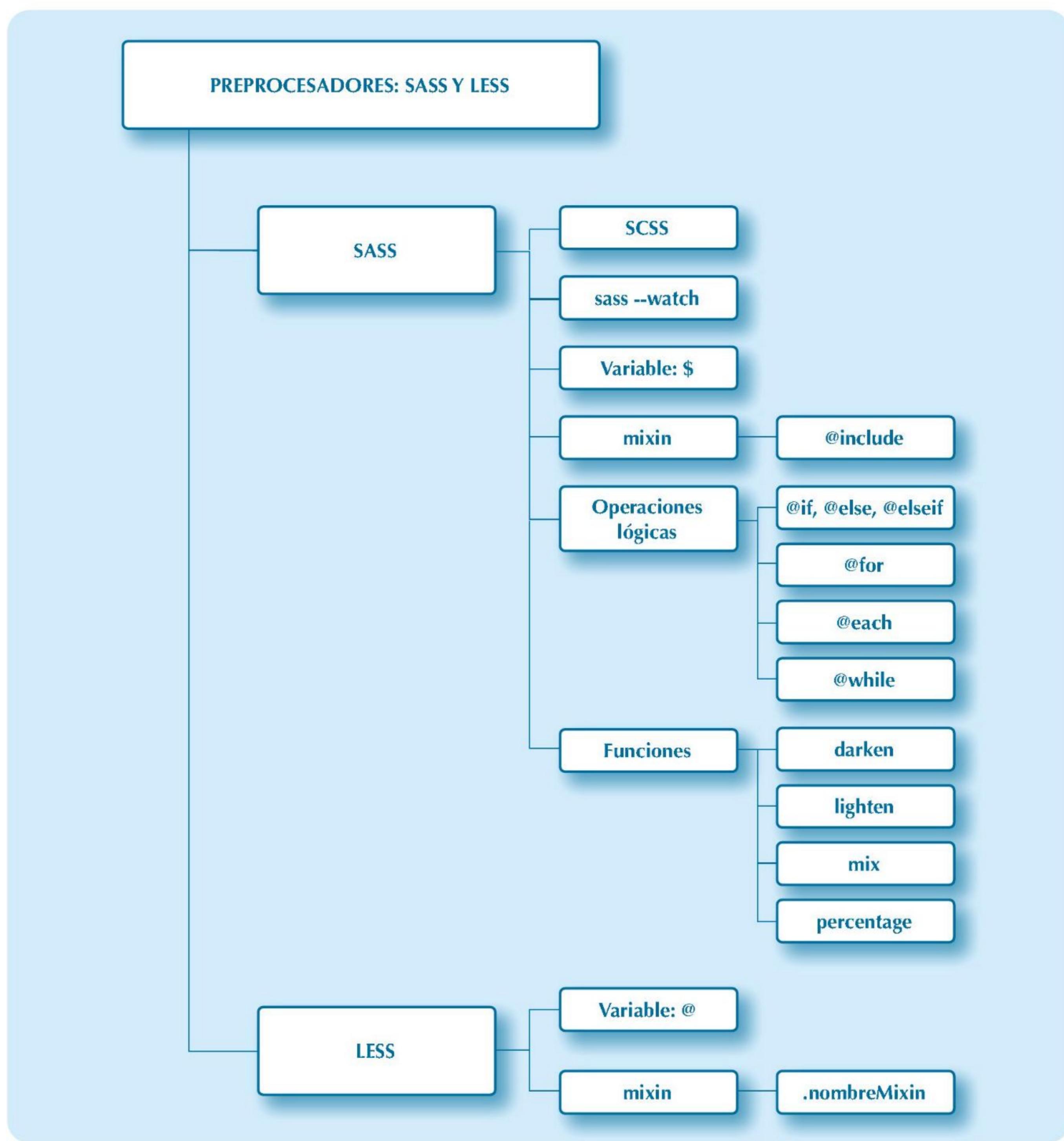


Preprocesadores Sass y Less

Objetivos

- ✓ Definir estilos de forma directa y asociar estilos globales en hojas externas.
- ✓ Redefinir estilos.
- ✓ Crear clases de estilos.
- ✓ Utilizar y actualizar guías de estilo.

Mapa conceptual



Glosario

Preprocesador. Herramienta que toma un conjunto de datos como entrada y realiza transformaciones en esos datos antes de pasarlos a la fase principal de procesamiento. En el caso de CSS, se realiza la conversión a fichero CSS interpretable por HTML.

Ruby. Lenguaje de programación dinámico, interpretado y de alto nivel, creado por Yukihiro, en la década de 1990, con el objetivo de combinar la simplicidad y la productividad de lenguajes como Perl y Python con la funcionalidad orientada a objetos de Smalltalk.

Sass. Preprocesador para CSS que permite la implementación de forma ágil para hojas de estilo que, tras un proceso de compilación, generan fichero CSS interpretable para HTML.

Less. Preprocesador CSS que incluye ciertos complementos para agilizar la implementación de las interfaces web, gracias al uso de una sintaxis propia. Estos ficheros son convertidos a lenguaje CSS, que HTML es capaz de comprender.

Mixin. Bloques de código reutilizables (también llamados componentes reutilizables) bajo los que se recogen propiedades de estilo que se podrían repetir en diferentes ocasiones a lo largo de todo el programa.

7.1. Introducción

El lenguaje de hojas de estilo CSS permite trabajar con recursos adicionales que agilizan la programación de estos 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 cada línea de código CSS e ir cambiando su valor en cada aparición. Los preprocesadores permitirán que esta tarea resulte más sencilla, gracias al uso de variables o bloques de código (*mixin*). En este capítulo, se trabajará con Sass y Less desde el proceso de instalación hasta la descripción y la puesta en práctica de algunos de sus elementos más característicos.

Si bien Sass y Less presentan un funcionamiento similar, también tienen ciertas diferencias, como se aprecia en el cuadro 7.1.

CUADRO 7.1
Comparativa entre Less y Sass

| LESS | SASS |
|--|--|
| Más similar a CSS | Menos similar a CSS |
| Funciona bajo JavaScript | Funciona bajo Ruby |
| Se requiere compilación <i>lessc</i> en cada modificación. Lado servidor | Actualización “dinámica” del fichero <i>.css</i> . Mejor compilación |

7.2. Sass

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

Sass es una herramienta multiplataforma 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 llaves. Lo habitual es utilizar la sintaxis SCSS en Sass.

WWW

Recurso web

Para obtener más información sobre Sass, puedes acceder al siguiente sitio web, escaneando el código QR.



7.2.1. 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 en el que se vaya a utilizar Sass. En función del sistema operativo, la descarga y la instalación de Ruby se realizan de las siguientes formas:

- *Windows*. Descargar el paquete de instalación desde el sitio web, ejecutarlo e instalarlo en el equipo. <https://rubyinstaller.org>
- *Mac*. Ruby ya está preinstalado en el sistema, aunque, si presenta una versión demasiado anticuada, es conveniente actualizarla a otra posterior, utilizando `rvm install` seguido de la versión que se quiera instalar.

```
rvm install ruby-version_actual
```

- *Linux*. Habitualmente aparece preinstalado 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, se introducirá la siguiente instrucción:

```
$ sudo gem install sass
```

RECUERDA

- ✓ `sudo` es un comando que se utiliza en sistemas operativos basados en Unix, como es el caso de Linux y macOS, para ejecutar otros comandos que requieren del uso de privilegios elevados. Para utilizarlo, basta con colocarlo en el inicio de la instrucción que se quiera ejecutar.

7.2.2. 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.

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

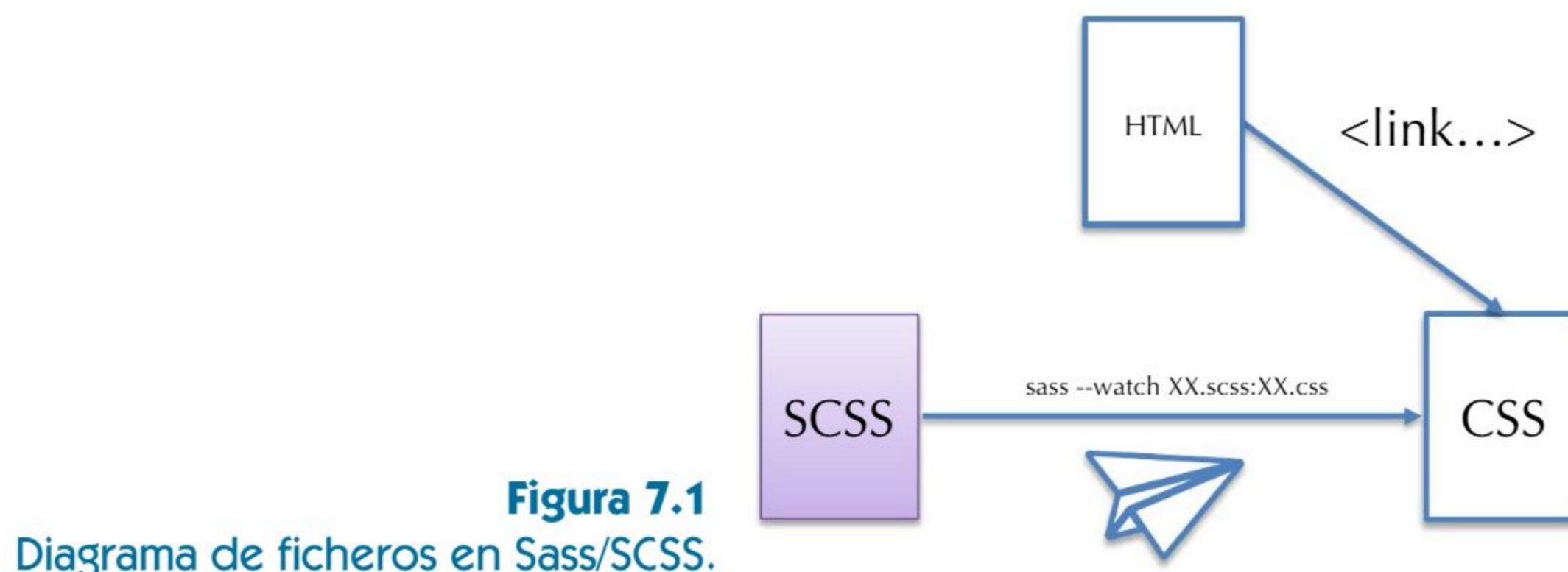
```
$color: #3498AA
body
    font-family: Arial
    color: $color
.contenedor
    width: 100%
    margin: 0 auto
```

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

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

7.2.3. 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 conversión a formato CSS para poder ser interpretado por un fichero en HTML. Una de las ventajas de este preprocesador es la actualización de manera automática del contenido del fichero CSS (figura 7.1).



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

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

- 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.
- 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 hasta que se produzca un cambio en el fichero *.scss* (en el ejemplo anterior, *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 archivo *.css* (*estilos_.css*).



TOMA NOTA

VENTAJAS Y FUNCIONAMIENTO DINÁMICO DE SASS

Los preprocesadores están cada vez más presentes. Uno de los más utilizados en la actualidad es Sass. Algunas de las ventajas son las siguientes:

- Sass resulta mejor para CSS3, sobre todo si usamos herramientas de desarrollo CSS3 como Compass o Bourbon.
- Permite minimizar la salida de ficheros CSS, mientras que otros, como Less, no lo permiten.
- Automatiza la creación de hojas de estilo.

Práctica guiada: primera prueba con Sass

En esta primera práctica, se va a comprobar el funcionamiento de Sass, es decir, si el fichero HTML se encuentra vinculado correctamente con el fichero de estilo CSS, y si este último se está creando convenientemente desde el SCSS.

En primer lugar, el fichero *index.html* se vincula con el fichero de estilo *estilos.css*, como se observa en la figura 7.2.

En el fichero *estilos.scss*, se realiza la implementa-

```
<!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 Sass con Variables</title>
</head>
<body>

  <div class="contenedor">
    <h1>Diseño de Interfaces Web</h1>
    <h1>Capítulo 4</h1>

    <p>Ejemplo de Sass con variables.</p>
    <a href="https://www.sintesis.com/" class="btn">Síntesis</a>

  </div>
</div>

</body>
</html>
```

Figura 7.2

Código de práctica guiada *index.html*.

ción de los estilos. A diferencia de en un fichero CSS, se pueden utilizar variables, *mixin*, operadores y otros elementos que se estudiarán en los siguientes apartados. Se muestra a continuación cómo quedará el fichero `estilos.scss`.

Desde el terminal se ejecutará `sass --watch estilos.scss:estilos.css`, por lo que, cada vez que se produzca un cambio en el fichero `.scss`, se compilará y se obtendrá un nuevo fichero `.css` comprensible para HTML (figura 7.3).

```
// Mixin para estilos de botón
@mixin boton($color-fondo, $color-borde) {
    display: inline-block;
    padding: 10px 20px;
    font-size: 16px;
    color: #fff;
    background-color: $color-fondo;
    border: 2px solid $color-borde;
    border-radius: 5px;
    text-decoration: none;
    transition: background-color 0.3s ease;

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

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

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

// Estilos para un contenedor
.contenedor {
    max-width: 800px;
    margin: 0 auto;
    padding: 20px;
    background-color: $color-fondo;
    border: 1px solid $color-primario;
}

// Aplicar mixin para un botón principal
.btn {
    @include boton($color-primario, darken($color-primario, 10%));
}
```

Figura 7.3
Código de práctica guiada `estilos.scss`.

7.2.4. Estructura de carpetas y ficheros

Para trabajar con Sass, resulta aconsejable crear un sistema de carpetas y ficheros de estilo SCSS. De esta forma se genera una estructura eficiente que permitirá optimizar el desarrollo del sitio web.

Como se ha expuesto previamente, el fichero HTML se enlaza con la hoja de estilo CSS utilizando el elemento `link`; en concreto, se enlaza con el fichero destino, donde se está realizando la traducción SCSS-CSS.

RECUERDA

- ✓ Para enlazar los ficheros SCSS y CSS utiliza:

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

- ✓ Para lanzar la escucha de cambios y traducción:

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

Desde `estilos.scss` se importan el resto de los ficheros de estilo necesarios, ya que HTML solo enlazará con un fichero, por lo que el fichero principal tendrá que importar los demás, en caso de realizar más de un fichero de estilo `.scss`.

Por ejemplo, si se crean un documento principal `estilos.scss` y otros secundarios (`config.scss`, `utilities.scss`, `mixins.scss`), desde `estilos.scss` se importarán el resto de los ficheros 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 lleve a cabo una modificación y esta se guarde.

7.2.5. 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 que utilicen ese color, sino que bastará con hacerlo en la variable creada para tal propósito y que posteriormente será utilizada por el resto de las reglas de estilo.

Para la creación de una variable, se usa `$` seguido del nombre identificativo que se le vaya a asignar a la variable:

```
$variable: (valor);
selector (etiqueta|id|clase) {
    propiedad: $variable;
}
```

En el siguiente ejemplo, en el fichero `estilos.scss` se definen las variables `$color-principal`, `$color-fondo`, `$fuente-principal` y `$tamanio-texto`, que posteriormente son utilizadas en el `body` (figura 7.4).

```
// Definir variables
$color-principal: #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 7.4
Código de ejemplo de variables.

Al compilar el fichero, se crea un nuevo archivo `estilos.css`, que quedaría de la siguiente forma. Sin incluir las variables, solo aparecerá el valor de cada elemento, lo que sí resulta comprensible para el fichero HTML (figura 7.5).

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

Figura 7.5
Código de ejemplo de variables compilado.



Actividad propuesta 7.1

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 tamaño de texto.

Recuerda compilar el archivo SCSS a CSS, y 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 inmediata.

7.2.6. *Mixin* en Sass

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

Para implementarlos, se utiliza la etiqueta `@mixin`, seguida del nombre del bloque que se quiera utilizar, pudiendo recibir de forma opcional parámetros (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 tratara. De esta forma, al ser llamado desde diferentes sitios, podrá recibir por parámetros, en cada caso, los valores que se necesiten, personalizado así el resultado final.

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

De esta manera, quedan definidos los bloques *mixin* para, posteriormente, incluirlos en el resto del código del fichero de estilo. Para utilizarlos se utilizará `@include` seguido del nombre del *mixin* creado.

```
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 100 px para ancho y alto.

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

Actividad propuesta 7.2



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

- Define, al menos, variables para el color de fondo, el color del texto, el borde y el tamaño del texto, y crea un *mixin* reutilizable que se usará, como mínimo, dos 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.

7.2.7. Operadores en Sass

Sass permite la inclusión de operadores en el código CSS. Este tipo de elementos hacen posible la realización de operaciones como ajustar las dimensiones de un elemento en función de un conjunto de condiciones.

CUADRO 7.2 Clasificación de operadores en Sass/SCSS

| Aritméticos | Comparación | Lógicos |
|-------------|---|---|
| Suma (+) | Igualdad (==). Compara si dos valores son iguales | AND (&&): devuelve <i>true</i> si ambas condiciones son verdaderas |
| Resta (-) | Desigualdad (!=). Compara si dos valores son diferentes | OR (): devuelve <i>true</i> si al menos una de las condiciones es verdadera |
| [.../...] | | |

CUADRO 7.2 (CONT.)

| Aritméticos | Comparación | Lógicos |
|---|--|--|
| Multiplicación (*) | Mayor que (>). Compara si un valor es mayor que otro | NOT (!): devuelve <i>true</i> si la condición es falsa, y <i>false</i> si es verdadera |
| División math. div(dividendo, divisor) | Menor que (<) | |
| Módulo (%) | Mayor o igual que (>=) Menor o igual que (<=) | |

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. Esta opción es utilizada de forma interna en la propiedad que evaluar.

```
$variable: if($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
}
```

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

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

En el siguiente código 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 (figura 7.6).

Figura 7.6
Código de ejemplo de *@for*.

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

RECUERDA

- ✓ Las funciones *lighten* y *darken* están incorporadas en Sass. Permiten ajustar el brillo de un color, y se suelen utilizar para obtener colores más claros (*lighten*) o más oscuros (*darken*) a partir de un color de base. Resultan especialmente útiles dentro de operadores que varían el estilo en función de otros aspectos.
- Estas funciones reciben como parámetros el valor del color base y cuánto se quiere que varíe el color resultante con respecto al básico.

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 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 (figura 7.7).

```
$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 7.7
Código de ejemplo de `@each`.



Actividad propuesta 7.3

Crea un sitio web con el código explicado sobre @each, pero añadiendo tres cajas y tres 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 estas tamaños diferentes, usando como base el valor de la variable `$ancho-base`, con el fin de incrementar su valor para cada caja (figura 7.8).

```
$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 7.8
Código de ejemplo de `@while`.



Actividad propuesta 7.4

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

6. `@warn`: devuelve un mensaje de advertencia durante la compilación.

```
@warn "Mensaje de advertencia";
```

Ejemplo

A continuación, se explica cómo crear dos botones y, en función del tipo, asignar un estilo. En el fichero de estilo `.scss`, se utilizan `@if` y `@else` para comprobar el valor de la variable `$activo`, que recibe por parámetro el *mixin* `estilo-botón`. Si es *true* (si el botón es de tipo activo), se le asignará el color de la variable `$color-activo`, mientras que, si `$activo` toma el valor *false* (al tratarse de un botón inactivo), tendrá el valor de la variable `$color-inactivo`.

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

@mixin estilo-botón($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-botón(true);
}

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

Para probar el funcionamiento de lo anterior, utiliza el siguiente código 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="botón-activo">Botón Activo</button>
    <button class="botón-inactivo">Botón Inactivo</button>

  </div>
</body>
</html>
```

Actividad propuesta 7.5



Implementa un sitio web como el analizado en la práctica guiada anterior, y añade dos nuevos botones de tipo inactivo. Incluye, además, cuatro cajas de texto, cada una asociada a un botón. Cada caja podrá ser verde (estado activo) o roja (estado inactivo). El estado de cada caja será común al botón asociado.

7.2.8. 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 parámetros el color vale y la cantidad que se quiera 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 parámetros el color vale y le cantidad que se quiera alterar el color.

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

3. *mix(\$color1, \$color2, \$proporción)*: esta función es utilizada para mezclar los colores que se reciban por parámetros en las proporciones indicadas.

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

4. *complement(\$color)*: esta función es utilizada para devolver el color complementario del que se reciba por parámetro.

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

5. *invert(\$color)*: esta función se utiliza para obtener el color invertido del que se reciba 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, con base en el 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.

```

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

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

7.3. Less

El preprocesador Less (*Leaner Style Sheets*) incluye ciertos complementos para agilizar la implementación de las interfaces web, gracias al uso de una sintaxis propia. Después, al igual que ocurre con el caso de Sass, estos ficheros son convertidos a lenguaje CSS, que HTML es capaz de comprender.

7.3.1. Instalación de Less

La instalación de Less puede realizarse desde el lado del cliente o del servidor, siendo más recomendada la segunda, puesto que la primera puede ocasionar una disminución en el rendimiento del sitio web si JavaScript se encuentra desactivado.

1. En primer lugar se realiza la descarga de Node.js desde su sitio web (<https://nodejs.org/en/download/>).
2. Desde el terminal o cmd, se introduce la instrucción siguiente (recuerda hacerlo como administrador):

```
npm install -g less
```

3. Ya estaría instalado. Ahora simplemente se lanza la compilación del fichero escrito en *.less* para ser convertido a *.css*.

```
lessc estilo.less > estilo.css
```

Desde un fichero en HTML, se incluiría la etiqueta *link* para vincular este documento con la hoja de estilo generada a partir del archivo *.less*. (figura 7.9).

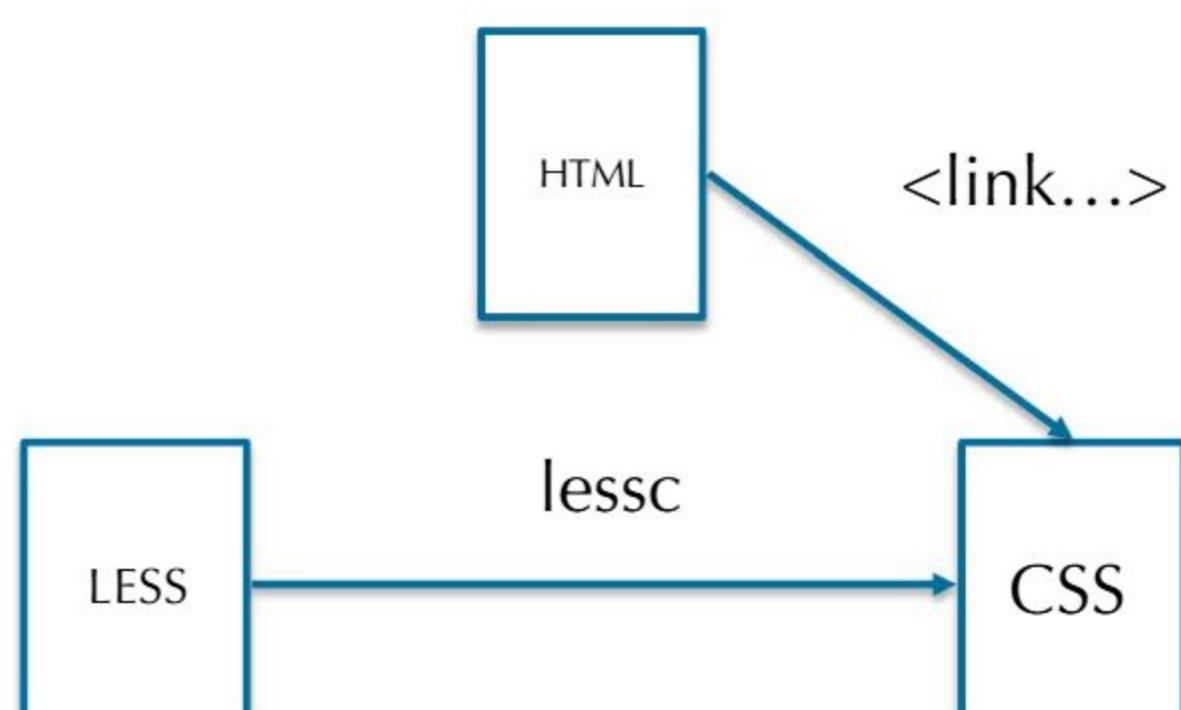


Figura 7.9
Diagrama de relación de Less.

7.3.2. Variables y operadores en Less

Al igual que en Sass/SCSS, Less permite la creación de variables para almacenar valores. En este caso, para la creación de variables se utiliza @:

```
@variable: (valor);
selector (etiqueta | id | clase) {
    propiedad: @variable;
}
```

Tras compilar el código anterior, se obtendrá un archivo .css, que queda de la forma que sigue:

```
selector (etiqueta | id | clase) {
    propiedad: valor;
}
```

Less permite también utilizar operadores en el código (+, -, *, /, %), que funcionan de la misma forma que la descrita en el apartado de Sass.

7.3.3. Mixin en Less

Los *mixin* nos permitían realizar agrupamiento de código. Luego se puede utilizar desde otro lugar el fichero .less. Para su implementación, se utiliza el símbolo “.” que precede al nombre asignado al bloque o *mixin*.

```
.nombreBloque{
    propiedad_1: valor;
...
    propiedad_N: valorN;
}
```

Al igual que ocurre con Sass, permite recibir por parámetro (entre paréntesis, a continuación del nombre) los valores que se quieran utilizar.

```
.nombreBloque(parámetro por defecto) {
    propiedad_1: valor;
...
    propiedad_N: valorN;
}
```

Resumen

- El uso de los preprocesadores puede agilizar en gran medida la implementación de hojas de estilo. Este tipo de programación resulta más eficiente, puesto que incorpora cierta pautas de diseño que hacen más intuitivo el desarrollo de hojas de estilo.
- Tanto Sass como Less incorporan funcionalidades bastante similares, tales como la posibilidad de utilizar operadores y la creación de variables o bloques de código a través de los elementos *mixin*. Si bien es cierto que cambian algunos aspectos en la sintaxis de su implementación, el resultado es prácticamente el mismo.
- Uno de los preprocesadores más comunes en la actualidad es Sass (*Syntactically Awesome Style Sheets*). Sass es un *gem* de Ruby, por lo que es necesario instalar Ruby en el equipo en el que se vaya a utilizar Sass.
- El desarrollo en Sass se basa en la implementación del código de estilo en un documento de tipo SCSS, el cual requiere 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.
- Desde el terminal se ejecutará `sass --watch estilos.scss:estilos.css`, por lo que, cada vez que se produzca un cambio en el fichero .scss, se compilará y se obtendrá un nuevo fichero .css comprensible para HTML.
- Sass permite crear variables en las que almacenar un valor. Para la creación de una variable se utiliza \$, seguida del nombre identificativo que se vaya a asignar a la variable:
- Se denomina *mixin* a los bloques de código reutilizables. Para implementarlos, se utiliza la etiqueta @mixin, seguida del nombre del bloque que se quiera usar, pudiendo recibir de forma opcional parámetro. Para que puedan ser llamados y reutilizados desde el fichero de estilos, se empleará @include.
- Sass permite la inclusión de operadores en el código CSS. Este tipo de elementos facilita la realización de operaciones como ajustar las dimensiones de un elemento en función de un conjunto de condiciones. Al realizar la conversión de .scss a .css, se harán los cálculos indicados en el fichero .css y se mostrará el resultado final, que es el implementado en el sitio. También se incorpora el uso de operadores y directivas (if, for, each...).
- El preprocesador Less (*Leaner Style Sheets*) incluye ciertos complementos para agilizar la implementación de las interfaces web, gracias al uso de una sintaxis propia. Después, al igual que ocurre con el caso de Sass, estos ficheros son convertidos a lenguaje CSS, que HTML es capaz de comprender.
- Al igual que Sass/SCSS, Less permite la creación de variables para almacenar valores. En este caso, para la creación de variables se utiliza @.
- Los *mixin* nos permiten realizar agrupamiento de código. Luego se puede utilizar desde otro lugar el fichero .less. Para su implementación se utiliza el símbolo “.” que precede al nombre asignado al bloque o *mixin*.

Ejercicios propuestos



1. Accede al sitio web oficial de Bootstrap y analiza todas las referencias a Sass que se incluyen. ¿Por qué Sass usa Bootstrap?
2. Crea un proyecto web básico que incluya una estructura de archivos HTML, CSS y Sass. En el archivo HTML, crea una página simple con, al menos, tres secciones. Utiliza Sass para definir estilos básicos, como colores, fuentes y márgenes para cada sección. Utiliza variables, anidamiento de selectores y la función de anidamiento para organizar tu código de manera eficiente. Compila tu archivo Sass en CSS, y vincúlalo a tu archivo HTML para ver los estilos aplicados en la página web resultante.
3. Tomando como punto de partida el sitio web sobre los Objetivos de Desarrollo Sostenible, en el que ya has trabajado, mejora su diseño y optimiza su desarrollo, implementándolo con Sass y Less como preprocesadores. Crea una página HTML básica con elementos estructurales y dos archivos de estilo separados: uno utilizando Sass y otro utilizando Less. Implementa las mismas reglas de estilo en ambos archivos, y observa las diferencias en la sintaxis y las características de cada preprocesador. Luego, compila los archivos para obtener CSS estándar y compara cómo se aplican al HTML.
4. Se propone el diseño que se muestra en la imagen, a través del uso de Sass/SCSS. Se solicita lo siguiente:
 - Crea tantas variables como necesites para que, en el caso de querer cambiar el color de la letra, modificar alguna dimensión, etc., solo tengamos que acudir a las variables comunes, buscar la deseada y modificarla.
 - Crea, al menos, dos bloques *mixin* que incluyan las propiedades de estilo comunes de la cabecera y la subcabecera, y del resto del texto.
 - Además, el tamaño del banner principal, el que aparece color azul, será siempre cinco veces mayor que el de color verde.

| Módulos por curso para el Ciclo Formativo de Grado Superior: Desarrollo de Aplicaciones Web | | |
|--|--|--|
| 1º CURSO Sistemas informáticos Bases de datos Programación Lenguaje de marcas y sistemas de gestión de la información Entornos de desarrollo | | |
| 2º CURSO 5h/semana Desarrollo web entorno cliente 6h/semana 5h/semana Desarrollo web entorno servidor 8h/semana 7h/semana Despliegue de aplicaciones web 5h/semana 4h/semana Diseño de interfaces web 6h/semana 3h/semana Empresa e iniciativa emprendedora 3h/semana | | |

Recuerda que, en primer lugar, en el fichero HTML se mantiene el elemento *link* apuntando al fichero .css de forma habitual. A continuación, se lanza desde la línea de comandos la instrucción `sass --watch estilos.scss:estilos.css`. Así, con cada cambio en el fichero `estilos.scss`, se producirá la traducción equivalente en el archivo `estilos.css`.

5. Se propone el diseño que se muestra en la imagen anterior, a través del uso de Less. En este ejercicio se pide implementar de nuevo el diseño expuesto en el ejercicio 1, utilizando ahora Less. De esta forma, se podrá realizar una comparativa en cuanto a su uso. Una de las ventajas del uso de Less es que, al lanzar el compilador, devuelve en el terminal los errores de sintaxis línea a línea.

- Crea tantas variables como necesites para que, en el caso de querer cambiar el color de la letra, modificar alguna dimensión, etc., solo tengas que acudir a las variables comunes, buscar la deseada y modificarla.
- Crea, al menos, dos bloques *mixin* que incluyan las propiedades de estilo comunes en la cabecera y la subcabecera, y en el resto del texto.
- Además, el tamaño del *banner* principal, el que aparece color azul, será siempre cinco veces mayor que el de color verde.

6. Implementa un sitio web en el que se incluya un sistema de estilos dinámico, utilizando @for con SCSS. Este sistema permitirá la creación de, al menos, cuatro cajas de color con propiedades variables. Cada una de las cajas tendrá un ancho y alto base de 100 px, y el color de fondo se aclarará gradualmente desde un color base.

Utiliza un *mixin* (componente de código reutilizable) para la definición de los estilos de cada una de las cajas caja, y usa un bucle @for para generar un número específico de estas cajas e ir modificando, en cada una de ellas, la propiedad solicitada.

7. Diseña un sitio web como el que se muestra en la imagen. En la parte inferior se mostrará un listado de tareas. Las tareas “Estudiar Sass” y “Estudiar examen” estarán completadas, mientras que “Hacer ejercicio” no lo estará. Si la tarea está completada, se mostrará de color verde; en caso contrario, no.

Para la implementación de este ejercicio, también se deben utilizar componentes reutilizables *mixin*.

Diseño de Interfaces Web

Capítulo 4

Ejemplo de Sass con variables.

Síntesis

8. Diseña un sitio web como el que se ve en la imagen. En la parte inferior, se mostrará un listado de tareas. Las tareas “Estudiar Sass” y “Estudiar examen” estarán completadas, mientras que “Hacer ejercicio” no lo estará. Si la tarea está completada, se mostrará de color verde; en caso contrario, no.

The screenshot shows a website layout with a dark header bar containing navigation links: Inicio (underlined), Acerca de, Servicios, Portafolio, and Contacto. Below the header are three main content boxes:

- ¡Bienvenido!**
Bienvenido a nuestro sitio web. Explora nuestras secciones para descubrir más sobre lo que ofrecemos.
- Acerca de Nosotros**
Somos una empresa dedicada a proporcionar soluciones creativas y efectivas para nuestros clientes.
- Nuestros Servicios**
Ofrecemos una amplia gama de servicios, desde diseño web hasta consultoría de negocios.

9. Diseña, utilizando Sass, un sitio web gestor de tareas. En cuanto a la funcionalidad, incluye secciones para agregar nuevas tareas, ver las listas de tareas pendientes y tareas completadas, y cambiar el estado de una tarea. Los elementos Sass mínimos que se deben utilizar son los siguientes:



- **Variables:** define variables tales como colores principales, tamaños de fuente y márgenes específicos para la interfaz de un gestor de tareas.
- **Mixins:** define, al menos, 10 *mixins* que apliquen estilos coherentes para elementos comunes, como botones, formularios y tarjetas de tareas.
- **Funciones:** utiliza las funciones incluidas en Sass (puedes usar otras que no aparezcan en listado). Emplea al menos ocho.
- **Operadores:** usa operadores para generar estilos para diferentes prioridades de tareas, ajustando colores y estilos automáticamente (los que consideres convenientes).
- Utiliza, al menos en 10 ocasiones, @if, @while, @each, @for...
- Realiza comentarios en tu código Sass para explicar el propósito de las variables, *mixins*, funciones y estructuras de control que has creado.

10. Modifica el código anterior para que el resultado sea el siguiente:



ACTIVIDADES DE AUTOEVALUACIÓN

1. ¿Cuál de las siguientes instrucciones se lanza desde la línea de comandos, actualiza el contenido del fichero .css cada vez que se modifica y guarda el contenido del archivo .scss?
 - a) sass --watch ficheroSCSS.scss:ficheroCSS.css
 - b) scss --watch ficheroSCSS.scss:ficheroCSS.css
 - c) <link rel="stylesheet" href="ficheroCSS.css">
 - d) Ninguna de las anteriores.
2. ¿Cuál de las siguientes opciones corresponde a la creación de un bloque *mixin* en Less?
 - a) @mixin1{text-size: 12px;}
 - b) .mixin1{text-size: 12px;}
 - c) #mixin1{text-size: 12px;}
 - d) \$mixin1{text-size: 12px;}
3. Para la creación de una variable en Sass, se utiliza:
 - a) &variable
 - b) \$variable
 - c) *variable
 - d) @variable
4. Para implementar un bloque de código reutilizable de tipo *mixin* en Sass, se debe indicar:
 - a) @mixin
 - b) %mixin
 - c) \$mixin
 - d) &mixin
5. Para la creación de una variable en Less, se usa:
 - a) &variable
 - b) \$variable
 - c) *variable
 - d) @variable
6. Indica cuál de las siguientes afirmaciones no es correcta:
 - a) La confección de ficheros .css en Sass no es automática.
 - b) La confección de ficheros .css en Less requiere la compilación a través de la línea de comandos.
 - c) La confección de ficheros .css en Less utiliza la instrucción lessc.
 - d) La confección de ficheros .css en Sass es automática a través del escuchador sass --watch.

7. Para implementar un bloque de código reutilizable de tipo *mixin* en Less, se debe indicar:

- a) @mixin
- b) %mixin
- c) .mixin
- d) &mixin

8. Las funcionalidades comunes que presentan Less y Sass son las siguientes:

- a) Posibilidad de utilizar operadores.
- b) Creación de variables.
- c) Uso de bloques de códigos reutilizables.
- d) Todas las anteriores son correctas.

9. ¿Cuál de las siguientes afirmaciones es correcta?

- a) Less trabaja bajo Ruby.
- b) Sass trabaja bajo Ruby.
- c) Sass trabaja bajo JavaScript.
- d) Less trabaja bajo C#.

10. ¿Cuál de las siguientes opciones corresponde a la creación de un bloque *mixin* en Sass?

- a) @mixin1{text-size: 12px; }
- b) .mixin1{text-size: 12px; }
- c) #mixin1{text-size: 12px; }
- d) \$mixin1{text-size: 12px; }

SOLUCIONES:

1. a b c d

2. a b c d

3. a b c d

4. a b c d

5. a b c d

6. a b c d

7. a b c d

8. a b c d

9. a b c d

10. a b c d