**Diferencias entre .saas y .scss**

.  
Sass es flexible y **compatible con el CSS tradicional,** por lo tanto, cuando trabajamos con él tenemos la opción de utilizar dos diferentes extensiones  
.

* **.SASS** = Utiliza una sintaxis identada, quitando el uso de { } y el ; tras cada declaración.  
  .
* **.SCSS** = Utiliza sintaxis tradicional de CSS, incluyendo el uso de { } y el ; tras cada declaración  
  .



**Características de SASS**

.

* **Modularización:** Permite dividir los estilos en módulos más pequeños y manejables para simplificar el proceso de desarrollo y mantenimiento.  
  .
* **Variables:** Permite definir variables para almacenar valores reutilizables en un archivo SASS, lo que facilita la creación y el mantenimiento de hojas de estilo.  
  .
* **Mixins:** Permite definir mixins, que son bloques de código reutilizables que se pueden incluir en diferentes partes del CSS. Los mixins pueden contener variables y lógica, lo que facilita la creación de estilos más complejos y dinámicos.  
  .
* **Selectores Anidados:** Se pueden anidar selectores y reglas de estilo, lo que ayuda a organizar y estructurar el CSS de manera más intuitiva.  
  .
* **Herencia:** SASS permite la herencia de estilos entre selectores, lo que puede simplificar la creación y el mantenimiento de hojas de estilo.  
  .
* **Funciones:** Admite funciones que permiten realizar cálculos y operaciones complejas en el CSS. Las funciones se pueden utilizar para definir valores dinámicos, como la altura o el ancho de un elemento en función de otros valores.  
  .
* **Importación:** SASS admite la importación de archivos CSS y SASS, lo que facilita la organización y el mantenimiento de hojas de estilo.  
  .
* **Directivas de control de flujo:** SASS admite directivas de control de flujo, como if/else y for, que permiten realizar operaciones condicionales y repetitivas en el CSS.  
  .
* **Operaciones aritméticas:** SASS admite operaciones aritméticas, lo que permite realizar cálculos matemáticos directamente en el CSS.

Q es pseudocodigo?

El pseudocódigo es una forma de representar código, como algoritmos, funciones y otros procesos, utilizando una combinación de lenguaje natural y elementos similares al lenguaje de programación.

Se llama «pseudocódigo» porque no es realmente ejecutable. En cambio, es una forma de que los humanos comprendan y planifiquen la lógica de la programación — describir los pasos de un programa de forma que sea fácil de entender para los humanos, sin dejar de ser lo suficientemente detallado como para convertirse rápidamente en un lenguaje de programación específico.

Aquí tienes un ejemplo sencillo de pseudocódigo, en el que estamos elaborando la lógica básica para saludar a un visitante por su nombre cuando navega por nuestro sitio o aplicación:

PROCESS GreetUser

INPUT userName

DISPLAY "Hello, " + userName + "!"

END

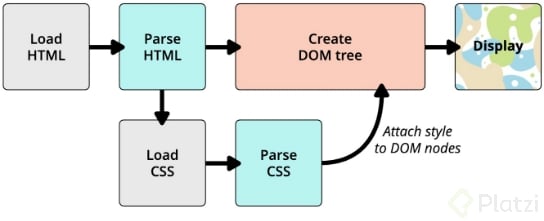
Como puedes ver, el pseudocódigo anterior no está escrito con la sintaxis de ningún lenguaje o framework real. En su lugar, utiliza elementos de lenguaje y programación sencillos y universalmente comprensibles — como PROCESS, DISPLAY, y +  — a modo de sintaxis, lo que lo hace fácil de seguir para cualquiera.

Ese es uno de los poderes de escribir pseudocódigo: Al exponer las intenciones del código en una sintaxis común, puedes saltar todas las barreras de la programación y del lenguaje basado en habilidades. Hablaremos más de esto en la siguiente sección.

# ¿Cómo funcionan los preprocesadores?

Un preprocesador es una herramienta que nos permite escribir pseudocódigo recibiendo como parámetro de entrada los estilos que posteriormente serán convertidos a CSS nativo. Podemos decir que funcionan de manera similar a los traductores pues al indicarle una sintaxis devuelve los valores en una sintaxis nueva.  
En SASS se incluyen características adicionales, como variables, mixins, herencia de clases, entre otras, que hacen que el proceso de escritura de CSS sea más fácil y rápido.

Para poder hacer uso de un preprocesador, primero es necesario entender cómo funcionan los estilos CSS y el navegador. Los estilos CSS son interpretados y representados por el navegador, el cual se encarga de leer el código CSS y aplicarlo a los elementos o componentes HTML de la página. Es el navegador quien recorre la hoja de estilos, línea por línea, y asigna propiedades a los elementos de la página según lo indicado en el código CSS. Este proceso es fundamental para permitir que la página se estilice de la manera deseada, teniendo control sobre el diseño y la disposición de la página, proporcionando al usuario una experiencia visual atractiva, donde todos los elementos están estilizados y presentados de una manera agradable a la vista y fácil de interactuar.



# Ventajas de utilizar un Preprocesador

Los principales beneficios de usar un preprocesador como SASS son la rapidez y la productividad. Permitiendo que el código se pueda escribir de manera mucho más rápida y sencilla, ayudando a los desarrolladores a ahorrar tiempo y ser más productivos. También hacen que el mantenimiento del código sea más fácil, pues todos los estilos se guardan en un solo archivo. La reutilización de código es otro de los beneficios que nos trae el uso de un preprocesador, esto significa que los mismos estilos se pueden aplicar en varias páginas sin tener que escribir el código una y otra vez.  
Finalmente el uso de preprocesadores nos permite que sea mucho más sencillo trabajar una página web de manera responsiva.

# Tipos de Preprocesadores

**[Stylus]**  
[Stylus](https://stylus-lang.com/) es un lenguaje de programación de hojas de estilo en cascada (CSS) que se compila en CSS estándar, está basado en JavaScript. Hay algunas diferencias importantes entre Stylus y SASS. La sintaxis de Stylus es más simple y clara, mientras que la sintaxis de SASS se considera más profesional y compleja. Stylus ofrece una mejor portabilidad y es más fácil de usar. Sin embargo, SASS ofrece mayor soporte al ser utilizado con una mayor cantidad de lenguajes de programación.

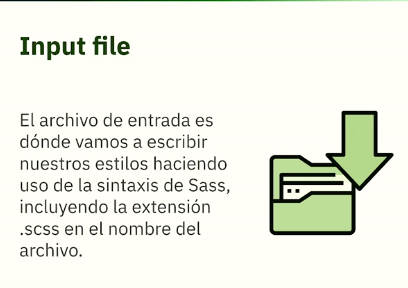
Conoce [Stylus](https://stylus-lang.com/)

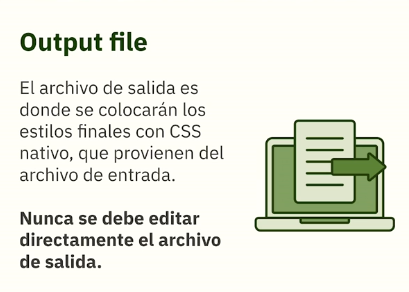
**LESS**  
Una de las principales diferencias entre [LESS](https://lesscss.org/) y Sass es que Sass está codificado en Ruby y, por lo tanto, se procesa del lado del servidor, mientras que Less es una biblioteca de JavaScript (Como Stylus) y se procesa del lado del cliente. Un ejemplo es la forma en que ambos lenguajes manejan las variables es distinta. En LESS, los nombres de las variables se inicializan con @ y en Sass los nombres de las variables se inicializan con el símbolo $.

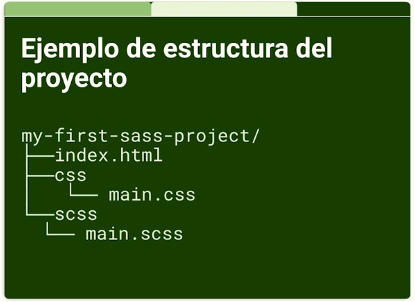
Conoce [LESS](https://lesscss.org/)

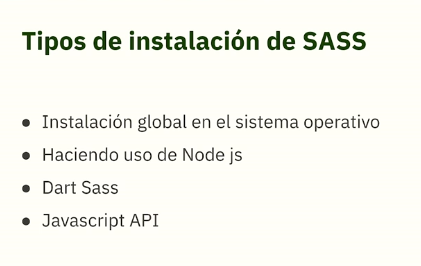
Anatomía de un proyecto en SASS





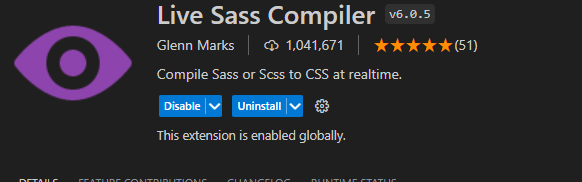
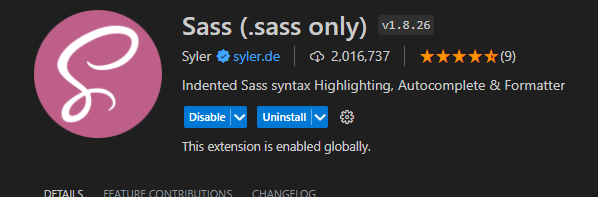






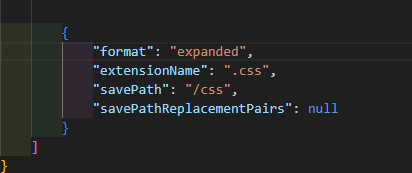


Extensiones para usar sass



**Configuraciones**

Buscar sass format y verificar q la configuración sea esta:

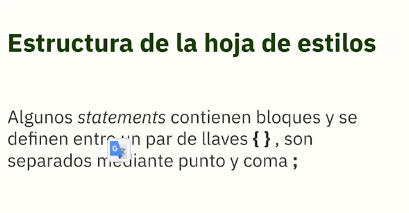


**El proceso de Compilado**

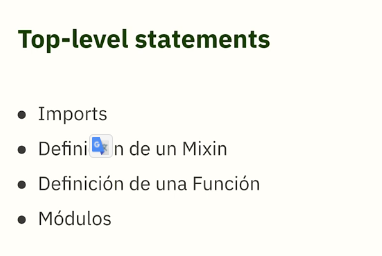
.  
Para hacer uso de SASS dentro de nuestro proyecto tenemos que tener en cuenta **3 puntos importantes que forman parte del proceso de compilación**.  
.

* **Imput file (archivo de entrada) —>** Es donde vamos a escribir nuestros estilos haciendo uso de la sintaxis de Sass, incluyendo la extensión .scss en el nombre del archivo.  
  .
* **Output file (archivo de salida) —>** Es donde se colocarán los estilos finales con CSS nativo, que provienen del archivo de entrada. *(nunca se debe editar directamente el archivo de salida)*  
  .
* **Los comandos para ejecutar y compilar Sass —>** Hay varias formas de ejecutar y compilar Sass, cada una con sus propios comandos y herramientas. La elección del método dependerá del entorno de desarrollo y las preferencias personales del desarrollador.

En React se pueden utilizar archivos scss directamente sin convertirlos en css, desde JavaScript se generan los estilos en automático. Es algo útil.



Las declaraciones que solo se puede usar en la parte superior de la hoja de estilos se conocen como : y son:



Se puede usar en cualquier parte de la hoja de estilos:

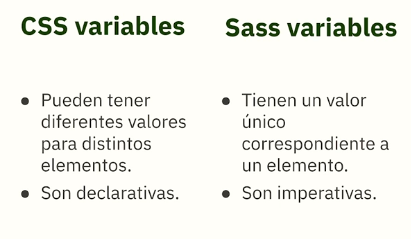


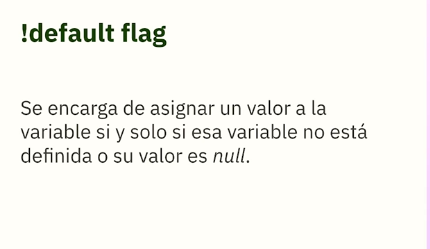
Q es una variables

Es un espacio en memoria y puede almacenar un unico dato. Y en sass van precedidas del símbolo $.



Existen diferencias entre las variables de css y sass





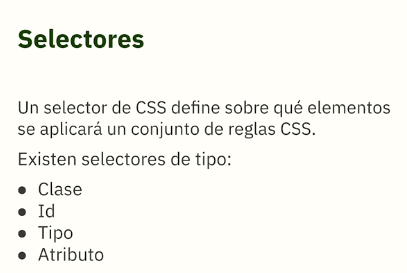
Una recomendación futura para próximos pasos, podría ser el modularizar nuestro código.

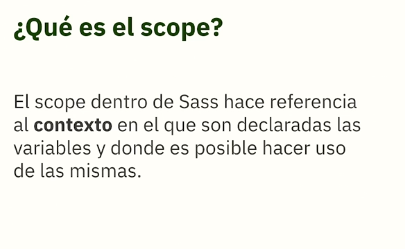
Modularizar es partir o dividir nuestro código en varios archivos, cada archivo se centra en algo muy específico para tener una mejor visión de nuestro proyecto y así facilitar el desarrollo.

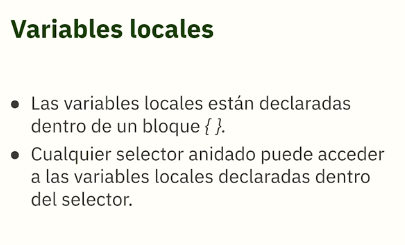
SASS nos da esa facilidad con ayuda de los Partials, les recomiendo consultar la documentación de sass en ese apartado, ya que es de gran ayuda, pero de todas formas lo veremos un poco más adelante en el curso.

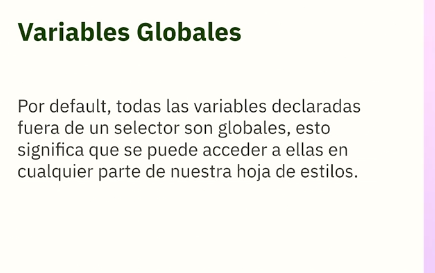
Les recomiendo este [video](https://youtu.be/9Ld-aOKsEDk)

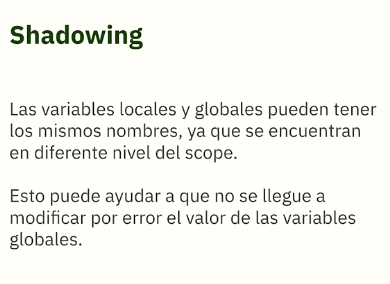
# Uso de selectores, scope de las variables y shadowing

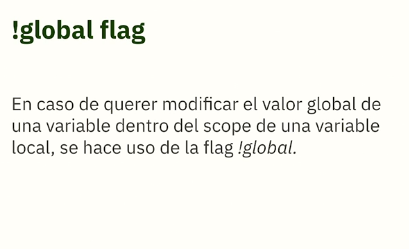












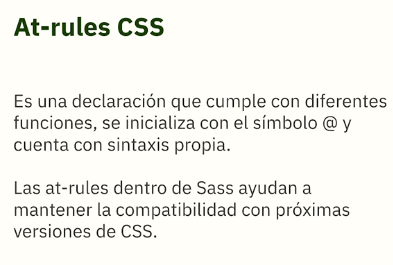
Mi resumen:  
El scope dentro de Sass hace referencia al contexto en el que son declaradas las variables y donde es posible hacer uso de las mismas, hay dos tipos de variables:  
Locales:  
• Declaradas dentro de un bloque { }  
• Cualquier selector anidado puede acceder a las variables locales declaradas dentro del selector  
Globales:  
• Todas las variables declaradas fuera de un selector son globales, esto significa que se puede acceder a ellas en cualquier parte de nuestra hoja de estilos.

Shadowing: Las variables locales y globales pueden tener el mismo nombre ya que se encuentran en diferentes niveles del scope para evitar que se modificquen por error las variables globales.

!global flag: Se usa en caso de querer modificar el valor global de una variable dentro del scope de una variable local.

en el min 6:05 la profe dice que es mas fácil manipular una imagen cuando esta dentro de un div, porque el div es un block e img es inline, esto lo pueden checar en <https://htmlreference.io/>

# At Rules: CSS y nesting

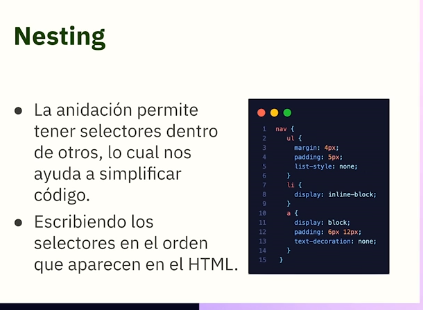




* @use: importa, modulos estilos y funciones de otras hojas de estilos. la diferencia con @import es que import se encarga de hacer los estilos globales.
* @function: permite crear funciones personalizadas, sin embargo Sass cuenta con funciones ya existentes.
* @forward: Recibe como parametro una URL y nos ayuda a cargar los estilos de nuestra hoja de estilos, es muy importante hacer uso de @use para que los modulos esten disponibles en nuestra hoja de estilos.
* @extend: tiene que ver con el concepto de herencia.
* @at-root: se encarga de cargar nuestros estilos en el root del css.
* @include: nos ayuda a invocar los mixins.
* @error, @warn @debug: sirver para cuando hay un error, una advertencia o se quiere debugear, respectivamente
* @for, @if, @each, @while: tienen que ver con estructuras de control, se pueden usar dentro de una función

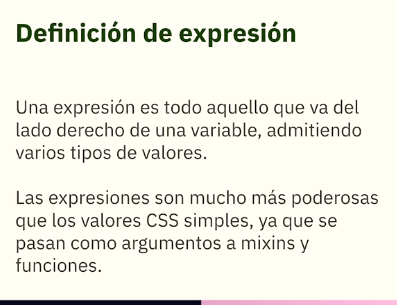


Nesting o anidacion de selectores



Expresiones

Es todo aquello que va del lado derecho de una variable.





**Herencia**

****

**Inheritance Sass**

El concepto de la herencia en Sass, es un concepto bastante sencillo, pero muy poderoso, que si lo sabes aplicar va a acelerar y optimizar tu desarrollo.  
.  
Muchas veces al estar creando nuestros estilos podemos caer en la repetición de código, esto conlleva a dos problemas, Primero se comienza a extender bastante la hoja de estilos y se vuelve difícil de mantener a lo largo del tiempo, el otro problema que puede surgir es el desorden, entonces, si queremos arreglar o cambiar alguna parte de nuestro código va a ser más difícil hacerlo.  
.  
Por suerte tenemos a nuestro buen amigo SASS, piénsalo así, muchas veces usamos [patrones de diseño, responsive](https://carlosazaustre.es/los-5-patrones-del-responsive-design) es patrones, nos dan ciertas guías o convenciones para saber como organizar CSS, el más usado es [BEM](https://getbem.com/).  
.  
BEM nos propone crear estilos generales con una clase y luego modificar esa clase según nos convenga, por ejemplo:

<**div** class="error error--serious">

Oh no! You've been hacked!

</**div**>

En este ejemplo tenemos un div con dos clases “error” y *“error–serious”*, una buena opción para estilizar este div, suponiendo que los necesitáramos reutilizar más adelante, sería de la siguiente forma:  
.

* Estilizamos la clase general con los estilos básicos que se repetirán a lo largo de los div
* Luego estilizamos el div en específico que tendrá otros estilos adicionales

.error {

border: 1px #f00;

background-color: #fdd;

}

.error--serious {

border-width: 3px;

}

Pero esto se puede simplificar a un más usando la [herencia con SASS](https://sass-lang.com/documentation/at-rules/extend)

.error {

border: 1px #f00;

background-color: #fdd;

&--serious {

**@extend** .error;

border-width: 3px;

}

}

De esta manera no solo estamos organizando el código y evitando reutilizar código, si no además se puede entender de una mejor forma de donde viene los estilos heredados y a que clase hacen referencia

La propiedad background-image en CSS se utiliza para establecer una imagen de fondo en un elemento. La propiedad background-size es la que se utiliza para controlar el tamaño de la imagen de fondo.

La opción cover es uno de los valores que puedes utilizar con la propiedad background-size. Cuando se establece background-size: cover, la imagen de fondo se ajustará de manera proporcional para cubrir completamente el área del elemento contenedor. Esto significa que la imagen se expandirá o contraerá según sea necesario para asegurarse de que todo el espacio esté cubierto, aunque parte de la imagen pueda quedar recortada.

Aquí tienes un ejemplo de cómo se utiliza background-image y background-size: cover en CSS:

css  
Copy code  
.elemento {  
background-image: url(“imagen.jpg”);  
background-size: cover;  
}  
En este ejemplo, se establece la imagen de fondo utilizando background-image y se especifica background-size: cover para que la imagen se ajuste y cubra completamente el área del elemento con la imagen proporcionada.

Recuerda que puedes ajustar y personalizar otras propiedades relacionadas con el fondo, como background-position, background-repeat, etc., según tus necesidades específicas.

**Ejemplo:**

        }

    }

    .forniture{

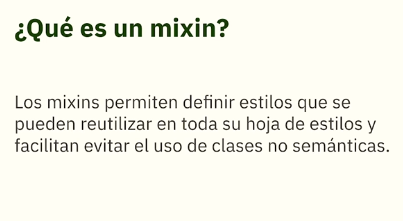
        @extend .healthcare;

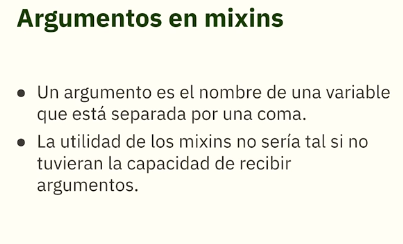
    }

}

La clase forniture estará heredando los estilos de la clase .healthcare

**Qué es un mixin?**

****

****

****

Los @mixen aceptan también parámetros opcionales:

Simplemente, hay que asignarle el valor por defecto ( en mi caso null\* )

**@mixin** flexCenter($direction, $content: null, $align: null){

display: flex;

flex-direction: $direction;

justify-content: $content;

align-items: $align;

}

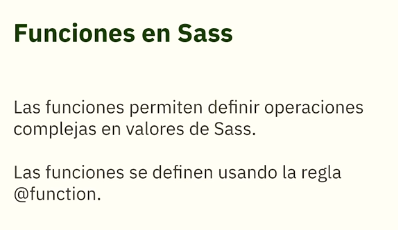
y luego pueden usar, sin especificar ese parámetro:

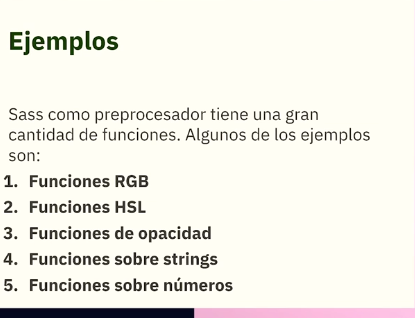
@**include** flexCenter(row, space-between, center);

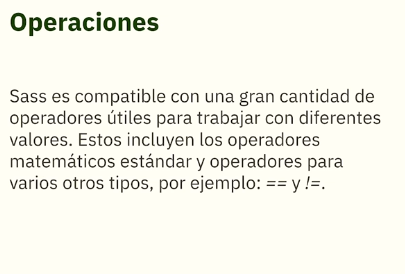
@**include** flexCenter(column, space-around);

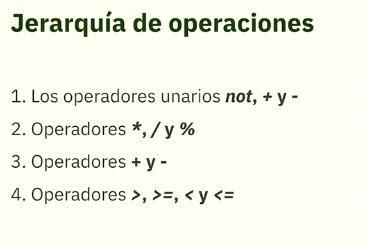
@**include** flexCenter(column);

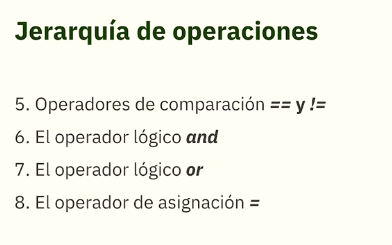
**Funciones**

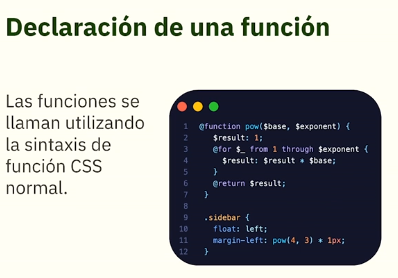
****

****

****

****

****

****