
}
.social{
     @extend %ir;
}
```

Al compilarlo por tanto tendremos:

application. css

```
.logo,
.social{
     border: 0;
     font: 0/0 a;
     text-shadow: none;
     color: transparent;
     background-color: transparent;
}
```

Por último, antes de acabar este punto, decir que debemos tener cuidado ya que las versiones anteriores a IE9 tiene el límite de 4095 selectores CSS por archivo, por tanto, si usamos muchos `extends` o muchos `imports` puede que alcancemos el total de 4095 selectores que tiene como límite, si se llega a sobrepasar ese límite todo lo que se sobrepase será ignorado.

Math + Color

Operaciones aritméticas básicas

Con Sass tenemos la posibilidad de realizar operaciones con números (+, -, *, /, %). Por defecto, cuando realizamos una operación matemática en Sass nos devuelve, si el resultado es decimal, 5 dígitos tras la coma redondeados hacia arriba.

Ejemplo: `font: normal 2em/1.5 Helvetica, sans-serif;`

Concatenación de strings

La forma de concatenar **string** en Sass es mediante el operador +:

```
font: "Helvetica " + "Neue"; // "Helvetica Neue"
```

Cuando concatenamos **string**, si el operando de la izquierda tiene comillas simples, el resultado es un **string** con comillas simples. Sin embargo, si no va entre comillas, el resultado será sin ellas (aunque el otro operando las tenga).

```
font: 'sans-' + serif; // 'sans-serif'
font: sans- + 'serif'; // sans-serif
```

Cuando realizamos operaciones en Sass donde las unidades entre las operaciones que realizamos varían como, por ejemplo:

application.scss

```
h2{
     font-size: 10px + 4pt;
}
```

Aquí estamos sumando píxeles y puntos. Sass nos hace la conversión de forma que tras compilarlo obtenemos:

application. css

```
h2{
     font-size: 15.33333px;
}
```

Si usamos por ejemplo unidades relativas como **em**, no es capaz de hacer esta conversión y nos devuelve un error diciéndonos que no son compatibles las unidades:

application.scss

```
h2{
     font-size: 10px + 4em;
}
```

application. css

```
Incompatible units: 'em' and 'px'.
```

Funciones matemáticas

Hay un gran número de funciones matemáticas predefinidas en Sass:

- **round**($number): Redondea al número entero más cercano
- **ceil**($number): Redondea hacia arriba
- **floor**($number): Redondea hacia abajo
- **abs**($number): Obtiene el valor absoluto
- **min**($list): Obtiene el número más pequeño de una lista
- **max**($list): Obtiene el número más grande de una lista
- **percentage**($number): Convierte el número en un porcentaje

¿Cómo podemos utilizar estas funciones?

La forma que hay de invocar estas funciones es igual que si las hubiéramos creado nosotros. Por ejemplo:

application.scss

```
$context: 1000px;
h2{
     line-height: ceil(1.2);
}
.sidebar{
     width: percentage(350px/$context);
}
```

application.css

```
h2{
     line-height: ceil(2);
}
.sidebar{
     width: 35%;
}
```

Estas funciones matemáticas son fáciles de recordar, además simplifican bastante los cambios de colores de los elementos de una web, vamos a ver un ejemplo:

application.scss

```
$color-base: #333;
.addition{
     background: $color-base + #123;
}
```

Cuando compilamos este archivo, nos genera el color resultante de haber sumado en hexadecimal *#333333 + #112233 = #445566*.

application.css

```
.addition{
     background: #456;
}
```

De la misma forma, podríamos haber restado, multiplicado o dividido colores:

application.scss

```
$color-base: #333;
.subtraction{
     background: $color-base - #123;
}

.multiplication{
     background: $color-base *2;
}
.division{
     background: $color-base /2;
}
```

Obteniendo:

application.css

```
.subtraction{
     background: #210;
}
.multiplication{
     background: #666;
}
.division{
     background: #191919;
}
```

Dentro de este rango de funciones podemos encontrar:

- **lighten**($color, $amount): Genera un color más claro
- **darken**($color, $amount): Genera un color más oscuro
- **saturate**($color, $amount): Modifica la intensidad del color (añadiéndole)
- **desaturate**($color, $amount): Modifica la intensidad del color (quitándole)
- **mix**($color1, $color2, [$weight]): Mezcla dos colores, el tercer parámetro es opcional y lo que hace es indicar el % del primer color que usamos
- **grayscale**($color): Convierte un color en escala de grises
- **invert**($color): Devuelve el inverso de un color
- **complement**($color): Devuelve el complementario.

Ejemplo:

application.scss

```
$color-base: #333;
$color-escala: #87bf64;
.lighten{
     color: lighten($color-base, 20%);
}
.darken{
     color: darken($color-base, 20%);
}
.saturate{
     color: saturate($color-base, 20%);
}
.desaturate{
     color: desaturate($color-base, 20%);
}
.mix-a{
     color: mix(#4ff0, $107fc9);
}
.mix-b{
     color: mix(#4ff0, $107fc9, 30%);
}
.grayscale{
     color: grayscale($color-escala);
}
.invert{
     color: invert($color-escala);
}
.complement{
     color: complement($color-escala);
}
```

Obtenemos:

application.css

```
.lighten{
     color: #666;
}
.darken{
     color: black;
}
.saturate{
     color: #82d54e;
}
.desaturate{
     color: #323130;
}
.mix-a{
     color: #87bf64;
}
.mix-b{
     color: #57a58c;
}
.grayscale{
     color: #929292;
}
.invert{
     color: #78409b;
}

.complement{
     color: #9c64bf;
}
```

Para aprender más sobre funciones en Sass: http://sass-lang.com/documentation/Sass/Script/Functions.html

Responsive

El diseño **responsive** se ha convertido en una práctica muy común hoy en día. Uno de los ejemplos típicos es el de conocer las características del dispositivo para, dependiendo del mismo, utilizar un estilo u otro.

Vamos a ver un ejemplo básico de cómo funciona una *media query* en CSS:

application.css

```
.sidebar{
     border: 1px solid #ccc;
}
```

```
@media(min-width: 700px){
     // Para los dispositivos con al menos 700px de ancho
     .sidebar{
          float: right;
          width: 30%;
     }
}
```

Si nos fijamos estamos modificando el `.sidebar` dependiendo de la resolución del dispositivo. Con Sass una *media query* la haríamos:

application.scss

```
.sidebar{
     border: 1px solid #ccc;
     @media(min-width: 700px){
          float: right;
          width: 30%;
     }
}
```

Y automáticamente se compilaría al CSS que mostramos antes.

Podemos combinar fácilmente las *media queries* con mixins utilizando la directiva `@content`. Lo que hacemos con esta directiva es incluir un bloque de propiedades dentro un mixin. Por ejemplo:

application.scss

```
@mixin respond-to{
     @media(min-width: 700px){
          @content
     }
}
.sidebar{
     border: 1px solid #ccc;
     @include respond-to{
          float: right;
          width: 30%;
     }
}
```

Una de las prácticas más frecuentes en estos casos es enviar al mixin un argumento sobre el media, de forma que se realice esta comprobación, por ejemplo, aplicar un estilo sólo si estamos usando una tablet:

application.scss

```
@mixin respond-to($media){
     @if $media == tablet{
          @media(min-width: 700px){
               @content
          }
     }
}

.sidebar{
     border: 1px solid #ccc;
     @include respond-to(tablet){
          float: right;
          width: 30%;
     }
}
```

En esta declaración podríamos añadir un`else` o `else-if` si queremos seguir realizando comprobaciones para los diferentes dispositivos. Sin embargo, esto no es todo lo flexible que debería. Una solución más correcta sería:

application.scss

```
@mixin respond-to($query){
     @media(min-width: $query){
          @content
     }
}

.sidebar{
     border: 1px solid #ccc;
     @include respond-to(900px){
          float: right;
          width: 30%;
     }
}
```

Con esto, lo que conseguimos es que se incluya si se cumple con el ancho mínimo que nosotros definimos, de forma que podamos usar este mixin en cualquier lugar donde tengamos un estilo para un valor `min-width`. Pero si queremos que por ejemplo este mixin no sólo se use para un mínimo, sino que pueda ser para un máximo también, la posibilidad que tenemos para modificarlo sería:

application.scss

```
@mixin respond-to($val, $query){
     @media($val: $query){
          @content
     }
}

.sidebar{
     border: 1px solid #ccc;
     @include respond-to(max-width, 600px){
          float: right;
          width: 30%;
     }
}
```

Bibliografía y/o páginas de interés

- Sass: http://sass-lang.com/
- Node: https://nodejs.org/es/
- Sass Wikipedia: https://en.wikipedia.org/wiki/Sass_(stylesheet_language)
- Less wikipedia: https://en.wikipedia.org/wiki/Less_(stylesheet_language)
- Stylus wikipedia: https://en.wikipedia.org/wiki/Stylus_(stylesheet_language)
- scout: https://scout-app.io/