TEMA 6. PREPROCESADORES

1. INTRODUCCIÓN

El lenguaje de hojas de estilo CSS permite trabajar con otros recursos adicionales que agilizan la programación de este tipo de archivos de estilo, se trata de los **preprocesadores CSS,** los cuales permiten trabajar de forma fluida y estructurada con las hojas de estilo.

Hasta ahora si se deseaba modificar el valor de una propiedad había que revisar línea a línea de código CSS buscándola e ir cambiado su valor en cada aparición, los preprocesadores permitirán que esta tarea resulte más sencilla, gracias al uso de variables o bloques de código (mixin).

2. SASS

Uno de los preprocesadores más comunes en la actualidad es **Sass**, **Syntactically Awesome Sytle Sheets**, que significa "Hojas de Estilo Sintácticamente Increíbles". El framework Bootstrap 4, se basa en este preprocesador.

> ***Toma nota:***
>
> **Puedes encontrar información sobre Sass desde el sitio https://sass-lang.com**

Sass es una herramienta **multiplaforma** escrita en Ruby para el desarrollo de hojas de estilo estructuradas. Presenta dos tipos de sintaxis (**Sass y SCSS**), cuya diferencia radica en ciertos aspectos de implementación en las reglas de formato que se utilizan para programar hojas de estilo en CSS. Por ejemplo, la sintaxis Sass elimina los ";" por saltos de línea o las llaves. Lo habitual es utilizar la sintaxis SCSS en Sass.

2.1. DESCARGA E INSTALACIÓN DE SASS

La instalación de Sass se puede realizar a través de línea de comandos, se trata de la opción más aconsejable, es un proceso sencillo que permite tener instalado Sass en pocos minutos.

Sass es un gem de Ruby por lo que es **necesario instalar Ruby** en el equipo donde se va a utilizar Sass.

En función del sistema operativo la descarga e instalación de Ruby se realiza:

- **Windows**. Descargar el paquete de instalación desde el sitio web, ejecutarlo e instalarlo en el equipo. https://rubyinstaller.org

- **Mac**. Ruby ya está pre-instalado en el sistema, aunque si presenta una versión demasiado anticuada es conveniente actualizarla a versiones posteriores, utilizando rvm install, seguido de la versión que se quiera instalar.

rvm install ruby-version_actual

- **Linux**. Habitualmente aparece pre-instalado en Linux, si no es así se utiliza la siguiente instrucción por línea de comandos en el terminal.

$ sudo apt-get install ruby

Tras la instalación de Ruby se realizará la instalación de Sass, para ello desde línea de comandos, (**cmd** en el caso de Windows o desde el **terminal** en Mac y Linux) la siguiente instrucción:

$ sudo gem install sass

2.2. DIFERENCIA ENTRE SASS Y SCSS

Sass y SCSS son dos sintaxis diferentes para el preprocesador Sass, si bien ambas presentan la misma funcionalidad, existen ciertas diferencias en cuanto a su sintaxis. En este capítulo se utilizará la sintaxis de SCSS, a continuación se muestran algunas de las diferencias principales.

Sass, se trata de una sintaxis con sangría en la que no se utilizan llaves o puntos y coma (como si ocurre con css), en su lugar, se utilizará la indentación, es decir, se usará la sangría. Es importante utilizarla, puesto que de lo contrario no será correctamente compilado y convertido a css.

```
$color: #3498AA
 body
    font-family: Arial
    color: $color
 .contenedor
    width: 100%
    margin: 0 auto
```

Por el contrario SCSS, sí utiliza llaves {} o puntos y coma, por lo que no será necesaria la sangría, que siempre resulta conveniente utilizarla para claridad en la lectura del código.

```
$color-: #3498AA;
body {
    font-family: Arial;
    color: $color;
}
.contendor {
    width: 100%;
    margin: 0 auto;
}
```

2.3. HTML Y SASS

El desarrollo en Sass se basa en la implementación del código de estilo en un documento de tipo SCSS, el cual requiere de cierta conversión a formato CSS para poder ser reconocido por un fichero en HTML. Una de las ventajas de este preprocesador es que actualiza de forma automática el contenido del fichero CSS.

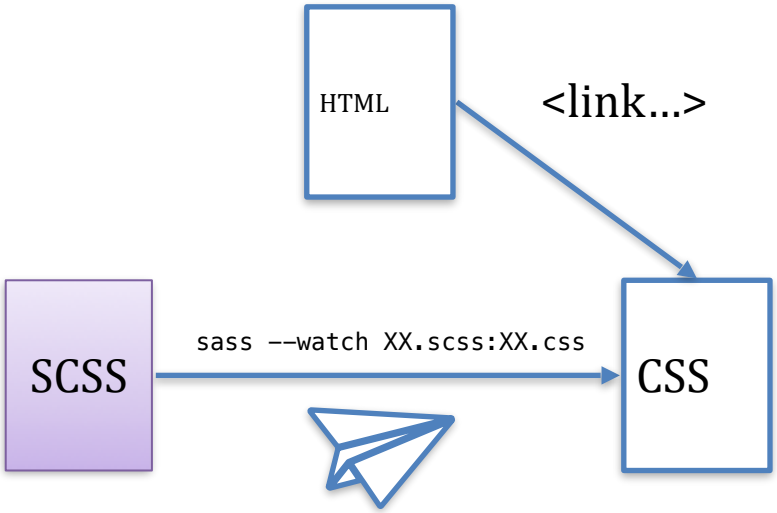

Figura 1. Diagrama de ficheros en Sass/Scss

1. En primer lugar, en el fichero **.html** mediante la etiqueta **<link>** se referencia la ruta de enlace al fichero **.css** de la siguiente forma:

<link rel="stylesheet" href="estilos.css">

2. Ahora bien, el fichero en el que se lleva a cabo la implementación de estilo presenta la extensión **.scss**, pero en el paso anterior el enlace se realiza hacia un fichero .css, por lo que será necesario para realizar la vinculación con el fichero HTML **generar un archivo CSS**.

3. Para ello se utiliza la instrucción "**sass --watch…**", la cuál tras ser ejecutada en el terminal por línea de comandos, permanece "escuchando" si se produce algún cambio en el fichero .scss y si es así genera un nuevo fichero .css, al que se apunta desde HTML.

sass --watch estilos**.scss**:estilos**.css**

Al lanzar esta instrucción se queda escuchando a que se produzca un cambio en el fichero .scss (en este caso estilos.scss) y si se modifica algo en el código y se guarda, se detecta el cambio y se almacena la nueva versión en el arhivo .css (estilos_.css).

2.4. ESTRUCTURA DE CARPETAS Y FICHEROS

Para trabajar desde Sass resulta aconsejable crear un sistema de carpetas y ficheros de estilo SCSS, que posea una estructura eficiente que permitirá optimizar la implementación del sitio.

Como se ha expuesto previamente, desde el fichero HTML se enlaza con la hoja de estilo CSS utilizando el elemento **link**, en concreto al fichero destino donde se está haciendo la traducción scss-css.

Recuerda:

Para enlazar los ficheros scss y css utiliza:

sass --watch estilos**.scss**:estilos**.css**

<link rel="stylesheet" href="**estilos.css**">

Desde **estilos.scss** se **importan el resto de ficheros de estilo necesarios**, ya que HTML solo enlazará con un fichero y este se encarga de importar los demás. El contenido de estilos.scss podría quedar de la siguiente forma:

```
@import "config";
@import "utilites";
@import "mixins";
```

Si los cambios se producen en cualquiera de los ficheros importados en estilos.scss, se detectará como un cambio y se actualizará el fichero .css, que se genera automáticamente cada vez que se produce una modificación y está se guarda.

2.5. VARIABLES EN SASS

Sass permite crear variables en las que almacenar un valor, por lo que por ejemplo, si se quiere modificar el color de fondo no sería necesario cambiarlo en todos los elementos utilizaban ese color, sino que bastaría con hacerlo en la variable creada para tal propósito, y que posteriormente será utilizada por el resto de reglas de estilo.

Para la creación de una variable se utiliza "**$**", seguida del nombre identificativo que se le va a asignar a la variable:

```
$variable: (valor);
selector (etiquetalidlclase) {
        propiedad: $variable;
}
```

Ejemplo: En el siguiente ejemplo, en el fichero estilos.scss se definen las variables $color-primario, $color-fondo, $fuente-principal y $tamanio-texto, que posteriormente son utilizados en el body.

```scss
// Definir variables
$color-primario: #3498db;
$color-fondo: #ecf0f1;
$fuente-principal: 'Arial', sans-serif;
$tamanio-texto: 16px;

// Estilos para el cuerpo de la página
body {
  font-family: $fuente-principal;
  font-size: $tamanio-texto;
  background-color: $color-fondo;
}
```

Figura 4. Código ejemplo variables

Al compilar el fichero se crea un nuevo archivo estilos.css que quedaría del a siguiente forma, sin incluir las variables, solo aparecerá el valor de cada elemento, lo que sí resulta comprensible para el html.

```css
body {
  font-family: "Arial", sans-serif;
  font-size: 16px;
  background-color: #ecf0f1;
}
```

Figura 5. Código ejemplo variables compilado

Actividad propuesta 6.1.

Crea un archivo Sass que contenga cuatro variables para la definición de ciertas propiedades clave en un sitio web: color principal, color de fondo, tipo de fuente y el tamaño de texto.

Recuerda compilar el archivo SCSS a CSS, realiza varias pruebas en las que modificando el valor de las variables se compila de forma automática el código y se muestra el nuevo resultado del sitio web de forma automática.

2.6. MIXIN EN SASS

Se denomina **mixin** a los bloques de código reutilizables (también llamadas componentes reutilizables), bajo los que se recogen propiedades de estilo que se podrían repetir en diferentes ocasiones a lo largo de todo el programa.

Para implementarlos se utiliza la etiqueta **@mixin** seguida del nombre del bloque que se quiera utilizar, pudiendo **recibir por parámetro de forma opcional parámetro** (si se van a utilizar varios parámetros estos se introducen separados por comas).

Esta funcionalidad permite crear un bloque de código reutilizable como si de una función se tratará, de esta forma al ser llamado desde diferentes elementos podrá recibir por parámetro los valores que se necesiten, personalizado así el resultado final.

```
@mixin nombreBloque (parámetro1, parámetro2, …) {
    propiedad_1: valor;
    …
}
```

De esta forma quedan definidos los bloques **mixin**, para incluirlos en el resto del código del fichero de estilo. Para que puedan ser llamados y reutilizados desde el fichero de estilos se utilizará la **@include**.

```
selector (etiqueta | id | clase){
    @include nombreBloque;
    resto de propiedades…
}
```

Por ejemplo, en el siguiente fragmento de código se crea un **mixin** con el nombre **sizes**, que recibe como parámetros dos valores, en este caso width y height. Este mixin es utilizado desde el body con valores de 100px para ancho y alto.

```
@mixin sizes($width, $height) {
  height: $height;
  width: $width;
}
.body {
  @include sizes(100px, 100px);
}
```

Actividad propuesta 6.2.

Realiza un componente reutilizable de estilo que incluya variables predefinidas, en este caso crea dos elementos de texto que aplique los conceptos de variables y mixins en Sass para diseñar un componente reutilizable.

- Define al menos variables para el color de fondo, color del texto, borde y tamaño del texto, y crea un mixin reutilizable que se usará al menos 2 veces, el mixin aceptará argumentos para personalizar los estilos del texto.
- Debes utilizar el mixin para crear al menos dos instancias de cuadros de texto con diferentes estilos, empleando las variables definidas para ajustar el aspecto de cada instancia.

2.7. OPERADORES EN SASS

Sass permite la inclusión de operadores en el código CSS. Este tipo de elementos permite la realización de operaciones como ajustar las dimensiones de un elemento en función de un conjunto de condiciones. Al realizar la conversión .scss a .css se realizarán los cálculos indicados en el fichero .css se mostrará el resultado final, que es el implementado en el sitio.

Aritméticos	Comparación	Lógicos
Suma (+)	**Igualdad (==)**. Compara si dos valores son iguales	**AND (&&)**: Devuelve true si ambas condiciones son verdaderas.
Resta (-)	**Desigualdad (!=)**. Compara si dos valores son diferentes	**OR (||):** Devuelve true si al menos una de las condiciones es verdadera.
Multiplicación (*)	**Mayor que (>).** Compara si un valor es mayor que otro	**NOT (!):** Devuelve true si la condición es falsa, y false si es verdadera.
División math.div(dividendo, divisor)	**Menor que (<)**	
Módulo (%)	**Mayor o igual que (>=)**	
	Menor o igual que (<=)	

Tabla 2. Clasificación operadores en Sass/SCSS

Además, es posible utilizar operadores que nos permiten realizar ciertas operaciones lógicas:

1. if: si la $condicion es verdadera, la variable $variable tomará el valor de $valor-true; de lo contrario, tomará el valor de $valor-false.

```
$variable: if($condicion, $valor-true, $valor-false);
```

2.@if, @else if, @else: se utilizan para realizar bloques de código condicional basado en varias condiciones.

```
@if $condicion {
  // si la condición es verdadera
} @else if $otra-condicion {
  // si la otra condición es verdadera
} @else {
  // si ninguna condición es verdadera
}
```

Práctica guiada: Crea dos botones y en función del tipo se le asigna un estilo.

En el fichero de estilo .scss, se utilizan @if y @else para comprobar el valor de la variable $activo que recibe por parámetro el mixin estilo-boton. Si es true (si el botón es de tipo activo) se le asignará el color de la variable $color-activo, mientras que si $activo toma el valor false (al tratarse de un botón inactivo), tomará el valor de la variable $color-inactivo.

```scss
$color-activo: #2ecc71;
$color-inactivo: #e74c3c;

@mixin estilo-boton($activo: true) {
  display: inline-block;
  padding: 10px 20px;
  color: #fff;
  border: 1px solid #fff;
  border-radius: 5px;
  cursor: pointer;

  @if $activo {
    background-color: $color-activo;
  } @else {
    background-color: $color-inactivo;
  }

  &:hover {
    background-color: darken(if($activo, $color-activo, $color-inactivo), 10%);
  }
}

// Uso del mixin para un botón activo
.boton-activo {
  @include estilo-boton(true);
}

// Uso del mixin para un botón inactivo
.boton-inactivo {
  @include estilo-boton(false);
}
```

Figura 6. Código práctica guiada operadores

Para probar el funcionamiento de lo anterior utiliza el siguiente código html:

```html
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <link rel="stylesheet" href="estilos.css">
  <title>Ejemplo con @if en Sass</title>
</head>
<body>

  <div class="contenedor">
    <h1>Botones con Sass</h1>

    <button class="boton-activo">Botón Activo</button>
    <button class="boton-inactivo">Botón Inactivo</button>

  </div>

</body>
</html>
```

Figura 7. Código práctica guiada operadores - index.html

Actividad propuesta 6.3.

Realiza un componente reutilizable de estilo que incluya variables predefinidas, en este caso crea dos elementos de texto que aplique los conceptos de variables y mixins en Sass para diseñar un componente reutilizable.

- Define al menos variables para el color de fondo, color del texto, borde y tamaño del texto, y crea un mixin reutilizable que se usará al menos 2 veces, el mixin aceptará argumentos para personalizar los estilos del texto.
- Debes utilizar el mixin para crear al menos dos instancias de cuadros de texto con diferentes estilos, empleando las variables definidas para ajustar el aspecto de cada instancia.

3.@for: itera sobre un rango de valores para la $variable.

```
@for $variable from <start> through <end> {
 // Código
}
```

En el siguiente fragmento se utiliza @for, que itera por el número total de elementos caja (recogido en la variable $numero-cajas). Para cada una de estas cajas se asigna un tamaño específico, en el caso del ancho se incrementará en cada iteración al ser multiplicado por el valor $i, al igual que el color de fondo.

```scss
@for $i from 1 through $numero-cajas {
  .caja-#{$i} {
    width: $ancho * $i;
    height: $ancho;
    background-color: lighten($color, 10% * $i);
    margin-right: 15px;
  }
}
```

Figura 8. Código ejemplo @for

Recuerda:

Las funciones **lighten** y **darken** son funciones incorporadas en Sass que permiten ajustar el brillo de un color, se suelen utilizar para obtener colores más claros (lighten) o más oscuros (darken) a partir de un color de base, y suelen ser especialmente útiles dentro de operadores que varían el estilo en función de otros aspectos.

Estas funciones reciben por parámetro el valor del color base y cuanto se quiere que varíe el colore resultante con respecto al tomado como base.

4. @each: itera sobre los elementos de la lista que se indica a continuación.

```
@each $variable in <list> {
    …
}
```

Para ilustrar el funcionamiento de @each crearemos un nuevo fichero html con varios elementos:

```
<div class="contenedor">
  <div class="elemento-rojo">Elemento Rojo</div>
  <div class="elemento-verde">Elemento Verde</div>
  <div class="elemento-azul">Elemento Azul</div>
  <div class="elemento-amarillo">Elemento Amarillo</div>
</div>
```

Utilizando @each se iterará por todos los elementos, extrayendo de cada elemento utilizando la función nth, el nombre (rojo, verde, azul y amarillo) y el color (en hexadecimal). Después se creará una clase dinámica que en función del nombre del elemento resultante, tomará un estilo u otro.

En este caso se utiliza la función lightness($color) que permite evaluar el brillo del color de fondo para ajustar el color del texto.

```
$elementos: "rojo" #e74c3c, "verde" #2ecc71, "azul" #3498db, "amarillo" #f1c40f;

@each $elemento in $elementos {
  $nombre: nth($elemento, 1);
  $color-fondo: nth($elemento, 2);

  .elemento-#{$nombre} {
    background-color: $color-fondo;
    color: if(lightness($color-fondo) > 50%, #000, #fff);
    padding: 10px;
    margin-bottom: 10px;
  }
}
```

Figura 9. Código ejemplo @each

Actividad propuesta 6.4.

Crea un sitio web con el código explicado sobre @each, pero añadiendo 3 cajas y 3 colores más.

5.@while: ejecuta un bucle mientras la condición sea verdadera.

```
@while $condicion {
  …
}
```

En este caso, utilizando @while, de nuevo se iterará por cada una de las cajas, asignando a cada de estas un tamaño diferente, usando como base el valor de la variable $ancho-base, incrementando su valor para cada una de las cajas.

```
$contador: 1;
$ancho-base: 50px;
$numero-cajas: 5;

.contenedor {
  @while $contador <= $numero-cajas {
    .caja-#{$contador} {
      width: $ancho-base * $contador;
      height: $ancho-base;
      background-color: #3498db;
      margin-bottom: 10px;
    }
    $contador: $contador + 1;
  }
}
```

Figura 10. Código ejemplo @while

Actividad propuesta 4.5.

Crea un sitio web con el código expuesto sobre @while, añadiendo 2 cajas más y utilizando un tono un color que varíe un 10% en cada iteración.

2.8. FUNCIONES ÚTILES EN SASS/SCSS

Como se han ido viendo a lo largo de este capítulo, Sass incluye varias funciones ya predefinidas que permiten agilizar el desarrollo de las hojas de estilo, añadiendo nuevas funcionalidades.

1. darken($color, $cambio): esta función es utilizada para reducir la luminosidad de un color, recibe por parámetro el color vale y le cantidad que se quiere alterar el color.

```
$color: #3498AA;
$color-cambio: 10%;
$color-modificado: darken($color, $cambio);
```

2. lighten($color, $cambio): esta función es utilizada para aumentar la luminosidad de un color, recibe por parámetro el color vale y le cantidad que se quiere alterar el color.

```
$color: #3498AA;
$color-cambio: 10%;
$color-modificado: lighten($color, $cambio);
```

3. mix($color1, $color2, $proporcion): esta función es utilizada para mezclar los colores que recibe por parámetro, en las proporciones indicadas.

```
$color1: #3498AA;
$color2: #20AB74;
$proporcion: 50%;
$color-mezcla: mix($color1, $color2, $proporcion);
```

4. complement($color): esta función es utilizada para devolver el color complementario ql que se recibe por parámetro.

```
$color: #3498AA;
$color-complementario: complement($color);
```

5. invert($color): esta función se utiliza para obtener el color invertido que recibe por parámetro.

```
$color: #3498AA;
$color-invertido: invert($color);
```

6. percentage($valor): esta función convierte a porcentaje el valor recibido por parámetro.

```
$valor: 0.75;
$porcentaje: percentage($valor);
```

7. grayscale($color): esta función se utiliza para realizar la conversión de un color a escala de grises.

```
$color: #3498AA;
$color-gris: grayscale($color);
```

6. transparentize($color, $cambio): esta función convierte un color recibido por parámetro en su versión más transparente, en base al valor indicado en el segundo parámetro recibido.

```
$color: #3498AA;
$cambio: 50%;
$color-transparente: transparentize($color, $cambio);
```

7. nth($lista, $n): esta función devuelve el n-ésimo elemento de una lista.

```
$lista: a b c d e;
$elemento: nth($lista, 3);
```

Existen muchas funciones predefinidas además de las expuestas en este apartado: opacify, change-color, adjust-hue, saturate, is-light, ceil, floor, abs, round, entre otros

Actividad propuesta 4.6.

Realiza un componente reutilizable de estilo que incluya variables predefinidas, en este caso crea dos elementos de texto que aplique los conceptos de variables y mixins en Less para diseñar un componente reutilizable.

- Define al menos variables para el color de fondo, color del texto, borde y tamaño del texto, y crea un mixin reutilizable que se usará al menos 2 veces, el mixin aceptará argumentos para personalizar los estilos del texto.
- Debes utilizar el mixin para crear al menos dos instancias de cuadros de texto con diferentes estilos, empleando las variables definidas para ajustar el aspecto de cada instancia.