Curso Completo de SASS

De Básico a Avanzado

¿Qué aprenderemos en este curso?

1. **Fundamentos básicos de SASS**: Variables, Anidación, Mixins, etc.
2. Funciones avanzadas como **Modularidad** y **Control de Flujos**.
3. **Casos prácticos** de implementación.
4. Ejercicios para aplicar lo aprendido.

Configuración de equipos:

1. Prerrequisitos

Antes de empezar, asegúrate de tener instalado:

- **Node.js** y **npm**: Puedes descargarlos desde Node.js.
- Un editor de texto o IDE (como **Visual Studio Code**).

Configuración de equipos:

2. Configurar el entorno

Paso 1: Crea una carpeta de proyecto

1. Abre la terminal.
2. Crea y entra en una carpeta para tu proyecto:

```
mkdir mi-proyecto-sass
cd mi-proyecto-sass
```

Paso 2: Inicializa el proyecto

1. Inicializa un proyecto con npm:

```
npm init -y
```

Esto generará un archivo `package.json`.

Configuración de equipos:

2. Configurar el entorno

Paso 3: Instala SASS

1. Usa npm para instalar **SASS** como una dependencia de desarrollo:

```
npm install sass --save-dev
```

2. Verifica la instalación ejecutando:

```
npx sass --version
```

Paso 4: Crea la estructura de carpetas

Organiza tus archivos en una estructura como esta:

```
mi-proyecto-sass/
|
├── scss/              # Archivos fuente SASS
|   └── style.scss     # Archivo principal SASS
|
├── css/               # Archivos CSS generados
|   └── style.css      # CSS compilado (se genera automáticamente)
|
└── index.html         # Archivo HTML principal
```

Configuración de equipos:

2. Configurar el entorno

Paso 5: Configura SASS para compilar

1. Abre el archivo `package.json`.
2. Añade un script para compilar automáticamente tus archivos SASS a CSS:
   json

```
"scripts": {
  "sass": "sass --watch scss:css"
}
```

Esto hará que SASS observe cambios en la carpeta `scss/` y genere el CSS correspondiente en `css/`.

Paso 6: Escribe tu código SASS

1. Crea el archivo `scss/style.scss` y añade algún código básico:

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

body {
  background-color: $primary-color;
  font-family: Arial, sans-serif;
}
```

Abre tu archivo `index.html` y enlaza el CSS generado:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Proyecto con SASS</title>
    <link rel="stylesheet" href="css/style.css">
</head>
<body>
    <h1>Hola, mundo!</h1>
</body>
</html>
```

Configuración de equipos:

2. Configurar el entorno

Paso 7: Compila y observa cambios

1. En la terminal, ejecuta el script que configuraste en `package.json`:
   bash

```
npm run sass
```

Ahora, cada vez que guardes cambios en `scss/style.scss`, SASS generará automáticamente el archivo `css/style.css`.

Paso 8: Prueba el proyecto

1. Abre `index.html` en tu navegador.
2. Verifica que los estilos SASS se apliquen correctamente.

Configuración de equipos:

2. Configurar el entorno

Configuración opcional: Automatización avanzada

Para mejorar tu flujo de trabajo, puedes usar herramientas adicionales como:

1. **Browsersync**: Refresca automáticamente el navegador al guardar cambios.

```bash
npm install browser-sync --save-dev
```

Luego, añade un script para iniciar un servidor:

```json
"scripts": {
  "start": "browser-sync start --server --files 'css/*.css, *.html'"
}
```

Ejecuta: *npm run start*

1. Variables y Mapas

Variables en SASS

Las variables en SASS te permiten almacenar valores reutilizables, como colores, tamaños, fuentes, etc. Esto facilita el mantenimiento del código, especialmente en proyectos grandes.

Sintaxis básica:

```
$nombre-variable: valor;

selector {
 propiedad: $nombre-variable;
}
```

Ejemplo básico:

```
$primary-color: #3498db;
$font-size-large: 18px;

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

CSS:

```
body {
    color: #3498db;
    font-size: 18px;
 }
```

1. Variables y Mapas

Ventajas de las variables

1. **Centralización de valores**: Cambiar el valor de una variable afecta a todos los selectores que la usan.
2. **Facilidad de mantenimiento**: Modificar estilos globales es rápido y sencillo.
3. **Reutilización**: Puedes usar la misma variable en múltiples lugares.

1. Variables y Mapas

Operaciones con variables

SASS permite realizar operaciones matemáticas con variables, lo que añade flexibilidad.

SCSS:

```
$base-font-size: 16px;
$line-height: $base-font-size *
1.5;

p {
 font-size: $base-font-size;
 line-height: $line-height;
}
```

CSS:

```
p {
   font-size: 16px;
   line-height: 24px;
 }
```

1. Variables y Mapas

Ejemplo avanzado con variables

Definir variables para un tema de diseño:

SCSS:

```scss
// Variables de colores
$primary: #3498db;
$secondary: #2ecc71;
$error: #e74c3c;

// Variables de tipografía
$font-family: 'Arial', sans-serif;
$base-font-size: 16px;

// Variables de espaciado
$padding: 10px;
$margin: 20px;

button {
 background-color: $primary;
 color: white;
 padding: $padding;
 font-family: $font-family;
}
```

1. Variables y Mapas

Mapas en SASS

Los mapas son una colección de pares clave-valor. Son ideales para agrupar variables relacionadas, como paletas de colores o puntos de ruptura.

Para acceder a un valor, usa la función `map-get($mapa, clave)`.

SCSS:

```
$nombre-mapa: (
 clave1: valor1,
 clave2: valor2,
 clave3: valor3
);
```

Ejemplo básico:

SCSS:

```
$theme-colors: (
 primary: #3498db,
 secondary: #2ecc71,
 danger: #e74c3c
);

button {
 background-color:
map-get($theme-colors, primary);
}

.alert {
 background-color:
map-get($theme-colors, danger);
}
```

CSS:

```
button {
   background-color: #3498db;
 }
  .alert {
   background-color: #e74c3c;
 }
```

1. Variables y Mapas

Iterar sobre mapas

SASS permite recorrer mapas con `@each`, lo que es útil para generar estilos dinámicos.

SCSS:

```
$theme-colors: (
 primary: #3498db,
 secondary: #2ecc71,
 danger: #e74c3c
);

@each $name, $color in
$theme-colors {
 .text-#{$name} {
   color: $color;
 }
}
```

CSS:

```
.text-primary {
   color: #3498db;
 }
.text-secondary {
   color: #2ecc71;
 }
.text-danger {
   color: #e74c3c;
 }
```

1. Variables y Mapas

Ejemplo práctico con mapas

Definir puntos de ruptura para diseño responsivo:

SCSS:

```scss
$breakpoints: (
 small: 480px,
 medium: 768px,
 large: 1024px
);
@mixin respond-to($size) {
 @media (max-width: map-get($breakpoints, $size)) {
   @content;
 }
}
.container {
 width: 100%;
 @include respond-to(small) {
   width: 90%;
 }
 @include respond-to(medium) {
   width: 80%;
 }
}
```

CSS:

```css
.container {
   width: 100%;
 }
  @media (max-width: 480px) {
   .container {
     width: 90%;
   }
 }
  @media (max-width: 768px) {
   .container {
     width: 80%;
   }
 }
```

1. Variables y Mapas

Funciones útiles para mapas

1. **`map-get($mapa, clave)`**: Obtiene el valor de una clave específica.
2. **`map-keys($mapa)`**: Retorna todas las claves del mapa.
3. **`map-values($mapa)`**: Retorna todos los valores del mapa.
4. **`map-merge($mapa1, $mapa2)`**: Combina dos mapas.

SCSS:

```
$theme-colors: (
 primary: #3498db,
 secondary: #2ecc71
);

$new-colors: (
 danger: #e74c3c
);

$merged-colors:
map-merge($theme-colors, $new-colors);

button {
 background-color:
map-get($merged-colors, danger);
}
```

CSS:

```
button {
   background-color: #e74c3c;
 }
```

1. Variables y Mapas

Conclusión

- Las **variables** centralizan valores globales, mejorando la consistencia y el mantenimiento del código.
- Los **mapas** organizan colecciones de valores relacionados, lo que los hace ideales para temas y configuraciones avanzadas.
- SASS permite combinar variables y mapas con mixins, iteraciones, y funciones para crear estilos altamente dinámicos y modulares.

Ejemplo Práctico

Uso en selectores:

```
body {
   background-color: $primary-color;
   padding: $padding;
 }
```

Usando Mapas:

```
button {
   color: map-get($theme-colors, 'primary');
 }
```

2. Anidación

Organiza estilos reflejando la jerarquía HTML.

Ejemplo:

```
nav {
    ul {
      list-style: none;
      li {
        display: inline;
      }
    }
 }
```

Ejemplo Avanzado

Usando el selector `&` para referenciar al padre:

```
a {
    text-decoration: none;
    &:hover {
      text-decoration: underline;
    }
    &:active {
      color: red;
    }
 }
```

Ejemplo Avanzado

Ejemplo básico: Pseudo-clases

Puedes usar & para añadir pseudo-clases como :hover, :focus, etc., al selector padre.

Código SASS:

```
button {
   background-color : #3498db;
   color: white;
   border: none;
    &:hover {
     background-color : darken(#3498db,
10%);
   }
    &:active {
     background-color : lighten(#3498db,
10%);
   }
 }
```

CSS generado:

```
button {
   background-color: #3498db;
   color: white;
   border: none;
 }
 button:hover {
   background-color: #2b7bbd;
 }
 button:active {
   background-color: #5daeff;
 }
```

Ejemplo Avanzado

Ejemplo avanzado: Anidación con combinadores

El selector & se puede usar junto con combinadores (>, +, ~) para crear reglas específicas.

Código SASS:

```
button {
   background-color: #3498db;
   color: white;
   border: none;
 }
 button:hover {
   background-color: #2b7bbd;
 }
 button:active {
   background-color: #5daeff;
 }
```

Código CSS generado:

```
.container > .child {
   padding: 10px;
 }
 .container + .sibling {
   margin-top: 20px;
 }
 .container ~ .nearby {
   border: 1px solid #ccc;
 }
```

Ejemplo Avanzado

Ejemplo avanzado: Anidación con combinadores

Cuando trabajas con clases que tienen modificadores (como el estilo de BEM), & puede ser muy útil.

Código SASS:

```
.button {
   padding: 10px 20px;
   background-color: #3498db;
    &--primary {
     background-color: #2ecc71;
   }
    &--secondary {
     background-color: #e74c3c;
   }
 }
```

Código CSS generado:

```
.button {
   padding: 10px 20px;
   background-color: #3498db;
 }
 .button--primary {
   background-color: #2ecc71;
 }
 .button--secondary {
   background-color: #e74c3c;
 }
```

Ejemplo Avanzado

Anidación con selectores específicos

Puedes usar & para construir selectores que dependan del contexto del selector padre.

Código SASS:

```
.card {
   background-color: white;
   padding: 20px;
    &.is-active {
     border: 2px solid #3498db;
   }
    & .card-header {
     font-weight: bold;
     color: #333;
   }
 }
```

Código CSS generado:

```
.card {
   background-color: white;
   padding: 20px;
 }
 .card.is-active {
   border: 2px solid #3498db;
 }
 .card .card-header {
   font-weight: bold;
   color: #333;
 }
```

Ejercicio 12.1

3. Mixins y Argumentos

Los Mixins encapsulan bloques reutilizables de código.

Ejemplo básico:

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

Ejemplo con Argumentos:

```
@mixin border-radius($radius: 5px) {
   border-radius: $radius;
 }
```

3. Mixins y Argumentos

Un **mixin** es un bloque de código reutilizable que puedes incluir en varios selectores. Son especialmente útiles para estilos que se repiten frecuentemente, como propiedades relacionadas con prefijos específicos del navegador, configuraciones de diseño o componentes de UI.

Sintaxis básica

```
@mixin nombre-del-mixin {
   // Propiedades CSS
 }
  selector {
   @include
nombre-del-mixin;
 }
```

Ejemplo Básico:

Código SASS:

```
@mixin flex-center {
   display: flex;
   justify-content:
center;
   align-items: center;
 }
  .container {
   @include
flex-center;
   height: 100vh;
 }
```

Código CSS generado:

```
 .container {
   display: flex;
   justify-content:
center;
   align-items: center;
   height: 100vh;
 }
```

3. Mixins y Argumentos

Mixins con argumentos

Los mixins pueden aceptar argumentos para ser más dinámicos. Esto permite personalizar los estilos según las necesidades.

Sintaxis básica

```
@mixin
nombre-del-mixin($parametro
1, $parametro2:
valor-predeterminado) {
   // Propiedades CSS que
usan parámetros
 }
  selector {
   @include
nombre-del-mixin(valor1,
valor2);
 }
```

Ejemplo práctico:

Código SASS:

```
@mixin border-radius($radius:
5px) {
   border-radius: $radius;
   -webkit-border-radius:
$radius; // Compatibilidad
   -moz-border-radius: $radius;
 }
  .card {
   @include border-radius(10px);
 }
  .button {
   @include border-radius(); //
Usa el valor predeterminado
 }
```

Código CSS generado:

```
.card {
   border-radius: 10px;
   -webkit-border-radius: 10px;
   -moz-border-radius: 10px;
 }
  .button {
   border-radius: 5px;
   -webkit-border-radius: 5px;
   -moz-border-radius: 5px;
 }
```

3. Mixins y Argumentos

Mixins con multiples parámetros

Un mixin puede aceptar varios parámetros y personalizar estilos de forma más detallada.

Ejemplo práctico:

Código SASS:

```
@mixin box-shadow($color: rgba(0, 0, 0, 0.2), $blur: 5px) {
   box-shadow: 0 2px $blur $color;
 }
  .card {
   @include box-shadow(rgba(0, 0, 0, 0.5), 10px);
 }
  .button {
   @include box-shadow(); // Usa valores predeterminados
 }
```

Código CSS generado:

```
.card {
   box-shadow: 0 2px 10px rgba(0, 0, 0, 0.5);
 }
  .button {
   box-shadow: 0 2px 5px rgba(0, 0, 0, 0.2);
 }
```

3. Mixins y Argumentos

Mixins con contenido dinámico

Puedes usar `@content` dentro de un mixin para incluir bloques personalizados de estilos en los lugares donde se llame.

Ejemplo práctico:

Código SASS:

```
@mixin media-query($breakpoint) {
   @media (max-width:
$breakpoint) {
     @content; // Inserta el
contenido dinámico
   }
 }
  .container {
   background-color: blue;
    @include media-query(768px) {
     background-color: red;
   }
 }
```

Código CSS generado:

```
.container {
   background-color: blue;
 }
  @media (max-width: 768px) {
   .container {
     background-color: red;
   }
 }
```

3. Mixins y Argumentos

Mixins con contenido dinámico

Puedes usar `@content` dentro de un mixin para incluir bloques personalizados de estilos en los lugares donde se llame.

Beneficios de usar Mixins

1. **Reutilización**: Escribe una vez, úsalo en varios lugares.
2. **Legibilidad**: Elimina redundancias y hace que los estilos sean más fáciles de entender.
3. **Mantenimiento**: Cambiar un mixin automáticamente actualiza todos los selectores donde se usa.
4. **Compatibilidad**: Agrega fácilmente propiedades con prefijos.

Resumen del apartado

Los mixins, combinados con argumentos y contenido dinámico, permiten crear un sistema de estilos modular y adaptable. Son esenciales para proyectos grandes y ayudan a mejorar tanto la legibilidad como la escalabilidad de los estilos.

4. Herencia y Extensiones

Permite reutilizar estilos con `@extend`.

Ejemplo:

```
%button-style {
    padding: 10px;
    border: none;
 }

.btn-primary {
    @extend %button-style;
    background-color: blue;
 }
```

4. Herencia y Extensiones

Herencia en SASS: `@extend`

La **herencia** permite que un selector comparta los estilos de otro. Esto se logra con la directiva `@extend`. Cuando un selector usa `@extend`, toma todas las reglas del selector extendido y las aplica a sí mismo.

Cómo funciona

1. **Selector base**: Define estilos comunes que pueden ser compartidos.
2. **Selector extendido**: Usa `@extend` para heredar esos estilos.

```
SCSS
.button {
   padding: 10px;
   border: none;
   border-radius: 5px;
 }
  .primary-button {
   @extend .button;
   background-color: #3498db;
   color: white;
 }
```

```
CSS
.button, .primary-button {
   padding: 10px;
   border: none;
   border-radius: 5px;
 }
  .primary-button {
   background-color: #3498db;
   color: white;
 }
```

4. Herencia y Extensiones

Herencia en SASS: `@extend`

La **herencia** permite que un selector comparta los estilos de otro. Esto se logra con la directiva `@extend`. Cuando un selector usa `@extend`, toma todas las reglas del selector extendido y las aplica a sí mismo.

Cómo funciona

1. **Selector base**: Define estilos comunes que pueden ser compartidos.
2. **Selector extendido**: Usa `@extend` para heredar esos estilos.

Ventajas de `@extend`

1. **Evita duplicación de código**: Los estilos comunes se definen una sola vez.
2. **Crea CSS más eficiente**: Combina selectores en lugar de repetir las propiedades.
3. **Facilita el mantenimiento**: Cambiar los estilos base actualiza automáticamente todos los selectores que los extienden.

4. Herencia y Extensiones

Herencia en SASS: `@extend`

Placeholder Selectors (%)

Un **placeholder** es un selector que solo existe para ser extendido y no se traduce directamente en CSS. Esto ayuda a evitar que selectores innecesarios aparezcan en el archivo CSS final.

SCSS

```scss
%button-style {
   padding: 10px;
   border-radius: 5px;
   border: none;
 }
  .primary-button {
   @extend %button-style;
   background-color: #3498db;
   color: white;
 }
  .secondary-button {
   @extend %button-style;
   background-color: #2ecc71;
   color: white;
 }
```

CSS

```css
.primary-button, .secondary-button
{
   padding: 10px;
   border-radius: 5px;
   border: none;
 }
  .primary-button {
   background-color: #3498db;
   color: white;
 }
  .secondary-button {
   background-color: #2ecc71;
   color: white;
 }
```

4. Herencia y Extensiones

Herencia en SASS: `@extend`

Ventajas de los Placeholders

1. **Evita selectores vacíos**: Si usas `.button` en lugar de `%button-style`, el CSS generado incluiría una clase `.button` aunque no se use en el HTML.
2. **Optimización**: Solo genera CSS para los selectores extendidos.

4. Herencia y Extensiones

Ejemplo avanzado: Herencia combinada

Puedes combinar herencia con otros selectores y propiedades específicas para crear sistemas modulares.

```
SCSS
%card {
   padding: 20px;
   border-radius: 10px;
   box-shadow: 0 2px 5px rgba(0, 0,
0, 0.1);
 }
  .product-card {
   @extend %card;
   background-color: white;
 }
  .profile-card {
   @extend %card;
   background-color: #f9f9f9;
   text-align: center;
 }
```

```
SCSS
.product-card, .profile-card {
   padding: 20px;
   border-radius: 10px;
   box-shadow: 0 2px 5px rgba(0, 0,
0, 0.1);
 }
  .product-card {
   background-color: white;
 }
  .profile-card {
   background-color: #f9f9f9;
   text-align: center;
 }
```

4. Herencia y Extensiones

Limitaciones de `@extend`

1. **CSS más complejo**: Puede generar combinaciones de selectores largas si no se usa correctamente.
2. **No combina selectores específicos**: Si extiendes un selector dentro de una regla anidada, no se aplicará automáticamente a otras anidaciones.

SCSS

```scss
.container .box {
   border: 1px solid black;
 }
  .special-box {
   @extend .box;
 }
```

CSS

```css
.container .box {
   border: 1px solid black;
 }
  .special-box {
   @extend .box;
 }
```

La clase `.special-box` no será aplicable fuera de `.container`.

4. Herencia y Extensiones

Cuándo usar `@extend`

1. **Estilos comunes**: Úsalo para compartir estilos básicos, como botones, tarjetas o encabezados.
2. **Optimización del código**: Reduce la duplicación en proyectos grandes.
3. **Placeholders (`%`)**: Prefiérelos para evitar clases no utilizadas.

Conclusión

La **herencia** y los **placeholders** son herramientas clave en SASS para crear estilos modulares y organizados. Usar estas herramientas correctamente te ayuda a mantener tu CSS limpio y fácil de mantener, evitando redundancias y mejorando la escalabilidad.

Ejercicio 12.3