CSS por Componente y SASS

SASS (Syntactically Awesome Style Sheets - Hojas de Estilo Súper Geniales) es un preprocesador de CSS que extiende las capacidades del CSS tradicional, proporcionando características que permiten escribir estilos de manera más eficiente, organizada y modular. **SASS añade funciones avanzadas que no están disponibles en CSS puro**, como variables, anidación, mixins y herencia, lo que facilita la escritura y mantenimiento de estilos en proyectos de cualquier tamaño.

Características de SASS

Variables:

- **SASS permite el uso de variables para almacenar valores reutilizables como colores, tamaños de fuente o márgenes.**
- **Ejemplo:**

```
$primary-color: #4CAF50;
$font-size: 16px;

body {
  color: $primary-color;
  font-size: $font-size;
}
```

Anidación:

- **Permite anidar selectores dentro de otros, replicando la estructura de los elementos HTML, lo que hace que el código sea más legible y organizado.**
- **Ejemplo:**

```
.navbar {
  background-color: #333;

  ul {
    list-style: none;

    li {
      display: inline-block;
      padding: 10px;
    }
  }
}
```

Mixins:

- **Un mixin es un bloque reutilizable de código que se puede incluir en múltiples selectores.**
- **Permite definir estilos complejos de manera repetitiva sin duplicar código.**
- **Ejemplo:**

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

.container {
  @include flex-center;
  height: 100vh;
}

// viewport height" (altura del área visible del navegador)
```

Herencia

- **Se puede hacer que un selector herede los estilos de otro usando @extend.**
- **Esto es útil para reutilizar estilos comunes.**
- **Ejemplo:**

```
.button {
  padding: 10px 20px;
  background-color: blue;
```

```
}

.button-primary {
  @extend .button;
  background-color: green;
}
```

Operaciones:

- **SASS permite realizar cálculos matemáticos directamente en los estilos, como sumar, restar, multiplicar o dividir valores.**

- **Ejemplo:**

```
$width: 1000px;

.container {
  width: $width / 2; // 500px
}
```

Funciones:

- **SASS tiene funciones integradas para manipular colores, tamaños y otros valores, lo que permite realizar ajustes dinámicos en los estilos.**

- **Ejemplo:**

```
$primary-color: #3498db;

.box {
  background-color: lighten($primary-color, 20%);
}
```

Sintaxis de SASS:

SASS ofrece dos sintaxis diferentes:

1. **SASS (indented syntax): Una versión que omite llaves y puntos y comas, utilizando la indentación para definir bloques de código (más parecida a Python).**

2. **SCSS (Sassy CSS): Es la versión más popular, y es completamente compatible con el CSS tradicional. Usa llaves {} y punto y coma ;, lo que facilita la transición desde CSS.**

Ejemplo en SCSS:

```
$primary-color: #3498db;

body {
  font-family: Arial, sans-serif;

  h1 {
    color: $primary-color;
  }
}
```

Ejemplo en SASS (sintaxis indentada):

```
$primary-color: #3498db

body
  font-family: Arial, sans-serif

  h1
    color: $primary-color
```

CSS por Componente:

- En lugar de tener un solo archivo CSS grande que incluya estilos para toda la aplicación, dividimos los estilos en archivos más pequeños específicos para cada componente.
- Esto mejora la **modularidad**, haciendo que el código sea más fácil de mantener y entender, ya que cada archivo de estilos está directamente relacionado con su componente.
- En el ejemplo, cada componente (**Header**, **ProfileCard**, **Footer**) tiene su propio archivo **.scss** asociado, donde definimos los estilos solo para ese componente.

Uso de SASS:

- SASS permite escribir estilos de forma más eficiente gracias a características como **anidación** de selectores, **variables**, y **mixins**.
- En el archivo ProfileCard.scss, por ejemplo, se usó **anidación** para definir estilos dentro de .profile-card, lo que hace que el código sea más limpio y legible.

Características de SASS:

- **Mantenibilidad**: Los cambios en los estilos de un componente afectan solo a ese componente y no a toda la aplicación.
- **Escalabilidad**: Facilita el trabajo en proyectos grandes donde hay muchos componentes, ya que los estilos están aislados.
- **Modularidad**: Cada componente es independiente, lo que permite reutilizar y compartir componentes con facilidad.