Guía de ejercicios 4 - Sass

 ¡Hola! Te damos la bienvenida a esta nueva guía de ejercicios.

¿En qué consiste esta guía?

En la siguiente guía podrás trabajar los siguientes aprendizajes:

- Utilizar Sass Maister (o codepen) para trabajar con Sass.
- Crear una variable utilizando Sass.
- Utilizar nesting.
- Utilizar el operador ampersand.
- Utilizar @extend para reutilizar estilos.
- Escribir el código en distintas hojas de estilos y unirlas utilizando @import.
- Utilizar Sass desde nuestro computador (puede ser con una aplicación como scout-app o con la línea de commando).
- Modificar bootstrap utilizando Sass.

¡Vamos con todo!

Tabla de contenidos

¡Comencemos!

¿Qué es Sass?

Sass es el acrónimo de "Syntactically Awesome Style Sheets". Esta es una poderosa herramienta que nos ayudará a crear hojas de estilo claras, consistentes, fáciles de mantener.

Sass es considerado un lenguaje de programación a diferencia del lenguaje CSS puro, contando con la posibilidad de crear variables, generar iteraciones, condicionales e incluso la modularización y reutilización del código.

Sintaxis

La forma de escribir estilos con Sass es muy parecida al código CSS, no obstante podemos encontrarnos con formatos y sentencias particulares de este lenguaje. A continuación, veremos cómo crear variables para ver un ejemplo de esta sintaxis.

Utiliza el editor online Sass Meister para poner en práctica los ejemplos de esta lectura.

Variables

Para crear una variable en Sass debemos ocupar el símbolo del peso ($) seguido del nombre de la variable y posteriormente la asignación de su valor.

```
$rojo-oscuro: #b30101;
```

Para ocupar esta variable en la asignación del valor de una propiedad, solo debemos llamarla ocupando el mismo símbolo del peso ($) y su nombre.

```
p{
    color: $rojo-oscuro;
}
```

Si ingresas el código SASS en Sass Meister verás cómo se transforma el código a CSS puro, quedando de la siguiente manera:

```
p {
  color: #b30101;
}
```

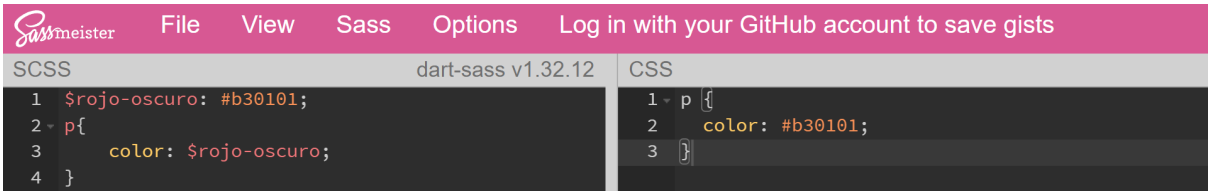

Imagen 1. Variables en Sass.
Fuente: Desafío Latam.

Actividad 1: Variables a la obra

Realiza los siguientes pasos en Sass Meister:

1. Crea una variable primary-color y una variable secondary-color.

2. Crea 2 elementos de texto en el HTML y asígnale una variable a cada uno a través de una clase.

3. Observa el resultado y usa el inspector de elementos para comprobar la compilación de estilos.

Desafío

Latam

Imagen 2. Resultado del uso de variables en Sass.
Fuente: Desafío Latam.

Es importante saber que CSS de forma nativa también permite crear variables, siendo unas de las novedades más destacadas de los últimos años en este lenguaje. Su sintaxis es diferente a la que utiliza Sass, si quieres saber cómo usarlas te recomendamos visitar la documentación oficial.

Nesting

Otra herramienta que nos ofrece Sass es el Nesting, la cual nos permite escribir estilos anidados de una forma más sencilla y coherente que hacerlo en instrucciones separadas. Por ejemplo, si quisiéramos asignar un color de fondo a una lista desordenada y alinear horizontalmente sus items, en CSS haríamos algo como esto

```
ul{
    background: darkgreen;
}

ul li{
    display: inline-block;
    color: white;
}
```

Con el Nesting, podemos asignar anidar los estilos de los elementos según la jerarquía en la que estos estén creados, quedando de la siguiente manera:

```
ul {
  background: darkgreen;
  li {
    display: inline-block;
    color: white;
  }
}
```

Como puedes observar, la anidación consiste en escribir los estilos dentro del bloque de su elemento padre. No obstante es importante que sepas que aunque esta herramienta ofrece - según Sass - un orden más claro del código, esto sigue siendo subjetivo y el lenguaje de todas maneras permite seguir escribiendo los estilos en instrucciones separadas y no anidadas sin errores de ejecución.

Actividad 2: De CSS a SASS

El siguiente código CSS, le asigna un color de fondo a una tabla HTML y un color de texto a las cabeceras de sus columnas.

```
table{
  background: #739c2d;
}

table th {
  color: white;
}
```

Escribe el código SASS equivalente a esta porción de código en Sass Meister para verificar y comparar el resultado.

Actividad 3: Estilos anidados

Utilizando Sass Meister aplica nesting aplicando los siguientes pasos:

1. Crea un Hero Section con una imagen de fondo y un texto en el centro, utiliza siguiente base de html:

```
 <section class="heroSection">

  <p>/* Todos pueden */</p>

</section>
```

2. Crea una variable $primary-color.

3. Utiliza Nesting para asignarle el color al texto centrado del Hero Section.

Imagen 3. Resultado de un ejemplo con Nesting.
Fuente: Desafío Latam.

Selector Parental (&)

El selector parental nos permite hacer el llamado del elemento en el que se están definiendo los estilos, sirviendo como una reutilización del elemento que especificamos en un bloque de estilos. Su símbolo se representa por el ampersand(&) y uno de los usos más típicos es la declaración de pseudoclases.

Por ejemplo, en el siguiente código se le asigna un efecto *hover* a una imagen que cambia el valor de su propiedad *filter:*

```
img {
  filter: blur(3px);
```

```
  transition: filter 500ms;
  &:hover {
    filter: blur(0px);
  }
}
```

Actividad 4: El Ampersand En Sass

Escribe un código SASS que defina un color de fondo a un input y utilice la pseudo clase *:focus* y el ampersand(&) para asignarle un borde negro.

1. Crea un HTML con 3 Input.
2. Asigna un color de fondo y texto a los inputs.
3. Utiliza un ampersand anidado para declarar un borde negro en los inputs a través de una pseudoclase *:focus*.

Imagen 4. Resultado de un ejemplo con el uso de un selector parental.
Fuente: Desafío Latam.

Focus es una de las diferentes pseudoclases que ofrece CSS y se activa en el momento que el navegador enfoca su atención en un elemento. Si quieres saber más sobre esta y otras pseudo clases te invitamos a revisar la documentación oficial.

Extend

Supongamos que tenemos un código CSS que necesitamos ocupar en muchos lugares, por ejemplo, queremos hacer bordes redondeados y un sombreado; y eso deberíamos agregarlo tanto a botones, divs y otros elementos. Con el *@extend* podremos reutilizar diferentes clases que queramos escribir con el objetivo de ser reutilizadas dentro de otros bloques.

Por ejemplo, el siguiente código nos ayudaría a definir un contenedor flexible con contenido centrado horizontal y verticalmente.

```
.flex-center-center{
  display:flex;
  justify-content: center;
  align-items: center;
```

```
}

article{
  background: #739c2d;
  padding: 100px;
  color: white;
  @extend .flex-center-center;
}
```

Actividad 5: Herencia de estilos

Utilizando Sass Maister aplica el @extend para reutilizar los estilos de una clase .grid:

1. Crea un div en el HTML que contenga 4 párrafos.
2. Crea una clase .grid que define una grilla con CSS Grid de 3 columnas.
3. Asigna un color de fondo al div y extiende la clase .grid.

Finalmente, lo que nos permite hacer la instrucción *@extend* de Sass es heredar un bloque de código dentro de otro.

Mixins

Los *mixins* son otra de las características de Sass que nos permite reutilizar código, pero a diferencia de los *extends,* éstos permiten ser modificados. Veamos el siguiente ejemplo en donde se ocupa un mixin llamado *texto-imponente* que nos ayudará asignarle un conjunto de estilos a un párrafo:

```
@mixin texto-imponente ($sombra, $tamano, $color ){
  text-shadow: $sombra;
  font-size: $tamano;
  color: $color;
}

p{
  background: #739c2d;
  padding: 10px;
  display: inline;
  @include texto-imponente(1px 2px 2px black, 30px, white)
}
```

Para probarlo, copia y pega el código en Sass Meister y deberás ver el siguiente resultado:

Imagen 5. Resultado de un ejemplo de Mixins.
Fuente: Desafío Latam.

Cómo se puede apreciar, la forma de ocupar un mixin dentro de un bloque de estilos es a través de *@include* seguido del nombre del mixin.

Los mixins son bloques de estilos dinámicos que pueden ser modificados en el momento que son ocupados. Similares a las funciones en la programación, permiten recibir parámetros y ocuparlos como variables en la asignación de valores o propiedades.

Si quieres aprender más sobre esta poderosa herramienta, consulta y sigue la documentación oficial.

Actividad 6: Mixins Power

Utiliza Sass Meister para crear y ocupar un mixin que defina el ancho y alto de un elemento .

1. Crea 3 Divs con diferentes ids en el HTML.
2. Define un color de fondo para los Divs.
3. Crea un Mixin que reciba el ancho y alto como parámetros de las propiedades width y height.
4. Incluye en cada Div el Mixin creado con valores diferentes.
5. Observa el resultado.

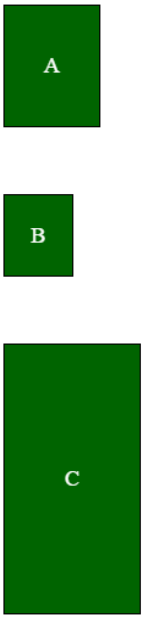

Imagen 6. Otro ejemplo del uso de un Mixin.
Fuente: Desafío Latam.

Además de Sass Meister, también puedes ocupar Codepen para hacer tus pruebas online de Sass. Solo deberás abrir la configuración del editor de CSS y seleccionar entre los preprocesadores el formato SCSS.

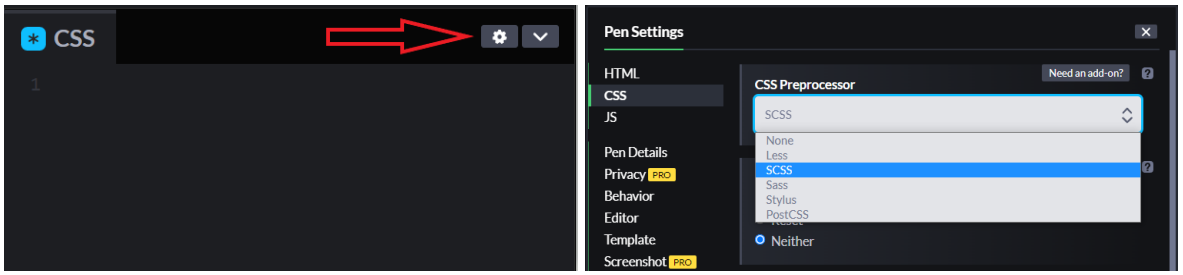

Imagen 7. Configuración de codepen para escribir Sass.
Fuente: Desafío Latam.

Mixin vs extend

Si quieres agregar bordes redondeados (border-radius) de 20 pixeles a varios elementos (divs, imágenes, etc…), ocupas *@extend*.

Si quieres agregar bordes redondeados (border-radius) a varios elementos, pero en el momento de agregarlos quieres poder especificar en qué cantidad, entonces utilizas *@mixin*. Todo dependerá del uso que le dés y la necesidad que tengas en tu desarrollo.

Flujo de trabajo Con SaSS

Cuando trabajamos en nuestro proyecto con Sass no utilizaremos Sass Maister, en lugar de eso, utilizaremos alguna herramienta para convertir los archivos SaSS que creemos o modifiquemos desde nuestro editor de código y luego observaremos los cambios en el archivo css final o con el navegador y el inspector de elementos.

Imagen 8. Flujo de trabajo local.
Fuente: Desafío Latam.

Para transformar los archivos existen distintos tipos de herramientas, algunas con interfaces gráficas como scout-app y otras que se pueden configurar en la línea de comando de forma que se ejecuten automáticamente cada vez que haya cambios en el archivo.

En esta clase utilizaremos la herramienta con interfaz gráfica llamada Scout App, la cuál descargar de su página oficial.

Para probar la compilación de SASS en local con Scout App sigue los siguientes pasos:

1. Crea una carpeta en el escritorio que contenga un index.html y una carpeta assets.
2. Crea una carpeta css y dentro una carpeta scss, en donde almacenaremos un main.scss.
3. Escribe código SASS en el manifiesto creado.

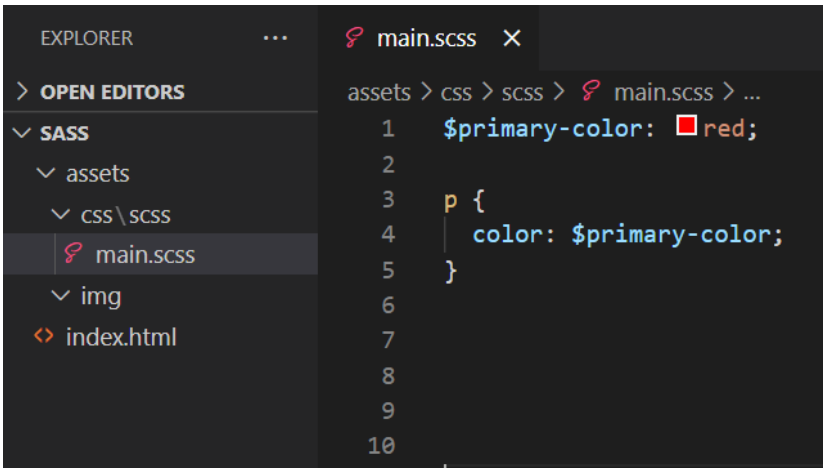

Imagen 9. Ejemplo de un proyecto local con Sass.
Fuente: Desafío Latam.

Ahora, abre Scout App y define la carpeta input y la carpeta output. Estas carpetas representan la ubicación de los archivos de extensión .scss y el directorio en donde se crearán los estilos .css compilados. Estos últimos son los que deben estar enlazados al HTML

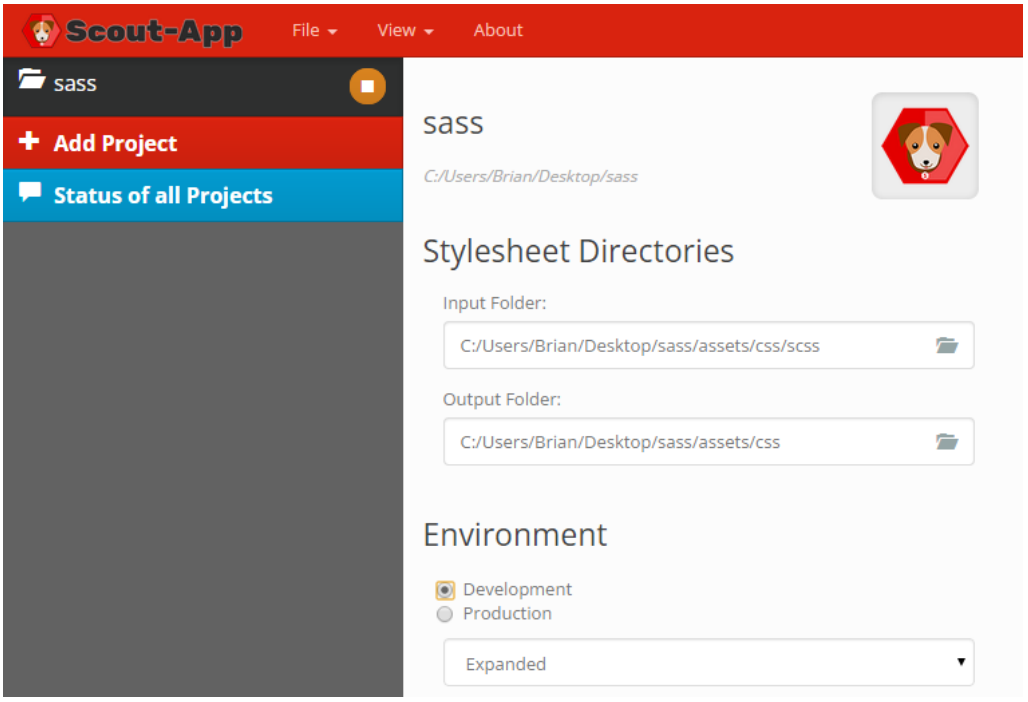

Imagen 10. Interfaz de Scout app.
Fuente: Desafío Latam.

Apenas ejecutes el preprocesador en el proyecto seleccionado, se generará un nuevo archivo con extensión *.css* y de nombre igual a el o los archivos de extensión *.scss* de la carpeta input seleccionada.

Cabe destacar que tu documento HTML debe estar enlazado al archivo generado de extensión CSS.

Import

Sass tiene la instrucción *@import* con la que podemos separar nuestros estilos en múltiples archivos separados y Sass se encargará de juntarlos y compilarlos en un único CSS final.

A continuación, un ejemplo de cómo importar las variables escritas en un archivo diferente al principal o manifiesto.

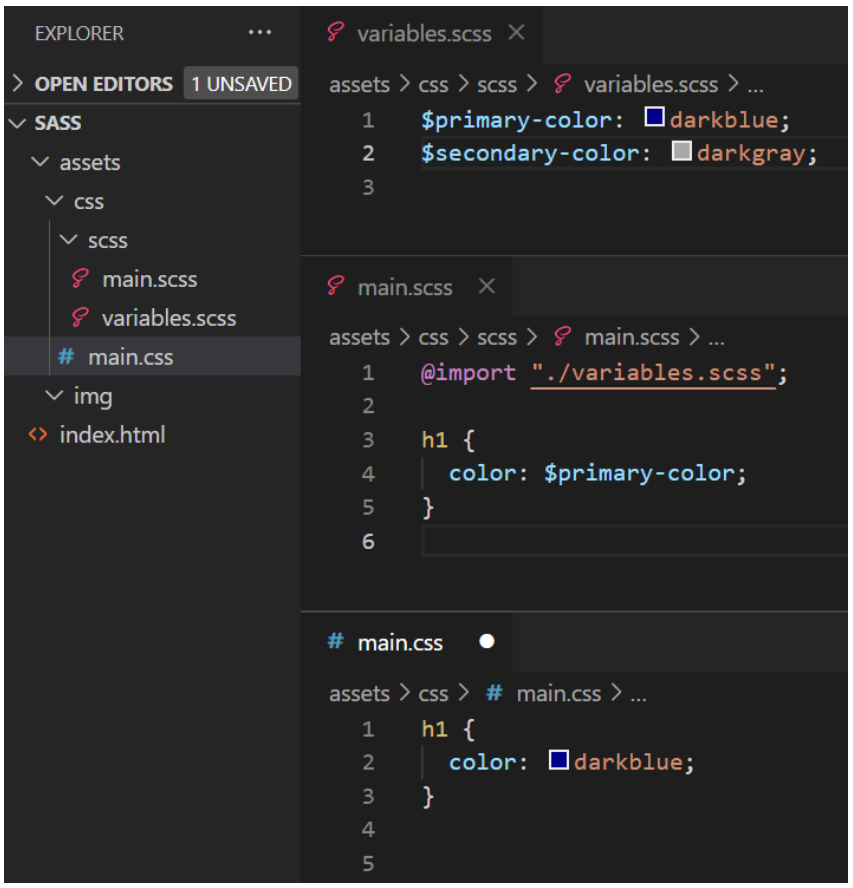

Imagen 11. Ejemplo del uso de imports en Sass.
Fuente: Desafío Latam.

En este ejemplo estamos mostrando 3 archivos:

- El archivo *variables.scss* tiene definidas 2 variables con variantes de colores.
- El archivo *main.scss* importa las variables y ocupa una de ellas en una etiqueta h1.
- El 3º archivo muestra el resultado después de la transformación a código CSS.

Actividad 7: Imports a la obra

Pongamos en práctica los *@imports* con los estilos de 2 componentes que utilizan variables para la asignación de colores de fondo y de texto.

Sigue los siguientes pasos:

1. Crea un nuevo proyecto con un archivo *index.html* y una carpeta *assets* que dentro contenga las carpetas *css* y *scss*.
2. Dentro de la carpeta *scss* crea 3 archivos con los siguientes nombres:
    a. *variables.scss*

   b. *componentes.scss*
   c. *main.scss*
3. En el HTML, agrega una base con el autocompletado y en el body agrega las etiquetas *nav* y *footer* con un texto de prueba dentro.
4. En *variables.scss crea 2 variables color-primario y color-secundario.*
5. En *componentes.scss* selecciona las etiquetas *nav* y *footer* para asignar un *background* y un *color* ocupando las variables creadas anteriormente.
6. Importa las variables y los componentes al *main.scss*
7. Preprocesa el manifiesto Sass con Scout App y observa el resultado.

Organizando el proyecto

Ahora que sabemos que podemos distribuir los estilos en diferentes archivos, en la práctica tarde o temprano nos encontraremos en la necesidad de agrupar por categorías las definiciones estéticas que vayamos definiendo. Para hacer esta tarea de una forma más ordenada, existen estructuras de carpetas predefinidas como el patrón 7-1 (7 carpetas, 1 manifiesto) que nos permiten ordenar un proyecto grande utilizando SASS. Algunos proyectos ocupan otros patrones pero la estructura es relativamente similar.

Estas distribuciones de archivos trabajan de la mano con la sentencia *@import*, con la que podremos modular e importar los estilos que queramos agrupar por categoría. El manifiesto se encarga de importar todos, o, la mayoría de los archivos

```
sass/
|
|- abstracts/
|   |- _variables.scss    # Sass Variables
|   |- _functions.scss    # Sass Functions
|   ...                       # etc..
|
|- base/
|   |- _reset.scss        # Reset/normalize
|   |- _typography.scss   # Typography rules
|   ...                     # Etc.
|
|- components/
|   |- _buttons.scss      # Buttons
|   |- _carousel.scss     # Carousel
|   ...                     # Etc.
|
|- layout/
|   |- _navigation.scss   # Navigation
|   |- _grid.scss         # Grid system
|   ...                     # Etc.
|
|- pages/
|   |- _home.scss         # Home specific styles
|   |- _contact.scss      # Contact specific styles
|   ...                     # Etc.
|
|- themes/
|   |- _theme.scss        # Default theme
|   |- _admin.scss        # Admin theme
|   ...                     # Etc.
```

Imagen 12. Ejemplo del uso de imports en Sass.
Fuente: Desafío Latam.

En la práctica, todos los proyectos son diferentes y la estructura que decidamos crear será en función de la exigencia y magnitud del proyecto. No es obligatorio crear todas las carpetas si éstas no serán utilizadas, por ejemplo Bootstrap ocupa solo algunas carpetas para distribuir todos sus estilos, y a continuación profundizaremos en la estructura de Bootstrap Sass para entender cómo se organiza un proyecto grande en sass.

Bootstrap Sass

El framework Bootstrap utiliza una distribución de archivos similar a la del patrón 7-1. Esto lo podemos observar descargando el código fuente en la página oficial y abriendo la carpeta SCSS en nuestro editor de códigos.

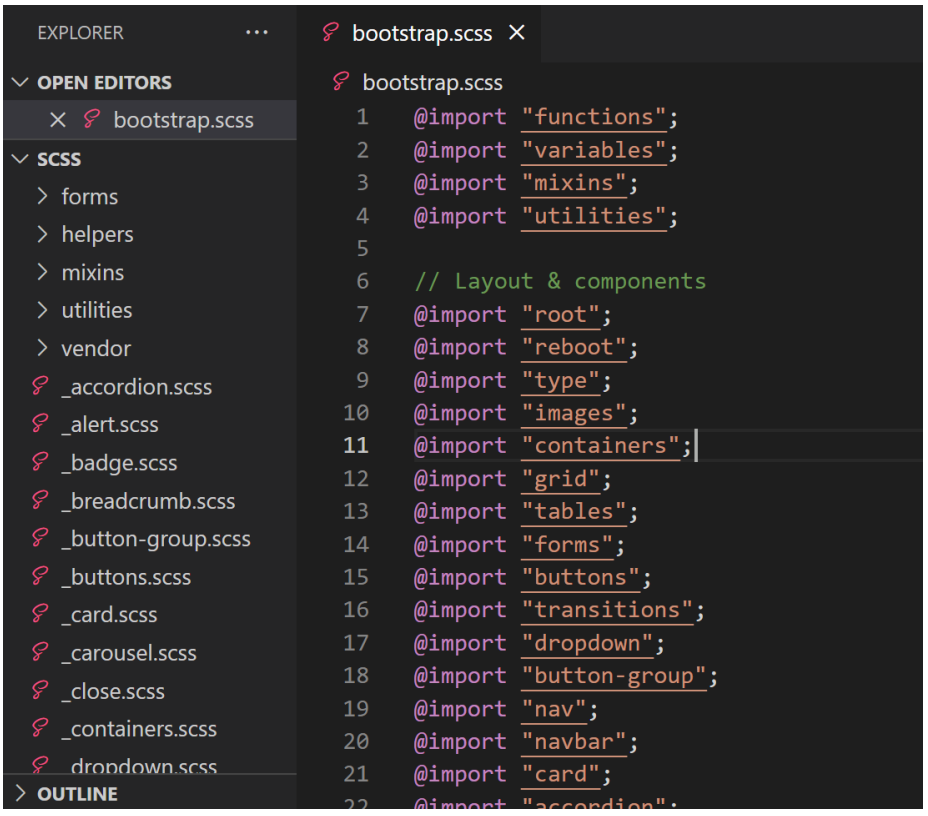

Imagen 13. Bootstrap Sass.
Fuente: Desafío Latam.

Ahora que sabemos ocupar SASS, podemos incluso modificar los valores predeterminados por Bootstrap en su código fuente y ajustarlo a nuestras necesidades. Por ejemplo, cambiar los colores de sus variantes:

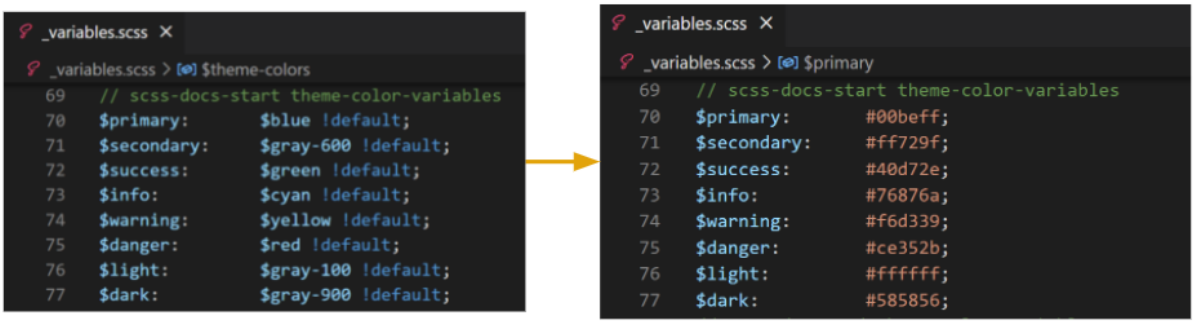

Imagen 14. Modificación de variables en Bootstrap Sass.
Fuente: Desafío Latam

Actividad 8: Modificando variables de Bootstrap Sass.

Crea un sitio web que utilice Bootstrap Sass para la renderización de una Card usando sus variantes con colores personalizados:

1. Crea un sitio web con la estructura básica.
2. Descarga los archivos SASS de Bootstrap.
3. Utiliza Scout App para preprocesar los archivos SASS.
4. Agrega el archivo style.css generado al HTML.
5. Prueba que funcione Bootstrap.
6. Modifica dentro del Sass el color del background-primario.
7. Agrega un botón con el background-primario.
8. Vuelve a preprocesar.
9. Observa los cambios.

Agregando un nuevo tema de Bootstrap Sass

Como bien sabemos, Bootstrap utiliza diferentes variantes de colores para dirigir semánticamente diferentes tonos y paleta de colores. Estos son determinados por el framework como *themes* y los puedes encontrar en su archivo *_variables.scss*

```
$theme-colors: (
  "primary":    $primary,
  "secondary":  $secondary,
  "success":    $success,
  "info":       $info,
  "warning":    $warning,
  "danger":     $danger,
  "light":      $light,
  "dark":       $dark,
) !default;
```

Imagen 15. Temas de colores en Bootstrap.
Fuente: Desafío Latam.

Si quisiéramos agregar un nuevo tema solo tendríamos que seguir el mismo formato agregando otro argumento con el nombre de nuestra nueva variante y la variable que ocupará como valor. A continuación se muestra un ejemplo de cómo sería agregar un tema *purple*.

```
$theme-colors: (
  "primary":    $primary,
  "secondary":  $secondary,
  "success":    $success,
  "info":       $info,
  "warning":    $warning,
  "danger":     $danger,
  "light":      $light,
  "dark":       $dark,
  "purple":     $purple
) !default;
```

Imagen 16. Nuevo tema agregado a Bootstrap Sass.
Fuente: Desafío Latam.

La variable *$purple* en este ejemplo se está ocupando como referencia de este nuevo tema, y esto logrará que puedas asignar las populares clases de Bootstrap como *.bg-dark* o *.text-light* pero ahora también *.bg-purple* o *.text-purple.* De esta misma manera puedes crear o modificar todos los temas que desees.

Existen diferentes modificaciones de Bootstrap disponibles en internet de forma gratuita a través del sitio de Bootswatch, que básicamente replica el mismo proceso que estamos haciendo en esta lectura para generar plantillas personalizadas del framework.

Resumen

- Sass incluye varias herramientas que nos permiten expandir las capacidades de CSS, entre ellas tenemos el uso de variables, nesting, mixings y extends e includes.
- Dentro de las especificaciones de CSS ya se han incorporado algunas de estas herramientas, por ejemplo hoy CSS soporta variables, sin embargo todavía no tiene soporte para las otras mencionadas.
- El Nesting nos permite simplificar y aligerar la interpretación del código escribiendo los estilos de los elementos de forma animada y siguiendo su jerarquía, no obstante no es obligatorio hacerlo de esta manera.
- Sass nos ofrece la posibilidad de extender o reutilizar bloques de estilos predeterminados usando el @extend, reduciendo la redundancia en nuestr
- Con los Mixins podemos reutilizar y modificar un mismo conjunto de estilos en función del elemento en el que lo apliquemos. Es una herramienta útil cuándo buscamos mantener una misma identidad en nuestro sitio web.
- Con Sass podemos escribir los estilos de nuestros sitios web en diferentes archivos y ocupar el *@import* para unificar todo en un solo manifiesto.
- Para preprocesar el código Sass de forma local podemos ocupar un software como **Scout App** o bien ocupar la terminal y el paquete *sass* de NPM para ejecutar un compilador atento a los cambios de nuestro archivo principal.
- Para conseguir un código más ordenado, en Sass podemos ocupar la sentencia *@import* para unificar los estilos que separamos y categorizamos en diferentes archivos y carpetas.
- Bootstrap también ocupa Sass y con lo aprendido en esta unidad ahora somos capaces de modificar los valores de sus variables y modificar o agregar un tema nuevo para la asignación de colores.

Para profundizar después del desafío

Utilizando Sass en la línea de comando

Es posible también preprocesar el código Sass que escribimos a través de la terminal, no obstante para esto es necesario tener instalado Node Js e instalar el paquete *sass*.

```
npm i -g sass
```

Posteriormente, podríamos ubicarnos desde la terminal en la carpeta donde alojamos el manifiesto Sass y ocupar la siguiente línea de comando:

```
sass <input.scss>:<output.css>
```

Si quieres aprender más sobre esta manera, visita su apartado en la documentación oficial de Sass.

Hasta ahora hemos hablado de que modificar los archivos Sass, el término técnico es preprocesar. Las aplicaciones que realizan esa acción se llaman preprocesadores.

Existen otros preprocesadores CSS además de Sass, existen por ejemplo Less, Stylus y PostCSS, no es necesario que los domines todos, es mejor manejar bien uno y SaSS hoy es el más utilizado.