

Teoría Sass

¡Hola! 🖐️ Te damos la bienvenida a Sass

Exploraremos cómo utilizar Sass, un preprocesador de CSS, para llevar tus habilidades en desarrollo web al siguiente nivel.

HTML, CSS y JavaScript son fundamentales para crear sitios web dinámicos y atractivos. Sin embargo, a medida que los proyectos se vuelven más grandes y complejos, gestionar y mantener hojas de estilo CSS puede volverse complicado y propenso a errores. Aquí es donde entra en juego Sass.

Sass, acrónimo de Syntactically Awesome Style Sheets, es un preprocesador de CSS que agrega características poderosas y eficientes al lenguaje de estilos web. Con Sass, podrás escribir código CSS más limpio, modular y reutilizable, lo que facilitará la creación y el mantenimiento de estilos en tus proyectos.

En este curso, comenzaremos desde lo básico, cubriendo los conceptos fundamentales de Sass. Aprenderás cómo definir variables para colores, tamaños de fuente y otras propiedades comunes, lo que te permitirá actualizar rápidamente los estilos en todo tu proyecto.

Exploraremos el anidamiento de selectores en Sass, lo que simplifica la estructura de tu código y mejora su legibilidad. Descubrirás cómo utilizar mixins para reutilizar estilos en diferentes partes de tu proyecto, lo que te ayudará a mantener una base coherente y ahorrar tiempo de desarrollo.

Además, te sumergirás en las funciones y directivas de control de Sass, lo que te permitirá crear estilos más dinámicos y adaptables. Aprenderás a dividir tus estilos en archivos parciales y utilizar la importación para mantener tu código organizado y fácilmente escalable.

A lo largo del curso, no solo aprenderás los conceptos teóricos de Sass, sino que también trabajarás en ejercicios prácticos y proyectos para aplicar lo que has aprendido. Obtendrás experiencia real en el uso de Sass en el contexto de

proyectos web reales, lo que te preparará para enfrentar desafíos del mundo real.

Al final de este curso, estarás preparado para utilizar Sass como una herramienta poderosa en tu flujo de trabajo de desarrollo web. Mejorarás tu productividad, escribirás estilos más eficientes y mantendrás un código más limpio y modular.

Así que, ¡prepárate para llevar tus habilidades en HTML, CSS, JavaScript y desarrollo web al siguiente nivel con Sass! Únete a este emocionante viaje y descubre cómo potenciar tus proyectos web con este increíble preprocesador de CSS.

¡Que comience el viaje! 🚀

Ambiente de trabajo

Instalación

Sass es un preprocesador CSS y para comenzar a usarlo vamos a necesitar realizar su instalación.

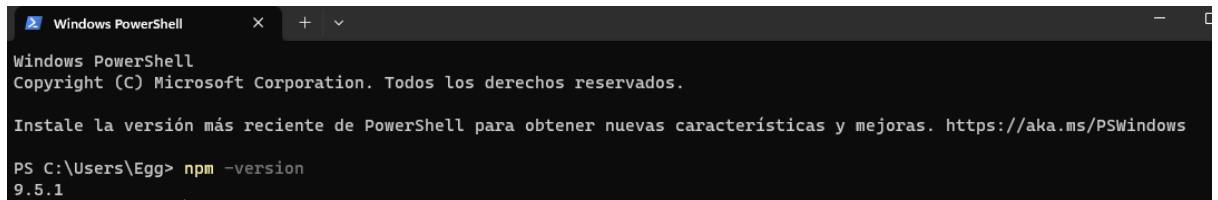
La página para descargar Sass es 👉 <https://sass-lang.com/install/>

💡 **Si no tienes instalado Node.js vamos a necesitarlo para instalar Sass, entonces vamos a <https://nodejs.org/> para descargarlo.**

Paso a paso

También puedes ver el siguiente video 👉 [Instala Sass](#)

1. Descargar node.js desde <https://nodejs.org/>
2. Una vez lo tengamos descargado vamos a verificar que se haya instalado correctamente, ingresando en nuestra terminal y tipeando el comando **"npm --v"**.



```
Windows PowerShell
Copyright (C) Microsoft Corporation. Todos los derechos reservados.

Instale la versión más reciente de PowerShell para obtener nuevas características y mejoras. https://aka.ms/PSWindows

PS C:\Users\Egg> npm -version
9.5.1
```

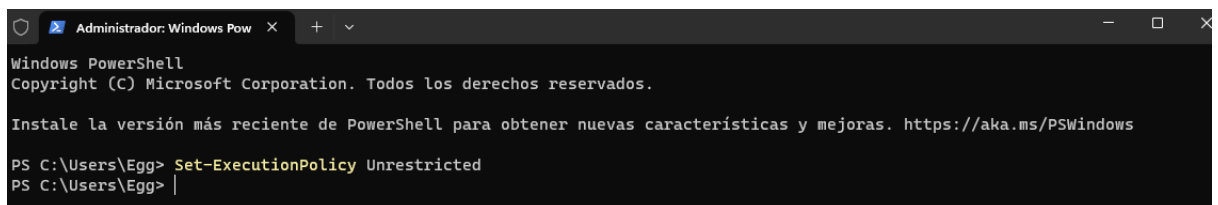
Si vemos números, eso querrá decir que está correctamente instalado.

3. Luego de verificar que está instalado, podremos ingresar en **Visual Studio Code**, abrir una carpeta en donde querramos empezar a trabajar y crear un archivo que se llame por ejemplo **main.scss**.
4. Abrimos la terminal en visual desde **Archivo > Terminal > Nueva Terminal** y ejecutamos el siguiente código **"npm install -g sass"**. Cuando el comando termine de ejecutarse ¡**Tendremos instalado SASS!**

Luego, para empezar a trabajar vamos a ver la parte de compilación

En el caso de tener errores al momento de realizar estos pasos podemos:

1. Cerrar y abrir visual studio code como administradores. (click derecho en la app y ejecutar como admin o abrir como admin).
2. Si sigue dando error darle click derecho en nuestro botón de inicio de windows y abrir Windows Powershell (administrador). Debemos ejecutar el comando **"Set-ExecutionPolicy Unrestricted"**.



```
Administrador: Windows Pow
Windows PowerShell
Copyright (C) Microsoft Corporation. Todos los derechos reservados.

Instale la versión más reciente de PowerShell para obtener nuevas características y mejoras. https://aka.ms/PSWindows

PS C:\Users\Egg> Set-ExecutionPolicy Unrestricted
PS C:\Users\Egg> |
```

3. Ahora si puedes probar ejecutar el comando **"npm -version"**.

Sintaxis

SCSS VS. SASS

Sass admite dos sintaxis diferentes.


SCSS

La sintaxis **SCSS** utiliza la extensión de archivo `.scss`. Con algunas pequeñas excepciones, es un superconjunto de CSS, lo que significa que prácticamente toda la sintaxis que conocemos de CSS es también válida en SCSS. **Debido a su similitud con CSS, es la sintaxis más fácil de acostumbrarse y la más popular.**

SASS

La sintaxis de indentación fue la sintaxis original de Sass, y utiliza la extensión de archivo `.sass`. Debido a esta extensión, a veces se le llama simplemente "Sass". La sintaxis de indentación admite todas las mismas características que SCSS, pero utiliza la indentación en lugar de llaves y punto y coma para describir el formato del documento.

En general, cada vez que escribirías llaves en CSS o SCSS, en la sintaxis de indentación solo debes indentar un nivel más profundo. Y cada vez que una línea termina, se cuenta como un punto y coma.

 **Es por este motivo que recomendamos utilizar la extensión `.scss` en los archivos que creamos, ya que su sintaxis es más parecida a la de CSS que ya conocemos.**

Primeros pasos con Sass

Compilación

Para trabajar con Sass no basta solo con tener creado el archivo scss. Vamos a necesitar compilarlo.

¿Cómo compilamos?

Para ello vamos a hacer uso de la consola, o símbolo del sistema en windows.

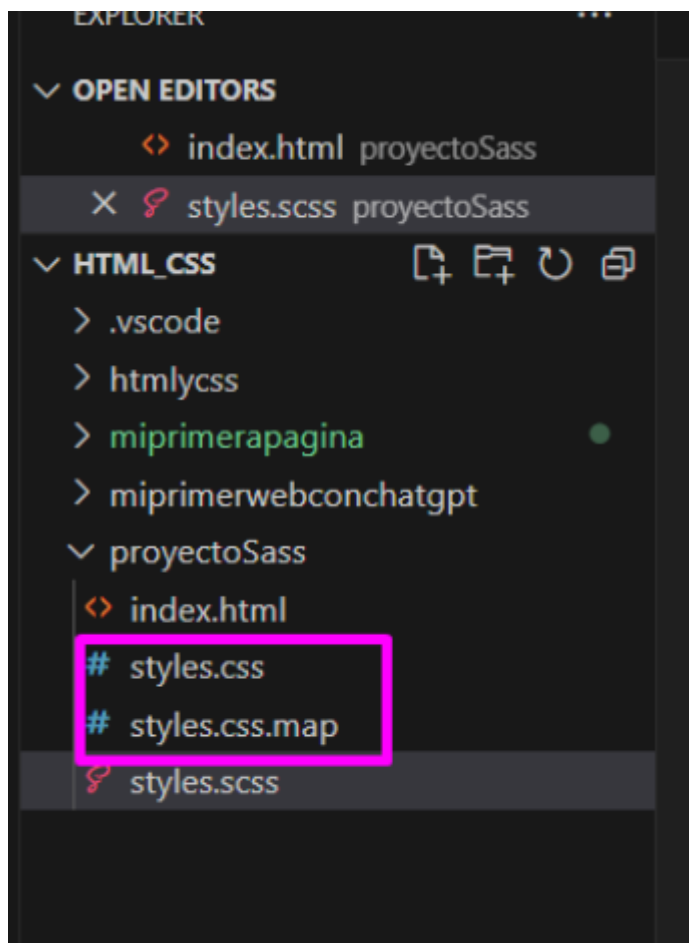
Entonces tendremos que:

- Abrir la consola
- Navegar con el comando `cd` hasta la carpeta donde está el archivo scss
- Una vez allí ejecutamos la siguiente línea: **`sass --watch styles.scss styles.css`**

```
^C
C:\Users\Egg\Dropbox\Projects\HTML_CSS\proyectoSass>sass --watch styles.scss styles.css
[2023-07-04 15:59] Compiled styles.scss to styles.css.
Sass is watching for changes. Press Ctrl-C to stop.
```

Esto creará un archivo .css y uno .map en nuestra lista de archivos, dentro de VSCode.

Se verá así:



💡 El archivo .css que se cree va a contener el código del archivo .scss adaptado al css

Por último, el archivo que tengo que linkear en el **index.html**, es el .css que se crea luego de la compilación (esto tal cual lo veníamos haciendo con anterioridad). En el caso del ejemplo será **styles.css**

```
<head>
  <title>Ejemplo de Sass</title>
  <link rel="stylesheet" href="styles.css"> <!-- Archivo CSS compilado desde Sass -->
</head>
```

Sass vs. Css

El principal diferencial de sass frente a css es que podemos trabajar con variables.

¿Por qué esto es útil?

✎ Veamos un ejemplo, si tenemos un color que se repite varias veces en nuestra página web o en el proyecto que estemos trabajando, podemos guardarlo en una variable y sólo con cambiar esa variable ya se modificará en todos los lugares que le indiquemos.

En este archivo css tengo las siguientes opciones:

```
header {  
  background-color: #333;  
  padding: 20px;  
}  
  
footer {  
  background-color: #333;  
  padding: 40px 0;  
  color: #fff;  
  text-align: center;  
}
```

Tanto el header como el footer se utiliza el mismo color **#333**. Ahora bien, si a futuro decido que el color va a ser blanco, tendré que cambiarlo en ambos lugares y no olvidarme de ninguno. En cambio con Sass, podré guardar el color en una variable y usar esa variable tanto para el header como para el footer, y cuando quiera modificar el color sólo tendré que hacerlo en la variable.

Quedaría de la siguiente manera:

```
// Variables
$color-principal: #333;

header {
  background-color: $color-principa;
  padding: 20px;
}

footer {
  background-color: $color-principa;
  padding: 40px 0;
  color: #fff;
  text-align: center;
}
```

Como vemos en el ejemplo, se crea la variable **\$color-principal**, se le asigna el color #333 y luego a la propiedad **background-color**, se le asigna la variable como valor.

Variables

Piensa en las variables como una forma de almacenar información que desees reutilizar en toda tu hoja de estilos. Puedes almacenar cosas como colores, conjuntos de fuentes o cualquier valor de CSS que creas que querrás reutilizar. Sass utiliza el símbolo \$ para declarar una variable.

 Veamos un ejemplo:

En CSS

```
body {  
  font: 100% Helvetica, sans-serif;  
  color: #333;  
}
```

En Sass

```
$tipo-fuente: Helvetica, sans-serif;  
$color-primario: #333;  
  
body {  
  font: 100% $tipo-fuente;  
  color: $color-primario;  
}
```

Cuando compilamos el archivo Sass, toma las variables que definimos para **\$tipo-fuente** y **\$color-primario**, y genera CSS normal con los valores de nuestras variables colocados en el CSS. Esto puede ser extremadamente poderoso al trabajar con colores de marca y mantenerlos consistentes en todo el sitio.

Nesting (anidamiento)

Esta funcionalidad nos permite estructurar y organizar nuestro código de manera más intuitiva y legible, así como nos ahorra tiempo y esfuerzo en el proceso.

El anidamiento en Sass se basa en la idea de que los estilos CSS de un elemento hijo a menudo están relacionados y son dependientes de los estilos del elemento padre. En lugar de repetir selectores y propiedades una y otra vez, el **Nesting** nos permite agruparlos de manera jerárquica, siguiendo la estructura del documento HTML.

Para utilizar el Nesting en Sass, simplemente debemos escribir nuestros selectores hijos dentro de los selectores padres, separándolos por espacios. Esto crea una relación de anidamiento visualmente clara y concisa.

 Veamos un ejemplo:

```
.contenedor {
  width: 100%;

  .titulo {
    font-size: 24px;
    color: #333;
  }

  .contenido {
    background-color: #f5f5f5;
    padding: 10px;

    p {
      margin-bottom: 10px;
    }

    a {
      text-decoration: none;
      color: blue;

      &:hover {
        text-decoration: underline;
      }
    }
  }
}
```

En el ejemplo anterior, estamos anidando los estilos de los elementos **.titulo**, **.contenido**, **p** y **a** dentro del selector **.contenedor**. Esto nos permite mantener una estructura clara y concisa, reflejando la jerarquía de los elementos HTML y facilitando la lectura y comprensión del código.

El anidamiento en Sass no solo se limita a los selectores, también se puede aplicar a las propiedades CSS. Esto significa que podemos agrupar propiedades comunes dentro de los selectores anidados, evitando la repetición innecesaria y mejorando la legibilidad del código.

 Veamos un ejemplo:

```
.contenedor {  
  width: 100%;  
  
  .titulo {  
    font: {  
      size: 24px;  
      weight: bold;  
    }  
    color: #333;  
  }  
}
```

En este caso, estamos utilizando el anidamiento para agrupar las propiedades de la fuente del `.titulo` en una sola declaración. Esto simplifica el código y hace que sea más fácil de leer y mantener.

En resumen, el Nesting en Sass nos ofrece una manera intuitiva y eficiente de estructurar y organizar nuestros estilos CSS. Al aprovechar el anidamiento, podemos reflejar la estructura del documento HTML de forma clara y concisa, evitando la repetición innecesaria y mejorando la legibilidad del código. Esta característica es una herramienta imprescindible para cualquier desarrollador web que desee optimizar su flujo de trabajo y mantener estilos más mantenibles y escalables.

En un ejemplo anterior vimos que se utilizaba el “&”, este símbolo se utiliza para indicar al padre.

 Veamos un ejemplo:

```
<div class="main">
  <h1 class="main_encabezado">Título</h1>
  <p class="main_parrafo">Lorem ipsum dolor sit amet,
  consectetur adipiscing elit.</p>
  <h2 class="main_subtitulo">Subtítulo</h2>
  <span class="main_parrafo">Lorem ipsum dolor sit amet,
  consectetur adipiscing elit.</span>
</div>
```

En el siguiente ejemplo vemos que el div con la clase **main** tiene varios hijos dentro. Dos de ellos tienen la clase **main_parrafo**. Entonces, si quisiéramos anidar un css, sería así:

```
.main {
  width: 100%;
  background-color: $color-cuadrado2;
  text-align: center;
  &_parrafo {
    font-size: tamano-fuente(parrafo,2.1);
  }
}
```

Ahora bien, esto, traducido al CSS se ve así

```
.main {
  width: 100%;
  background-color: #e4ec4f;
  text-align: center;
}
.main_parrafo {
  font-size: 33.6px;
}
```

Pero lo que queremos es que el font-size se aplique a .main .main_parrafo. Para ello tendremos que agregarle `#{}` a nuestro .scss.

Quedaría así:

```
.main {
  width: 100%;
  background-color: $color-cuadradito2;
  text-align: center;
}
#&_parrafo {
  font-size: tamano-fuente(parrafo,2.1);
}
```

y finalmente el css se verá así:

```
.main {
  width: 100%;
  background-color: #e4ec4f;
  text-align: center;
}
.main .main_parrafo {
  font-size: 33.6px;
}
```

Partials

Los partials en Sass son archivos que contienen fragmentos de código reutilizables y modulares. Estos archivos tienen una extensión de nombre de archivo .scss y se utilizan para dividir estilos CSS en partes más pequeñas y manejables. Los partials en Sass desempeñan un papel importante en la organización y estructuración de proyectos de estilos más grandes.

La convención para nombrar los partials en Sass es comenzar el nombre del archivo con un guion bajo (`_`). Por ejemplo, un partial para estilos de botones podría llamarse `_botones.scss`. El guion bajo al comienzo del nombre del archivo indica a Sass que el archivo es un partial y no debe ser compilado directamente a un archivo CSS.

Una de las ventajas clave de los partials en Sass es su capacidad para dividir los estilos en componentes más pequeños y específicos. Esto facilita la reutilización de estilos y mejora la legibilidad y mantenibilidad del código. Además, los partials ayudan a evitar la repetición innecesaria de código, ya que se pueden importar en diferentes archivos según sea necesario.

Para importar un partial en un archivo Sass principal, se utiliza la directiva `@import`. La importación de un partial permite que su contenido esté disponible en el archivo principal, lo que permite que los estilos se compartan y reutilicen de manera efectiva.

```
// En el archivo main.scss
@import 'botones';

// Resto del código del archivo main.scss
```

En este ejemplo, el partial `_botones.scss` se importa en el archivo `main.scss`. Esto hace que los estilos definidos en el partial estén disponibles en el archivo principal, lo que permite utilizar los estilos de botones en el contexto del proyecto.

Además de dividir los estilos en componentes, los partials también pueden contener variables, mixins, funciones y otros constructores de Sass. Esto significa que se pueden crear partials para almacenar estilos comunes, como paletas de

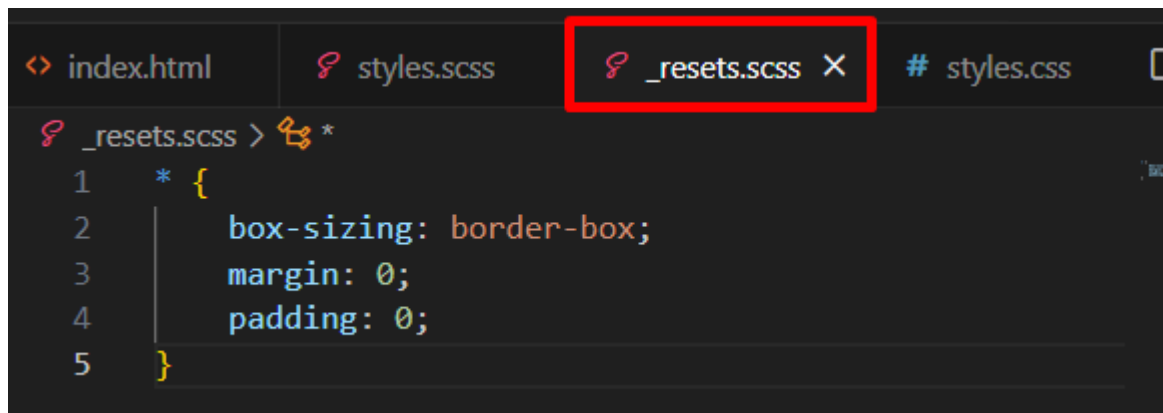
colores, estilos de tipografía o estilos de animación, y luego importarlos según sea necesario.

En resumen, los partials en Sass son archivos que contienen fragmentos de código reutilizables y modulares. Al dividir los estilos en partes más pequeñas y específicas, los partials mejoran la organización, reutilización y mantenibilidad del código en proyectos Sass más grandes. Al utilizar la directiva `@import`, los partials pueden ser importados en archivos Sass principales para compartir y utilizar los estilos definidos en ellos.

 Veamos un ejemplo:

Podríamos usar un partial, para por ejemplo tener un archivo diferente con estilos de reset. Llamamos estilos de reset a los estilos que colocamos en todos nuestros archivos CSS para comenzar a plasmar los estilos.

Entonces, creamos un archivo que se llame **`_reset.scss`** y colocamos allí los estilos que queremos



```
<> index.html  ? styles.scss  ? _reset.scss X  # styles.css
? _reset.scss > ? *
1  * {
2      box-sizing: border-box;
3      margin: 0;
4      padding: 0;
5  }
```

Luego, tenemos que importarlo en nuestro archivo principal, de la siguiente manera:



```
<> index.html  ? styles.scss X  ? _reset.scss
proyectoSass > ? styles.scss > ...
1  @import 'reset';
2
```

Listo, ahora se importarán los estilos que creamos por separado a nuestro archivo principal.

Esta forma de trabajar es particularmente útil cuando se trabaja en equipo, ya que se puede ir trabajando en componentes por separado, por ejemplo, de botones, y ya solo con importarlo lo podremos reutilizar.

Maps

Es similar al concepto de arrays (arreglos) en los lenguajes de programación.

Es una lista de pares de llave valor, se escriben así:

(«expresión»: «expresión», «expresión»: «expresión»)

La expresión antes de los : es la llave, y la expresión luego de los : es el valor.

 Veamos un ejemplo:

Primero, defino el mapa

```
$font-weight: (  
  "regular": 400,  
  "medium": 500,  
  "bold": 700,  
);
```

En este caso estamos definiendo diferentes pesos de fuentes.

Cuando lo queramos usar, tendremos que asignarle el valor a la variable


```
$font-weight: (  
  "regular": 400,  
  "medium": 500,  
  "bold": 700,  
);  
  
/*ACA LOS APLICO*/  
  
body {  
  font-weight: map-get($font-weight, bold);  
}
```

En este caso estamos aplicando al body(todo el documento), que el tipo de fuente sea bold y valdrá 700.

Lo bueno de los mapas es que si quisiera a futuro cambiar algún valor lo puedo hacer simplemente desde la variable.

Functions (funciones)

Las funciones son herramientas poderosas que te permiten manipular valores, realizar cálculos y generar estilos dinámicos dentro de tus hojas de estilo. Proporcionan una forma de escribir piezas de código reutilizables que se pueden llamar con diferentes argumentos para producir los resultados deseados. Sass proporciona una variedad de funciones integradas, y también puedes definir tus propias funciones personalizadas.

Las funciones en Sass tienen una sintaxis específica. Se llaman utilizando el nombre de la función seguido de paréntesis (). Cualquier argumento requerido por la función se pasa dentro de los paréntesis, separados por comas.

Por ejemplo:

```
// Llamando a la función integrada `darken()`  
$nuevo-color: darken($color-base, 10%);
```

En este ejemplo, **darken()** es la función que se está llamando, y toma dos argumentos: **\$color-base** y **10%**. El resultado de la función se asigna a la variable **\$nuevo-color**.

En Sass existen dos tipos de funciones:

- Funciones integradas
- Funciones personalizadas

Las integradas son funciones preestablecidas que podemos utilizar. En cambio, las funciones personalizadas son las que podemos crear desde cero.

Las funciones personalizadas son útiles cuando tienes operaciones o cálculos específicos que no están disponibles mediante las funciones integradas.



Veamos un ejemplo:

```
// Mapa de peso de fuente
$font-weight: (
  "regular": 400,
  "medium": 500,
  "bold": 700,
);

// Función para obtener el peso de fuente
@function peso($nombre-peso){
  @return map-get($font-weight, $nombre-peso);
}

// Uso de la función
.main {
  width: 100%;
  background-color: $color-cuadradito2;
  text-align: center;

  #{&}_parrafo {
    font-size: tamano-fuente(parrafo,2.1);
    font-weight: peso(bold);

    &:hover {
      color: $color-cuadradito3,
    }
  }
}
```

En el siguiente ejemplo vemos el mapa que creamos anteriormente con los diferentes pesos de fuentes.

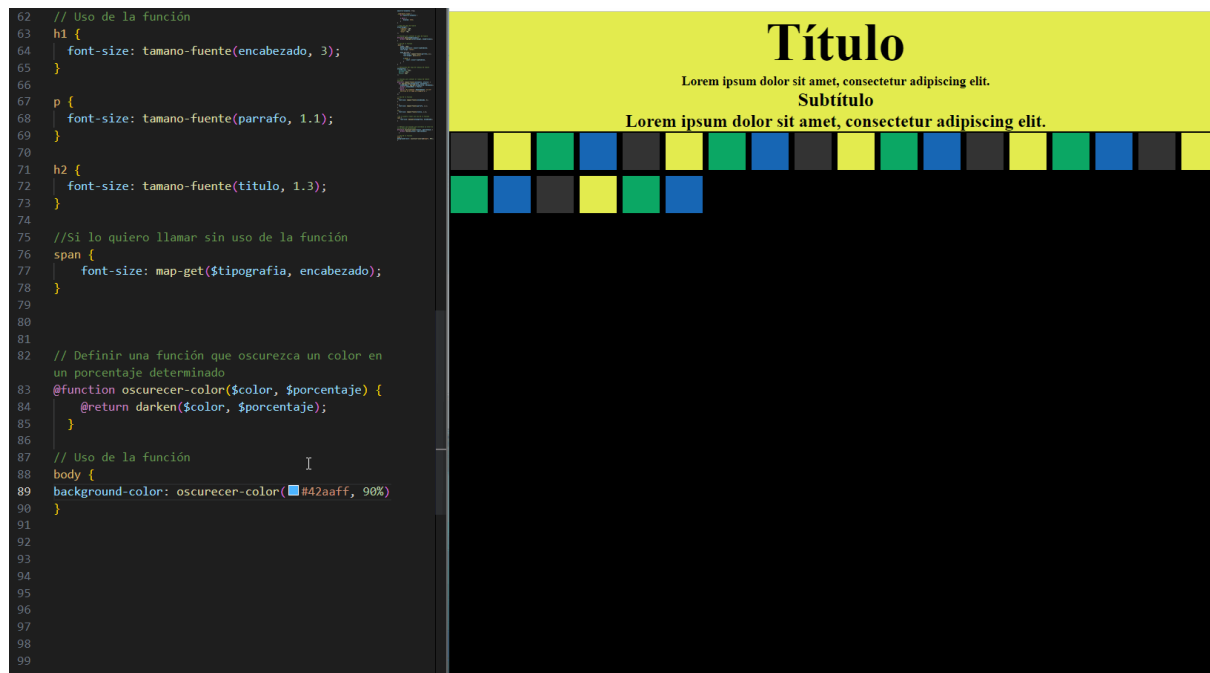
Luego creamos una función **peso** que recibe un parámetro **\$nombre-peso**, que es el nombre del peso (regular, medium, bold).

La función va a devolver el peso del nombre que le pasamos como parámetro.

Si vemos luego al usarla colocamos **font-weight: peso(bold);**

Esto en CSS se verá como **font-weight: 700;**

En el siguiente ejemplo vemos un gif de una función que cambia el color de fondo en base a un porcentaje.



Flow Controls (control de flujo)

Así como en Javascript teníamos opciones de manejar la estructuras de control, Sass proporciona una serie de reglas **at-rules (@)** que permiten controlar si los estilos van a emitirse o emitirlos varias veces con pequeñas variaciones. También se pueden utilizar en mixins y funciones para escribir algoritmos simples que facilitan la escritura de tu código en Sass.

Sass admite cuatro reglas de control de flujo:

- **@if** controla si un bloque es evaluado o no.
- **@each** evalúa un bloque para cada elemento en una lista o cada par en un mapa.
- **@for** evalúa un bloque un cierto número de veces.
- **@while** evalúa un bloque hasta que se cumpla una condición determinada.



Veamos un ejemplo de cómo utilizar el if:

```
$mostrar-elemento: false;  
  
.cuadradito.tipo1 {
```

```
@if $mostrar-elemento {  
  display: block;  
} @else {  
  display: none;  
}  
}
```

En el siguiente ejemplo el css ocultará los cuadraditos de tipo1, dado que la variable **\$mostrar-elemento** está evaluada en **false**.

If y Maps

Si queremos usar maps e if combinados, podríamos hacerlo a través de una función y definir, por ejemplo, diferentes tamaños de fuentes para h1,h2, y p (o lo que queramos).

 Veamos un ejemplo:

```

// Definición del mapa de tamaños de fuente
$tipografia: (
  encabezado: 24px,
  parrafo: 16px,
  titulo: 20px
);

// Función para obtener el tamaño de fuente escalado
@function tamano-fuente($elemento, $factor) {
  @if map-has-key($tipografia, $elemento) {
    $tamanoBase: map-get($tipografia, $elemento);
    @return $tamanoBase * $factor;
  } @else {
    @error "El elemento '#{ $elemento }' no está definido en el mapa de tipografía.";
  }
};

// Uso de la función
h1 {
  font-size: tamano-fuente(encabezado, 3);
}

p {
  font-size: tamano-fuente(parrafo, 1.1);
}

h2 {
  font-size: tamano-fuente(titulo, 1.3);
}

```

En este ejemplo, primero definimos el mapa **\$tipografia** que contiene los tamaños de fuente para diferentes elementos. Luego, creamos la función **tamano-fuente** que toma dos argumentos: **\$elemento** y **\$factor**. La función verifica si el elemento existe en el mapa y, si es así, obtiene el tamaño de fuente base y lo multiplica por el factor proporcionado.

Después, utilizamos la función **tamano-fuente** en los estilos de los elementos h1, p y h2. Puedes tomarlo como base y agregar más estilos y elementos según tus necesidades.

Math Operators (operadores matemáticos)

En Sass, los operadores matemáticos son herramientas poderosas que te permiten realizar cálculos numéricos en tus estilos. Estos operadores te permiten realizar operaciones como suma, resta, multiplicación y división, y son útiles cuando necesitas realizar cálculos dinámicos para establecer valores de propiedades.

A continuación, te presentamos los operadores matemáticos disponibles en Sass:

- **Suma (+):** El operador de suma se utiliza para sumar dos valores. Por ejemplo:

```
$resultado: 10px + 5px; // Resultado: 15px
```

- **Resta (-):** El operador de resta se utiliza para restar un valor de otro. Por ejemplo:

```
$resultado: 10px - 5px; // Resultado: 5px
```

- **Multiplicación (*):** El operador de multiplicación se utiliza para multiplicar dos valores. Por ejemplo:

```
$resultado: 2 * 3; // Resultado: 6
```

- **División (/):** El operador de división se utiliza para dividir un valor por otro. Por ejemplo:

```
$resultado: 10 % 3; // Resultado: 1
```

Además de estos operadores básicos, Sass también proporciona funciones matemáticas útiles, como `round()`, `ceil()`, `floor()`, `abs()`, etc., que te permiten redondear números, obtener el valor absoluto, etc.

Es importante tener en cuenta que Sass sigue las reglas estándar de precedencia de operadores matemáticos, lo que significa que las operaciones se evalúan en el siguiente orden: paréntesis, multiplicación y división, y finalmente suma y resta.

Estos operadores matemáticos en Sass te permiten realizar cálculos dinámicos y simplificar tu código, ya que puedes combinar valores numéricos con unidades (como píxeles) y realizar operaciones con ellos de manera sencilla.

 Veamos un ejemplo

- Ejemplo de suma y multiplicación:

```
$ancho-base: 200px;
$espaciado: 20px;

.elemento {
  width: $ancho-base + $espaciado; // Suma: 200px + 20px = 220px
  height: $ancho-base * 2; // Multiplicación: 200px * 2 = 400px
}
```

En este ejemplo, se utiliza el operador de suma para sumar el valor de **\$ancho-base** con **\$espaciado**, y el resultado se asigna como el ancho del elemento. También se utiliza el operador de multiplicación para multiplicar el valor de **\$ancho-base** por 2, y el resultado se asigna como la altura del elemento.

- Ejemplo de cálculo de porcentaje:

```
$total: 500px;
$porcentaje: 20%;

.barra-progreso {
  width: ($porcentaje / 100) * $total; // División y multiplicación: (20 / 100) * 500px = 100px
}
```


En este ejemplo, se utiliza el operador de división para dividir el valor de **\$porcentaje** por 100 y luego se multiplica por el valor de **\$total**. El resultado se asigna como el ancho de la barra de progreso.

Estos ejemplos ilustran cómo puedes utilizar los operadores matemáticos en Sass para realizar cálculos y generar valores dinámicos en tus estilos. Puedes combinar variables, valores con unidades y realizar operaciones matemáticas para obtener los resultados deseados.

Mixins

En Sass, un mixin es un bloque de código reutilizable que contiene un conjunto de declaraciones CSS. Los mixins te permiten definir estilos que se pueden incluir y reutilizar en varias partes de tu hoja de estilos. Esto facilita la escritura de estilos más limpios, más modulares y más fáciles de mantener.

El uso de mixins en Sass sigue la siguiente sintaxis:

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

Veamos paso a paso cómo sería una declaración de un Mixin

1. **Definir el mixin:**

Puedes definir un mixin utilizando **@mixin** seguida de un nombre descriptivo para el mismo. Dentro del bloque de código, puedes incluir cualquier número de declaraciones CSS que desees.

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

2. Incluir el mixin:

Para utilizar el mixin en un selector específico, debes usar la directiva **@include** seguida del nombre del mixin.

```
.container {  
  @include center-element;  
}
```

3. Compilar el código

Los mixins en Sass también pueden aceptar argumentos, lo que los hace aún más flexibles y poderosos. Puedes pasar valores a los mixins utilizando parámetros y luego utilizar esos valores dentro del bloque del mixin.

```
@mixin text-color($color) {  
  color: $color;  
}  
  
.title {  
  @include text-color(blue);  
}
```

En este caso, el mixin `text-color` acepta un parámetro **\$color**, que se utiliza para establecer el color del texto. Luego, al incluir el mixin en el selector **.title** y pasar el valor `blue` como argumento, el texto dentro de ese selector tendrá el color azul.

En resumen, los mixins en Sass te permiten definir bloques de código CSS reutilizables que se pueden incluir en diferentes partes de tu hoja de estilos. Esto facilita la escritura y el mantenimiento de estilos más modulares y legibles.

Un ejemplo frecuente es utilizar el mixin para elegir entre **Light Theme** o **Dark Theme** (modo claro o modo oscuro).

 Veamos un ejemplo:

```
@mixin theme($color-theme){
  @if $color-theme {
    background: lighten($color-cuadrado3,
      50%);
    color: darken($color-cuadrado1, 100%);
  } @else {
    background: lighten($color-cuadrado3,
      0%);
    color: darken($color-cuadrado2, 0%);
  }
}

.color-theme {
  @include theme($color-theme: true);
}
```

En el ejemplo tendríamos que agregarle la clase **color-theme** a los elementos que quisiéramos que cambiasen de color y en el mixin lo que hacemos es agregar un **if** que oscurezca o aclare según el parámetro **\$color-theme** sea true o false.

Diferencia entre Mixin y Function

Si bien son parecidas, las funciones se utilizan para devolver valores, como veíamos en un ejemplo anterior devolvía por ejemplo un porcentaje de opacidad. En cambio, los Mixins son bloques de código que vamos a necesitar repetir varias veces de la misma manera. Deben definir ciertos estilos.

Extension & Inheritance (extender elementos o herencia)

En Sass, tanto la extensión (extension) como la herencia (inheritance) son mecanismos que te permiten reutilizar y compartir estilos entre diferentes selectores o reglas. Aunque ambos conceptos son similares en términos de reutilización de estilos, existen algunas diferencias clave entre ellos.

Extensión (extension):

La extensión en Sass te permite tomar estilos existentes de un selector y aplicarlos a otro selector utilizando la directiva **@extend**. Esto significa que el segundo selector heredará todos los estilos del primer selector. La extensión es útil cuando deseas aplicar estilos similares a múltiples selectores sin tener que repetir el código.

Por ejemplo, supongamos que tienes los siguientes estilos en Sass:

```
.btn {  
  color: white;  
  background-color: blue;  
  padding: 10px;  
}  
  
.btn-primary {  
  @extend .btn;  
  font-weight: bold;  
}  
  
.btn-secondary {  
  @extend .btn;  
  border: 1px solid gray;  
}
```

En este caso, la clase **.btn-primary** hereda todos los estilos de **.btn** y también agrega una propiedad **font-weight: bold**. De manera similar, la clase **.btn-secondary** también hereda los estilos de **.btn** y agrega una propiedad **border: 1px solid gray**. El resultado final en CSS será:

```
.btn, .btn-primary, .btn-secondary {  
  color: white;  
  background-color: blue;  
  padding: 10px;  
}  
  
.btn-primary {  
  font-weight: bold;  
}  
  
.btn-secondary {  
  border: 1px solid gray;  
}
```

Herencia (inheritance):

La herencia en Sass se basa en el concepto de herencia de estilos CSS y se logra utilizando el selector **@extend junto con una clase base o un selector existente**.

Al igual que la extensión, la herencia también permite compartir estilos entre diferentes selectores, pero la diferencia radica en cómo se generan los selectores finales en el código CSS compilado.

Cuando utilizas la herencia, el selector original no se incluye directamente en el CSS final. En cambio, los estilos se aplican a los selectores que utilizan la herencia. Esto evita la duplicación de selectores en el CSS compilado.

En resumen, tanto la extensión como la herencia en Sass te permiten compartir y reutilizar estilos entre selectores o reglas. Sin embargo, la extensión copia todos los estilos del selector original en el CSS final, mientras que la herencia aplica los estilos a los selectores heredados sin duplicar el código en el CSS compilado. La elección entre extensión y herencia dependerá de tus necesidades específicas y de cómo deseas estructurar tus estilos en Sass.

Extras

Cada lenguaje que existe trae acompañada consigo una documentación oficial. Esa documentación es el recurso original, por eso queremos dejarte el link para que puedas seguir investigando por tu cuenta y conocer aún más de este lenguaje. 🙌 [Documentación Sass](#)