UD4. Sass
Continuación

Pedro Puerma

1. Estructuras de control

Estructuras típicas de lenguajes de programación que nos van a permitir desarrollar CSS de una manera óptima, más organizada y reusable.

1. Estructuras de control

@if: Permite **aplicar estilos condicionalmente** en función de una expresión booleana. Por ejemplo:

Estilo SCSS:

```scss
$color: blue;
.element {
    @if $color == blue {
      background-color: $color;
    } @else {
      background-color: red;
    }
}
```

Compila a:

```css
.element {
    background-color: blue;
  }
```

1. Estructuras de control

@else if/ @else:

Estilo SCSS:

```scss
$light-theme: true;
$dark-theme: false;
header {
 @if $light-theme == true {
 background-color: #fff;
 color: #000;
 } @else if $dark-theme {
 background-color: #000;
 color: #fff;
 } @else { //Default theme
 background-color: #aaa;
 color: #444;
 }
}
```

Compila a:

```scss
.element {
    background-color: blue;
  }
```

1. Estructuras de control

@for: permite crear **bucles for para generar reglas CSS repetitivas**.

Por ejemplo:

Estilo SCSS:

```scss
@for $i from 1 through 3 {
  .element-#{$i} {
    font-size: 10px * $i;
  }
}
```

Compila a:

```css
.element-1 {
  font-size: 10px;
}
.element-2 {
  font-size: 20px;
}
.element-3 {
  font-size: 30px;
}
```

1. Estructuras de control

@each: Utilizado para **iterar sobre listas o mapas y aplicar estilos** a cada elemento. Por ejemplo:

Estilo SCSS:

```scss
$colors: red, green, blue;
@each $color in $colors {
      .element-#{$color} {
        background-color: $color;
      }
}
```

Compila a:

```css
.element-red {
    background-color: red;
  }
  .element-green {
    background-color: green;
  }
  .element-blue {
    background-color: blue;
  }
```

1. Estructuras de control

@while: Permite crear **bucles while basados en una condición**. Por ejemplo:

Estilo SCSS:

```scss
$i: 1;
@while $i < 4 {
      .element-#{$i} {
        width: 100px * $i;
      }
      $i: $i + 1;
}
```

Compila a:

```css
.element-1 {
    width: 100px;
  }
  .element-2 {
    width: 200px;
  }
  .element-3 {
    width: 300px;
  }
```

2. Funciones

Sass proporciona una serie de funciones incorporadas que puedes utilizar para **realizar cálculos y manipulaciones de datos** en tus estilos.

Además, **podemos definir funciones** en las que, por un lado pondremos, normalmente un trozo de código que vayamos a utilizar frecuentemente y , por otro lado, nos deben devolver un valor.

2. Funciones

```
// Definir una función
@function anchura-col($col,$total) {
    @return percentage($col/$total);
}

//Llamadas a la función
.sidebar {
    ...
    width: anchura-col(2,10);
    ...
}
.main {
    ...
    width: anchura-col(5,10);
    ...
}
```

3. Mixins

Es una directiva de Sass que permite **definir estilos que luego puedo reutilizar** a lo largo del resto mi hoja de estilos. Son especialmente útiles cuando se quiere aplicar el mismo conjunto de estilos a múltiples elementos **sin tener que repetir el código**.

3. Mixins

Para definir un mixin en Sass, utiliza la palabra clave **@mixin**, seguida del nombre del mixin y, opcionalmente, los parámetros que aceptará el mixin. Ejemplo:

```
@mixin box-shadow($x, $y, $blur, $color) {
    box-shadow: $x $y $blur $color;
}
```

En este ejemplo, se define un mixin llamado «**box-shadow**» que acepta cuatro parámetros: **$x, $y, $blur, y $color**.

3. Mixins

Para utilizar un mixin en tu código Sass, necesitamos uti**lizar la palabra clave @include seguida del nombre del mixin** y los valores para los parámetros, si es necesario. Con el ejemplo anterior:

```
.element {
  @include box-shadow(2px, 2px, 4px, #888);
}
```

3. Mixins

Cuando compilemos el código Sass, esta regla se compilará en CSS con el conjunto de estilos especificado en el mixin:

```
.element {
  box-shadow: 2px 2px 4px #888;
}
```

4. Mixins con parámetros por defecto

Podemos **asignar valores por defecto** a los parámetros de un mixin **en caso de que no se proporcionen** valores cuando se llama al mixin. Por ejemplo:

```
@mixin box-shadow($x: 0, $y: 0, $blur: 4px, $color: #000) {
    box-shadow: $x $y $blur $color;
  }
```

Si no se proporcionan valores para los parámetros al llamar al mixin, los valores por defecto se utilizarán en su lugar. Por ejemplo:

```
.element {
  @include box-shadow;
}
```

4. Mixins con parámetros por defecto

Cuando compilemos el código Sass, esta regla se compilará en CSS con el conjunto de estilos especificado en el mixin:

```
.element {
box-shadow: 0 0 4px #000;
}
```

4. Mixins sin parámetros

Los mixins también **pueden definirse sin parámetros**, lo que los hace útiles para agrupar un conjunto de propiedades CSS relacionadas. Por ejemplo:

```
@mixin flexbox-center {
  display: flex;
  justify-content: center;
  align-items: center;
}
```

Luego, podemos incluir este mixin en cualquier regla que necesite centrar elementos utilizando **@include**:

```
.header {
  @include flexbox-center;
}
```

5. Directiva @import

Se utiliza para **importar otros archivos Sass en un archivo principal**:

- **Creación de archivos Sass:** En tu proyecto Sass, es posible **dividir tu código en varios archivos** según la lógica y la organización que necesites. Por ejemplo, puedes tener **un archivo para variables, otro para mixins, uno para estilos de encabezado, otro para estilos de botones, etc.**
- **Uso de @import:** En el archivo principal de Sass (generalmente llamado «styles.scss»), utilizamos la directiva **@import para importar otros archivos** Sass. Puedes importar archivos individuales o carpetas completas.

5. Directiva @import

La regla **@import espera como argumento el nombre del archivo** a importar. Por defecto busca un archivo Sass y lo importa directamente.

Ejemplos:

```
// Importa un archivo Sass individual (la extensión ".scss" es opcional)
@import '_colors';
// Importa un archivo Sass individual
@import 'scss/_colors.scss';
// Importa varios archivos
@import 'scss/_colors.scss', 'scss/_layout.scsss;
// Importa un archivo CSS individual
@import 'footer.css';
// Importa todos los archivos de una carpeta
@import 'scss/*';
```

5. Directiva @import

Cuando compilemos el archivo principal de Sass (por ejemplo, «styles.scss»), el **compilador Sass combinará todos los archivos importados en uno solo** y generará un archivo CSS resultante.:

5. Directiva @extend

Es una forma de **compartir un conjunto de propiedades y reglas CSS entre dos o más selectores**. En lugar de duplicar el código CSS idéntico en varios lugares, puedes definir un conjunto de propiedades en un selector y luego extender ese conjunto a otros selectores en los que quieras aplicar las mismas propiedades.

5. Directiva @extend

La sintaxis básica de @extend en Sass es la siguiente:

```
selector-a {
  propiedad: valor;
}
selector-b {
  @extend selector-a;
  /* Más reglas de estilo para selector-b */
}
```

5. Directiva @extend

A menudo, **@extend se utiliza para aplicar estilos comunes a diferentes elementos, como botones o elementos con estilos similares**.

```
.button {
    padding: 10px 20px;
    background-color: #3498db;
    color: #ffffff;
    text-decoration: none;
}
.primary-button {
    @extend .button;
    background-color: #e74c3c;
}
.secondary-button {
    @extend .button;
    background-color: #27ae60;
}
```