Preprocesadores CSS. Sass

1. Introducción a los preprocesadores CSS

CSS es un lenguaje para definir y establecer las propiedades de estilo que tienen los elementos de una interfaz. Los selectores que se utilizan para aplicar esos estilos pueden ser más o menos específicos y existen algunas opciones para agrupar la forma de aplicarlos, de forma más mantenible. Sin embargo, la sencillez del lenguaje CSS hace que también tenga sus limitaciones.

Por ejemplo, si se necesita cambiar determinado color para que sea igual en muchos elementos, o se precisa que una propiedad sea relativa a otro valor, o establecer condicionales, etc., CSS es demasiado rígido.

Con el fin de poder desarrollar estilos que sean más flexibles, mantenibles y, sobre todo, con una sintaxis más rica, se han desarrollado diferentes **preprocesadores CSS**.

Con los preprocesadores CSS se crean ficheros similares a las hojas de estilo pero con elementos de los lenguajes de programación: variables, estructuras anidadas, operadores, condicionales, includes, etc.

Estos ficheros se compilan y se traducen a hojas de estilo CSS válidas para los navegadores.

Definición: https://developer.mozilla.org/es/docs/Glossary/Preprocesador_CSS

Los prepocesadores más populares son *Less* y *Sass*. Bootstrap, desde su versión 4, utiliza este último para su personalización.

2. Instalación

- https://sass-lang.com/install
- Compilar fichero: *sass fichero.scss fichero.css*
- Ejecución en modo **watch** (poner en observación un directorio y hacer que todos los cambios que se produzcan sean compilados de inmediato):
  sass --watch app/src:app/css
- Si no hay errores de sintaxis, se obtiene como resultado un fichero CSS válido.
- Sass permite dos tipos de ficheros fuente distintos: **.sass** y **.scss**. El primero se basa en tabulaciones, no utiliza llaves, puntos ni comas. El segundo es una versión más reciente, siendo más similar a un fichero CSS (opción recomendada).
- Conceptos básicos de Sass: http://sass-lang.com/guide
- Referencia completa: https://sass-lang.com/documentation

*Posible error de permisos en Powershell (abrir como administrador): `Set-ExecutionPolicy Unrestricted`

- Opcional: extensión VSCode "Live Sass Compiler" by Glenn Marks

3. Operadores, variables, listas y mapas

- **Comentarios** en Sass:
  - // >> no se compila
  - /* */ >> sí se compilan
- Las **variables** son identificadores precedidos de **$**
- **Tipos de datos**:
  - Números
  - Cadenas
  - Colores
  - Booleanos
  - Null
  - Mapas y listas
- Sass posibilita usar los **operadores aritméticos** habituales: (+, -, *, /, %), (<, >, <=, >=), (==, !=)
- Se pueden hacer **operaciones** sobre las variables:

$size: 2px * 5;

>> Documentación operadores: https://sass-lang.com/documentation/operators

- Ámbito de las variables(scope):
  - ***Variables globales:*** disponibles a lo largo de todo el código Sass; se definen fuera de los bloques (**{}**), normalmente al principio de los ficheros Sass.
  - ***Variables locales:*** declaradas dentro del bloque de un selector; solo están disponibles y tienen valor para ese selector y sus descendientes.

>> Documentación variables: https://sass-lang.com/documentation/variables

- **Listas**
  - Son colecciones de valores de datos
  - Ejemplo: $sizes: (40px, 80px, 160px);
  - Acceso a un elemento concreto: nth($sizes, 2); //80px; (DEPRECATED)
  - También con list.nth($sizes, 2);
    *necesario @use 'sass:list';

>> Documentación funciones nativas que ofrece Sass para trabajar con listas: https://sass-lang.com/documentation/values/lists

- **Mapas**
    - Colecciones de valores de datos a los que accedemos mediante una clave
    - Ejemplo:
      $breakpoint: ("pequeño": 576px, "medio": 768px, "grande": 992px);
    - Acceso: map-get($breakpoint, "pequeño");
    - También con map.get($breakpoint, "pequeño");
      *necesario @use 'sass:map';

>> Documentación funciones nativas que ofrece Sass para trabajar con mapas: https://sass-lang.com/documentation/values/maps

4. Módulos y partials

IMPORTAR FICHEROS (MODULARIZACIÓN)

- **@use**, **@forward** y **@import** son herramientas para hacer código más modular.
- Los archivos que serán incluidos en otros se denominan **partials** y deben comienzar por _ (guion bajo). No son compilados.

Por ejemplo, definimos un módulo para las variables:

_variables.scss

```
$color1: #001122;
$color2: #0011FF;
```

- Los módulos se importan en otros archivos mediante la at-rule **@use**. No hace falta poner **el guion bajo ni la extensión** a la hora de llamar al módulo.
- Para referenciar una variable importada de otro módulo, hay que poner delante el nombre del módulo (espacio de nombres).

En el ejemplo:

app.scss

```
@use 'variables';

h1 {
    color: variables.$color1;
}
```

- Variables **!default**
    - Mecanismo que nos permite darle valor a las variables desde otro módulo.

Ejemplo:

_variables.scss

```
$color1: tomato !default;
```

app.scss

```
@use 'variables' with (
    $color1: lemonchiffon
);
h1 {
    color: variables.$color1;
}
```

*Si no indicamos !default, tendremos un error de compilación al tratar de cambiar el valor de la variable.

- Alias

```
@use 'variables' as v;
h1 {
color: v.$color1;
}
```

@import (deprecated)

- Mecanismo de Sass para separar las reglas en distintos ficheros (partials) y luego importarlas o reutilizarlas en el fichero principal. Usado en Bootstrap.
  ...
  @import "variables";
  @import "functions";
  ...

- Ficheros (deben comenzar por "_"):
  ...
  _functions.scss
  _variables.scss
  ...
- Documentación: https://sass-lang.com/documentation/at-rules/import

@use

La directiva @import presenta varios problemas:

- los **miembros** (variables, funciones, mixins) cargados tienen un **alcance global**. Esto significa que podemos acceder a ellos desde cualquier hoja de estilos, haciendo muy difícil saber dónde fueron declarados.

@use soluciona este problema:

- los miembros solo pueden ser visibles en la hoja de estilos en la que son cargados**.** Debido a esto, podremos nombrar nuestros miembros de una manera muy simple y no tendremos ningún problema gracias a que podemos referenciar de qué hoja de estilos provienen sin que haya conflictos entre nombres.

Es una regla reciente que sustituye a @import (quedará obsoleta próximamente).

Documentación: https://sass-lang.com/documentation/at-rules/use

@forward

Prepara a los partials para ser usados con @use. El contenido del fichero especificado en @forward no se utiliza en ese archivo. No se compilan. Idóneos para crear un único fichero que actúa como punto de entrada (entrypoint).

Documentación: https://sass-lang.com/documentation/at-rules/forward

5. Anidación (nesting)

- Ayuda a estructurar el código mediante anidación de etiquetas, facilitando la legibilidad de los ficheros css, haciéndolos similares a los HTML.
- Mediante **&** podemos referenciar al **elemento padre**.

Por ejemplo:

```
// Estructura en CSS
    .card {
        ...
    }
    .card:hover {
        ...
    }
```

```
// Alternativa en Sass usando & para referirse al selector padre
    .card {
        ...
        &:hover {...}
    }
// ...que no daría el mismo resultado que:
    .card {
        ...
        :hover {...}
    }

// Otro ejemplo en el que se facilita la escritura de media-queries (la media-query afecta al padre, en este caso a menú)
    .menu {
        @media (min-width: 576px) {
            font-size: 5rem;
            flex-direction: row;
        }
    }
```

- En el selector podemos usar cualquier **combinador** en cualquier lugar, al igual que css.

```
// Ejemplo 1:
    .card > {
        .item {
            display: none;
        }
        span {
            font-weight: bold;
    }
```

// Ejemplo 2:

```
.card {
    > .item {
        display: none;
    }
    span {
        font-weight: bold;
    }
```

>> Ejemplos y documentación: https://sass-lang.com/documentation/style-rules#nesting

6. Interpolación

- Mediante este mecanismo que ofrece Sass podemos **inyectar una expresión** en casi cualquier lugar de un documento Sass (variables, selectores, nombres de propiedades, comentarios, cadenas, funciones,...), cuyo resultado, tras ser evaluada la expresión, formará un trozo de código CSS.
- Sintaxis: **#{expresión_a_evaluar}**
- Ejemplo:

```
$buttton-type: "error";
$btn-color : #f00;
.btn-#{$button-type} {
    background-color: $btn-color;
}
```

*Si no usaramos la interpolación, al crear la clase .btn-error tendríamos un error al compilar.

*La interpolación convierte a cadena de texto sin comillas.

>> Documentación: https://sass-lang.com/documentation/interpolation

7. Funciones

- En Sass es posible definir funciones, de la forma:

```
@function nombre($parámetro1, $parámetro2,...) {

    @return ...;

}
```

- Deben retornar un valor obligatoriamente.
- Pensadas principalmente para realizar operaciones.
- Documentación funciones: https://sass-lang.com/documentation/at-rules/function

Built-in modules (funciones predefinidas)

- Sass proporciona varios módulos que incorporan muchas funciones útiles y mixins, como:
  - sass:math
  - sass:string
  - sass:color
  - sass:list
  - sass:map
  - sass:selector
  - sass:meta
- Se cargan con @use:

```
@use "sass:math";

@debug math.div(10, 2);
```

- Funciones predefinidas de Sass: https://sass-lang.com/documentation/modules

8. Directivas de control de flujo

- Sass ofrece distintas estructuras de control que nos permitirán incluir estilos en base a ciertas condiciones o construir conjuntos de estilos similares con pequeñas variaciones. Estas directivas van principalmente enfocadas en la generación de mixins o funciones reutilizables.
- @if / @else / @else if
- for:
  @for $p from X through Y {
  ...
  #{$p} //acceso a los elementos
  }
- @each:

$breakpoints: 480px, 576px, 768px, 996px;

@each $i in $breakpoints {
...
}

- while:

  @while $p < X {
  ...
  $p:$p+1;
  }
- Documentación: https://sass-lang.com/documentation/at-rules/control

9. Mixins

- Los mixins definen estilos que podremos **reutilizar** a lo largo de la hoja de estilos. Evitamos así el uso de clases genéricas.
- Definición del mixin: **@mixin** *nombre_del_mixin* { ..... }
- Uso del mixin: **@include** *nombre_del_mixin*;
- **Admite argumentos**, incluso opcionales.
- Ejemplo:

```
@mixin centrado-flex {

  display: flex;
  justify-content: center;
  align-content: center;
}

header {

  @include centrado-flex;
}
footer {

  @include centrado-flex;
}
```

- Documentación: https://sass-lang.com/documentation/at-rules/mixin

10. @extend (extensión de selectores)

- Mediante **@extend** podemos reutilizar los estilos definidos en una clase en otra.
- Similar a los mixins, pero @extend no admite argumentos. Orientado a ser usado en clases.
- Ejemplo:

```
  .error {
    border: 1px #f00;
    background-color: #fdd;
  }

.error--serious {
  @extend .error;
  border-width: 3px;
 }
```

- Selectores no permitidos en @extend:
  - Solo se permiten selectores individuales (especificidad al mínimo): clases, id, elementos html, ...
  - Por ejemplo, no podríamos hacer un **@extend h1.title**
- Documentación: https://sass-lang.com/documentation/at-rules/extend

11. Otras directivas

DEPURACIÓN

@debug, @error, @warning

Permiten mostrar mensajes y valores de variables y funciones durante el proceso de desarrollo por la terminal.

@debug expresión;

Se pueden incluir interpolaciones.