



IT Academy
by KIBERNUM



Sass: Organización y Modularización de Estilos

Propósito de la clase

En esta sesión, exploraremos metodologías avanzadas para la organización y modularización de estilos en SASS. Aprenderás a estructurar y escribir código SASS de manera eficiente, escalable y mantenible, aprovechando técnicas como anidación, parciales, mixins, extends y el uso de variables. Al finalizar esta clase, serás capaz de:

- Comprender los conceptos fundamentales de SASS, incluyendo variables, anidación, mixins, extends y parciales.
- Diferenciar el uso de @extend, @mixin y @use, identificando cuándo y cómo aplicar cada uno en un proyecto real.
- Implementar un sistema de estilos escalable mediante SASS, organizando el código en parciales y estructurando los estilos de manera modular.
- Optimizar el código CSS reduciendo la redundancia mediante @extend, @mixin y @use, mejorando la mantenibilidad y la escalabilidad del proyecto.

Uso de Variables en Sass





Aprendiendo Variables en SASS

💡 ¿Qué es una variable en SASS?

Una variable en SASS es un valor reutilizable que te permite escribir código CSS más limpio y fácil de mantener. En lugar de repetir colores, fuentes o tamaños en todo el código, podemos guardarlos en una variable y usarlos cuando los necesitemos.

📌 Símbolo de las variables en SASS:

Todas las variables en SASS comienzan con el símbolo \$.

```
$brand-primary: darken(#428bca, 6.5%) !default; // #337ab7
$brand-success: #5cb85c !default;
$brand-info: #5bc0de !default;
$brand-warning: #f0ad4e !default;
$brand-danger: #d9534f !default;

$font-family-sans-serif: "Helvetica Neue", Helvetica, Arial, sans-serif !default;
$font-family-serif: Georgia, "Times New Roman", Times, serif !default;
```

Ejemplo Básico de Variables en SASS

Imagina que queremos usar un color en varios elementos.

En CSS normal lo haríamos así:

```
h1 {  
    color: blue;  
}  
  
button {  
    background-color: blue;  
}
```

Si después decidimos cambiar el color azul por verde, tendríamos que buscar y cambiar manualmente en todas partes. 

 Con SASS usamos una variable y hacemos esto:

```
$primary-color: blue;  
  
h1 {  
    color: $primary-color;  
}  
  
button {  
    background-color: $primary-color;  
}
```

Si queremos cambiar el color, solo modificamos una sola línea (`$primary-color: green;`) y se actualizará en todo el código. 



Buenas Prácticas al Usar Variables en SASS

1 Usar nombres descriptivos en lugar de colores directos:

✗ Mal: \$red: red;

✓ Bien: \$danger-color: red;

2 Agrupar variables en un solo archivo (`variables.scss`) para mantener ordenado el código.

3 Usar variables para tamaños, fuentes, espaciado y bordes, no solo colores.

4 No exagerar con variables, solo crea las que realmente necesites.



❖ Ejemplo: Variables en SASS para Botones

```
// Definir variables
$primary-color: blue;
$secondary-color: gray;
$success-color: green;
$danger-color: red;

$font-size: 16px;
$border-radius: 5px;
$padding: 10px 20px;

// Estilos de botones
.btn {
  font-size: $font-size;
  padding: $padding;
  border-radius: $border-radius;
  color: white;
  border: none;
  cursor: pointer;

  &.primary {
    background-color: $primary-color;
  }

  &.secondary {
    background-color: $secondary-color;
  }

  &.success {
    background-color: $success-color;
  }

  &.danger {
    background-color: $danger-color;
  }
}
```

Vemos cómo definir variables y usarlas en botones.

➊ Ejemplo En este caso, simplemente creamos las clases sin seguir una metodología específica.

📌 Uso en HTML

```
<button class="btn primary">Primary</button>
<button class="btn secondary">Secondary</button>
<button class="btn success">Success</button>
<button class="btn danger">Danger</button>
```



¿Qué es & en SASS y para qué se usa?

El símbolo & en SASS se llama "selector padre" y se usa dentro de un selector para referirse a sí mismo. Es útil para evitar escribir el mismo selector varias veces y ayuda a mejorar la organización del código.

❖ Ejemplo básico de &

- ◆ En CSS tradicional, escribiríamos:
- ◆ Con & en SASS, podemos simplificarlo así:

```
.btn {  
    background-color: blue;  
    color: white;  
}  
  
.btn:hover {  
    background-color: darkblue;  
}
```

```
.btn {  
    background-color: blue;  
    color: white;  
  
&:hover {  
    background-color: darkblue;  
}
```

❖ Se reemplaza por .btn, haciendo que SASS lo convierta en .btn:hover cuando se compile a CSS.

Ejemplo Sass con BEM

```
// Definir variables
$primary-color: blue;
$secondary-color: gray;
$success-color: green;
$danger-color: red;

$font-size: 16px;
$border-radius: 5px;
$padding: 10px 20px;

// Estilos base de botones
.btn {
  font-size: $font-size;
  padding: $padding;
  border-radius: $border-radius;
  color: white;
  border: none;
  cursor: pointer;
  display: inline-block;
  text-align: center;

  &--primary {
    background-color: $primary-color;
  }

  &--secondary {
    background-color: $secondary-color;
  }

  &--success {
    background-color: $success-color;
  }

  &--danger {
    background-color: $danger-color;
  }
}
```

BEM (Block-Element-Modifier) es una metodología para escribir CSS más estructurado. En este caso, los botones seguirán la estructura btn--modificador.

📌 Uso en HTML con BEM

```
<button class="btn btn--primary">Primary</button>
<button class="btn btn--secondary">Secondary</button>
<button class="btn btn--success">Success</button>
<button class="btn btn--danger">Danger</button>
```

Anidamiento y Namespaces



Elementos Anidados y Namespaces

Cuando escribimos CSS normal, muchas veces tenemos que repetir selectores y eso hace que el código sea largo y difícil de leer.

Ejemplo en CSS (sin SASS):

```
.navbar {  
    background: black;  
    padding: 20px;  
}  
  
.navbar ul {  
    list-style: none;  
}  
  
.navbar ul li {  
    display: inline-block;  
}  
  
.navbar ul li a {  
    text-decoration: none;  
    color: white;  
}
```

Ejemplo en CSS (con SASS):

```
.navbar {  
    background: black;  
    padding: 20px;  
  
    ul {  
        list-style: none;  
  
        li {  
            display: inline-block;  
  
            a {  
                text-decoration: none;  
                color: white;  
            }  
        }  
    }  
}
```

✓ Anidación en SASS

En SASS podemos anidar elementos dentro de sus padres.

Mira cómo se vería el mismo código usando anidación

📌 ¿Qué hace esto?

- ul está dentro de .navbar, así que SASS entiende que es .navbar ul.
- li está dentro de ul, por lo que se convierte en .navbar ul li.
- a está dentro de li, así que genera .navbar ul li a.

🎯 ¿Por qué es útil?

- Hace que el código sea más corto y fácil de leer.
- Es más fácil ver la relación entre los elementos.

💡 Mucho código repetido, ¿cierto?



Namespaces (&) en Anidación

El símbolo & nos permite hacer referencia al selector padre.

💡 Ejemplo con & para hover:

```
.button {  
    background: blue;  
    color: white;  
    padding: 10px 20px;  
  
    &:hover {  
        background: darkblue;  
    }  
}
```

SASS lo compila a esto:

```
.button {  
    background: blue;  
    color: white;  
    padding: 10px 20px;  
}  
  
.button:hover {  
    background: darkblue;  
}
```

Uso de Parciales e Imports



Manejo de Parciales e Imports

💡 ¿Qué es un parcial en SASS?

Es un archivo SASS que divide el código en partes más organizadas.

💡 Ejemplo de una mala práctica (sin parciales):

```
// styles.scss
$primary-color: blue;
$secondary-color: gray;

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

.button {
| background: $primary-color;
| color: white;
| padding: 10px;
}

.header {
| background: $secondary-color;
| padding: 20px;
}
```

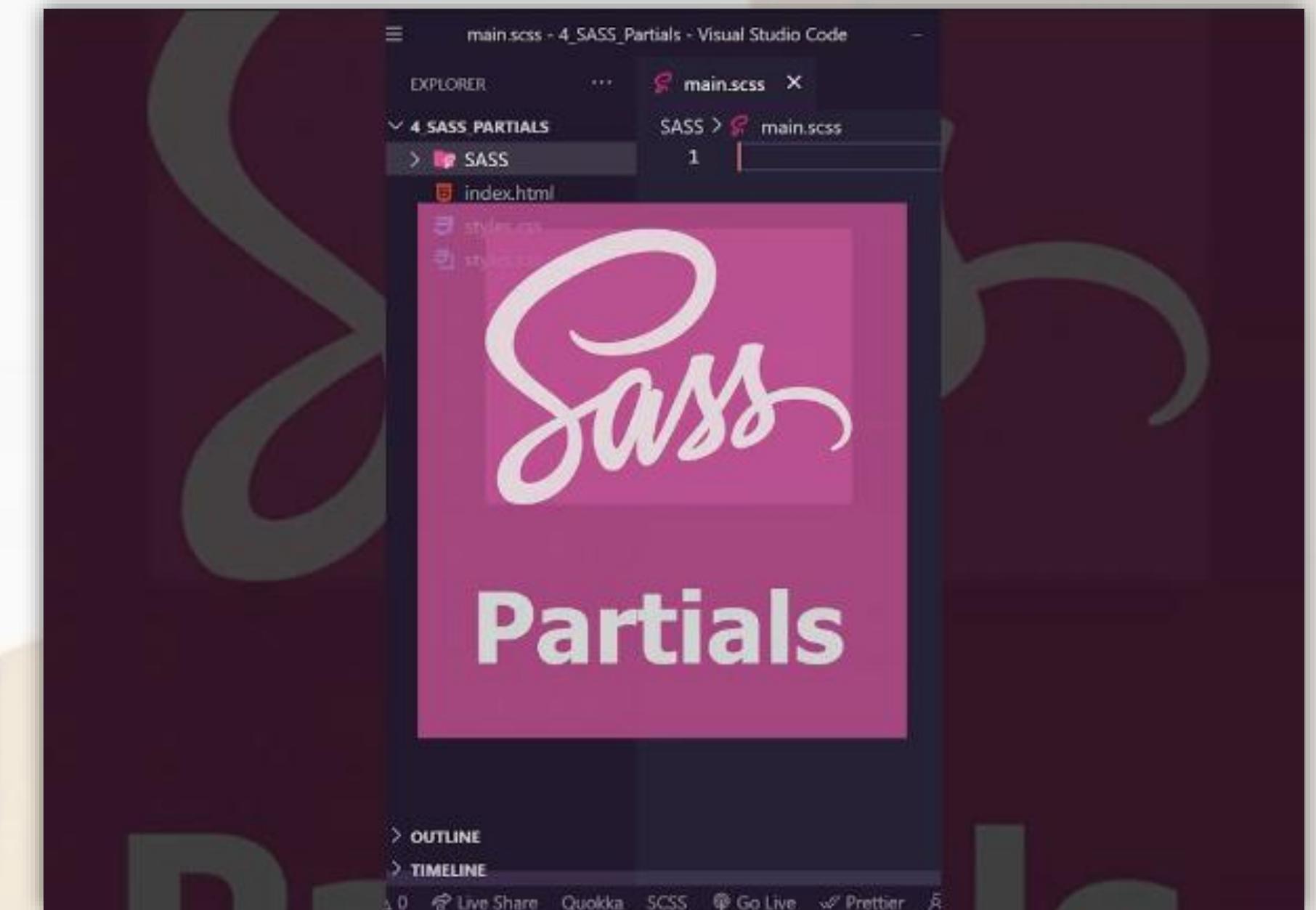


Problema: Todo el código está en un solo archivo. Cuando el proyecto crezca, se hará un caos.

Optimización y Modularización del Código CSS con Archivos Parciales en SASS

El uso de **archivos parciales en SASS** es una práctica clave para estructurar y modularizar el código de manera eficiente. Al dividir los estilos en diferentes archivos especializados, como variables, botones y encabezados, se mejora la **organización del proyecto**, facilitando la lectura y mantenimiento del código. Además, los parciales permiten separar los estilos en módulos reutilizables, lo que agiliza el desarrollo y evita la duplicación de código.

Otro beneficio importante es la **escalabilidad** del proyecto. A medida que el diseño de una aplicación o sitio web crece, gestionar un solo archivo CSS puede volverse complicado y poco práctico. Con los parciales, cada sección de la interfaz mantiene sus propios estilos sin interferir con otros elementos, permitiendo actualizaciones más rápidas y una mejor colaboración entre desarrolladores. 





Solución: Usar archivos parciales

- En SASS, un parcial es un archivo con un nombre que empieza con _ (guion bajo).
- Luego usamos @import o @use para unir los archivos.

📌 Estructura correcta con parciales:

```
/scss
  ├── _variables.scss
  ├── _buttons.scss
  ├── _header.scss
  └── main.scss
```

📌 `_buttons.scss` (Guarda estilos de botones)

```
@import 'variables';

.button {
  background: $primary-color;
  color: white;
  padding: 10px;
}
```

📌 `main.scss` (Importa todo)

```
@import 'variables';
@import 'buttons';
@import 'header';

body {
  font-family: $font-family;
}
```

Ejemplo de cada archivo parcial:

📌 `_variables.scss` (Guarda colores y tamaños)

```
$primary-color: blue;
$secondary-color: gray;
$font-family: Arial, sans-serif;
```

📌 `_header.scss` (Guarda estilos del header)

```
@import 'variables';

.header {
  background: $secondary-color;
  padding: 20px;
}
```

🎯 ¿Por qué usar parciales?

- ✓ Mantiene el código **ordenado y modular**.
- ✓ Permite **reutilizar código** fácilmente.
- ✓ Hace que el mantenimiento sea **más fácil**.

Uso de Mixins e Includes





Manejo de Mixins e Includes

💡 ¿Qué es un mixin (@mixin)?

Un **mixin** es como una **función** en SASS:

- Permite escribir código reutilizable.
- Puedes **pasarle parámetros** para personalizar estilos.

✖ Ejemplo sin Mixins (mala práctica)

```
.btn {  
    display: inline-block;  
    padding: 10px 20px;  
    border-radius: 5px;  
}  
  
.btn-large {  
    display: inline-block;  
    padding: 15px 25px;  
    border-radius: 5px;  
}
```

📌 Problema: Se repite código en **.btn** y **.btn-large**.

✓ Ejemplo con Mixins (@mixin y @include)

```
@mixin button($size) {  
    display: inline-block;  
    border-radius: 5px;  
  
    @if $size == small {  
        padding: 5px 10px;  
    } @else if $size == large {  
        padding: 15px 25px;  
    } @else {  
        padding: 10px 20px;  
    }  
  
.btn {  
    @include button(medium);  
}  
  
.btn-small {  
    @include button(small);  
}  
  
.btn-large {  
    @include button(large);  
}
```

📌 Se compila en:

```
.btn {  
    display: inline-block;  
    border-radius: 5px;  
    padding: 10px 20px;  
}  
  
.btn-small {  
    display: inline-block;  
    border-radius: 5px;  
    padding: 5px 10px;  
}  
  
.btn-large {  
    display: inline-block;  
    border-radius: 5px;  
    padding: 15px 25px;  
}
```

🎯 Ventajas de los Mixins:

- ✓ Evita escribir el mismo código muchas veces.
- ✓ Hace que los estilos sean más dinámicos.
- ✓ Es fácil cambiar los estilos sin modificar cada clase.

SASS @extend y @use

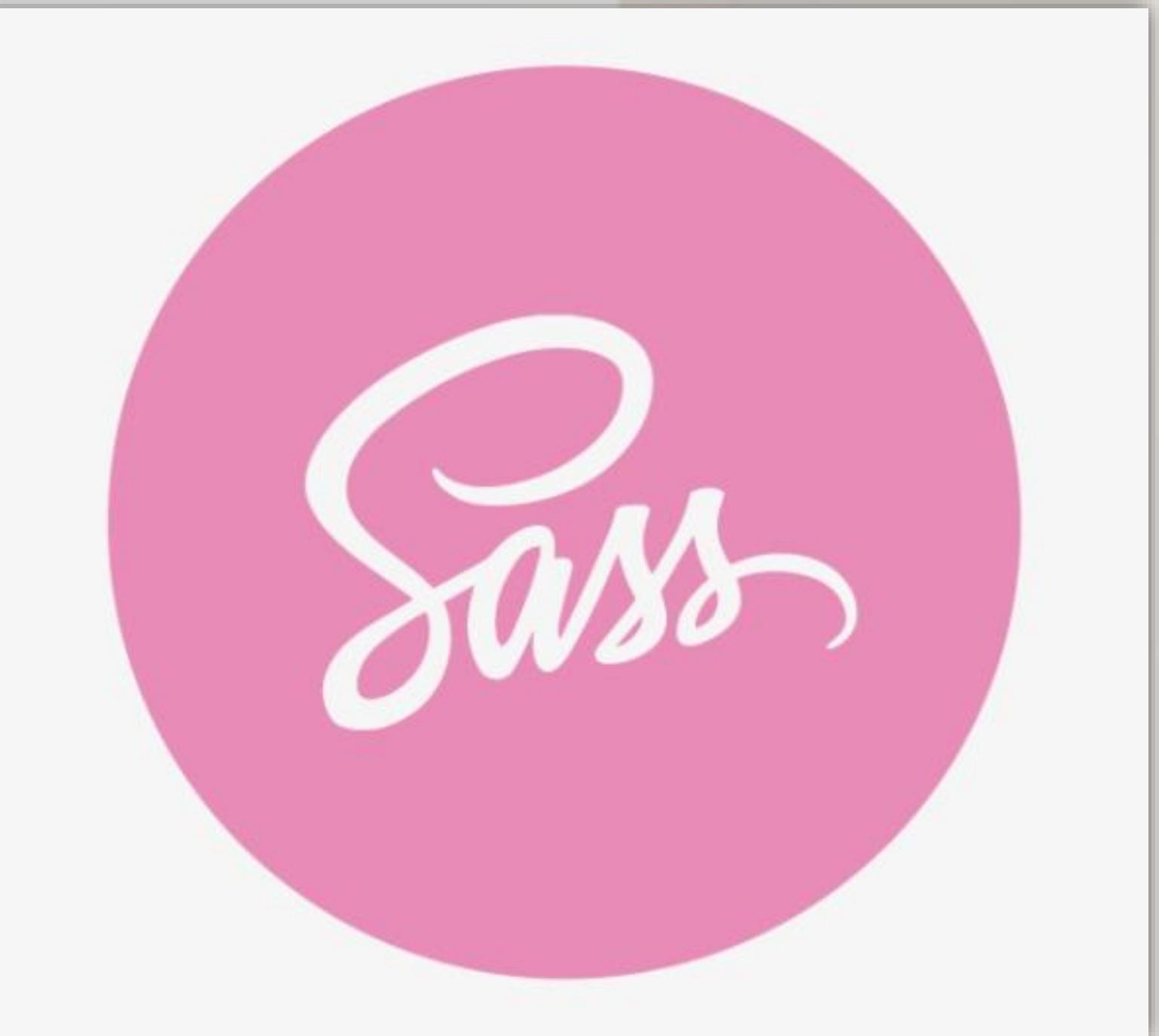


📌 SASS : @extend y @use

Si llegaste hasta aquí, ya sabes:

- Anidación (&)
- Parciales e Imports (@import)
- Mixins e Includes (@mixin, @include)

- ① **@extend** ✎ Para heredar estilos y evitar código repetido.
- ② **@use** ✎ Para importar archivos de forma moderna y evitar problemas.





IT Academy

by KIBERNUM