Metodologías para la organización y el preprocesador Sass

Sass y variables

{desafío} latam_

Implementar una interfaz de usuario web utilizando buenas prácticas en el manejo de estilos para brindar un aspecto visual e interacciones acordes a lo requerido

- Unidad 1:
  Metodologías para la organización y el preprocesador Sass
- Unidad 2:
  El modelo de cajas y el Layout
- Unidad 3:
  Utilizando Bootstrap como Framework CSS

Te encuentras aquí

{desafío} latam_

¿Qué aprenderás en esta sesión?

- *Utiliza variables para la reutilización de código CSS.*

¿Cómo definimos el concepto de variable?

{desafío} latam_

¿Qué valores de CSS podríamos guardar en una variable?

{desafío} latam_

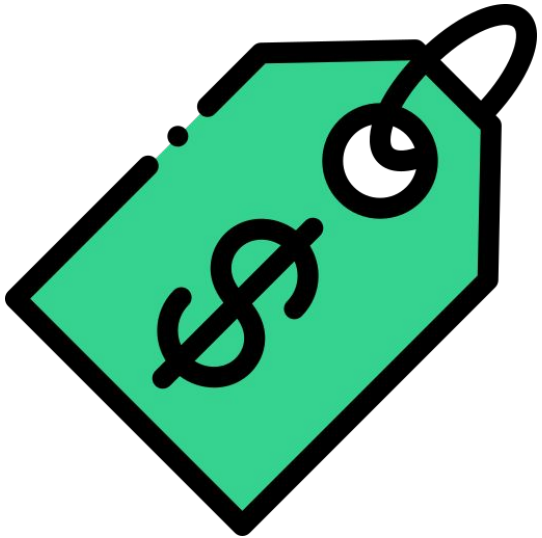
/* Uso de variables para reutilización de código */

¿Qué es una variable?

Una variable es un espacio en memoria con un nombre o alias representativo en dónde guardaremos algún valor para poder utilizarlo después.

Algunos ejemplos de variables son:

- El nombre de un usuario.
- La marca de un automóvil.
- Una contraseña.
- El precio de un producto.
- Entre otras.



Uso de variables para reutilización de código

Para utilizar variables estas primero se deben declarar utilizando el signo dólar ($) seguido por el nombre de la variable, luego le definimos un valor al igual que en una propiedad CSS.
Y para usarlas, debemos llamarlas como valor en una propiedad.

```scss
// SCSS

//Variables
$poppins-font: 'Poppins', sans-serif;
$primary-color: #171717;

body {
    font-family: $poppins-font;
    color: $primary-color;
}
```

```css
/* CSS */

body {
    font-family: 'Poppins', sans-serif;
    color: #171717;
}
```

/* Elementos anidados y namespaces */

Nesting

HTML trabaja en base a jerarquía de elementos, donde sus etiquetas se organizan en forma de árbol, gracias a esto, facilita la lectura del código HTML permitiendo saber con más claridad cuáles elementos se encuentran dentro de otro.

En cambio en CSS, el código es más plano y muchas veces es necesario repetir selectores para aplicar reglas a elementos que se encuentran dentro de otros.

```
.post h1 {
    font-weight: bold;
    color: #171717;
}

.post p {
    line-height: 1.5em;
}

.post img {
    width: 200px;
}
```

Nesting

Cómo vimos en el ejemplo anterior, la clase .post se repite en 3 reglas diferentes.

Con Sass podemos evitar la repetición de código anidando los elementos hijos de la clase .post

```
.post {
    h1 {
        font-weight: bold;
        color: #171717;
    }
    p {
        line-height: 1.5em;
    }
    img{
        width: 200px;
    }
}
```

{desafío} latam_

Nesting

En el caso que el elemento padre tenga sus propios estilos, estos se pueden aplicar sin ningún problema, manteniendo el orden y la legibilidad del código.

Esto nos permitirá mantener mucho más organizado el código facilitando su lectura.

```
.post {
    padding: 20px;
    border: 1px solid #ccc;
    h1 {
        font-weight: bold;
        color: #171717;
    }
    p {
        line-height: 1.5em;
    }
    img{
        width: 200px;
    }
}
```

Sass y BEM

La anidación de Sass se lleva muy bien con la metodología BEM, ya que podemos anidar los elementos usando `&__elemento`.
Veamos un ejemplo de un bloque que contiene 2 elementos en su interior:

- logo__img.
- logo__text.

```
<div class="logo">
  <img class="logo__img" src="assets/img/adl.png" alt="Logo Blog FrontEnd" />
  <span class="logo__text">Blog FrontEnd</span>
</div>
```

Elementos

Este sería el resultado de anidar usando esta sintaxis.

```scss
.logo {
    display: flex;
    align-items: center;
    width: 100%;

    &__img {
        width: 55px;
        height: 55px;
        border-radius: 10px;
    }

    &__text {
        font-size: 24px;
        font-weight: 700;
        margin-left: 20px;
    }
}
```

Modificadores

Los modificadores también se pueden anidar de esta manera, sin embargo en estos se deben usar los dos guiones -- y luego el modificador.

```
.form-login__button {
    width: 100%;
    margin: 5px 0;
    background-color: #729E2E;
    height: 40px;
    border-radius: 5px;
    border: 1px solid #ccc;
    color: #fff;
    font-weight: bold;
    font-size: 20px;

    &--google {
        display: flex;
        justify-content: center;
        align-items: center;
        background-color: #fff;
        color: #171717;
        margin-bottom: 10px
    }
}
```

Namespaces

Namespaces o espacio de nombre, se puede definir como un grupo de elementos relacionados que tienen un nombre o identificador único.

Trabajar con espacios de nombre permite evitar ambigüedades de identificadores o clases que tengan el mismo nombre en diferentes archivos.

Algunos ejemplos de espacios de nombres son:

- Archivos.
- Funciones.
- Mixin.

/* Manejo de parciales e imports */

Parciales

Uno de los conceptos más importantes a la hora de trabajar con SASS son los parciales, los que se pueden definir como “fragmentos de códigos” que nos permiten crear módulos que podemos reutilizar una y otra vez.

Ya creamos parciales antes al momento de organizar el proyecto según el patrón 7-1, algunos de ellos fue `_header.scss` y `_buttons.scss`.

Parciales

¿Cómo nombramos los parciales?

A la hora de nombrar estos archivos o parciales, **es importante** anteponer un guión bajo al nombre.

- _buttons.scss

Este guión bajo le informa al compilador que este archivo no necesita convertirse en un archivo CSS, de lo contrario, todos nuestros parciales se convertirían en múltiples archivos CSS.

Import

Cuando utilizamos el patrón 7-1 trabajamos con un manifiesto en el cual importamos todos los demás parciales para utilizar sus estilos con la directiva `@import`.

Al hacer uso de @import podemos acceder a las variables, funciones o mixin de otros parciales.

Import

Para importar un archivo se debe utilizar la directiva @import y el nombre del parcial o la ruta en caso que se encuentre ubicado en diferentes directorios.

```scss
// Importar parcial header ubicado en el mismo directorio
@import 'header';

// Importar parciales dentro del directorio components
@import 'components/buttons';

// Importar múltiples parcuales dentro un mismo directorio
@import
    'base/reset',
    'base/base',
    'base/typography';
```

{desafío} latam_

No es necesario incluir el guión bajo del nombre del parcial, ni su extensión.

Import

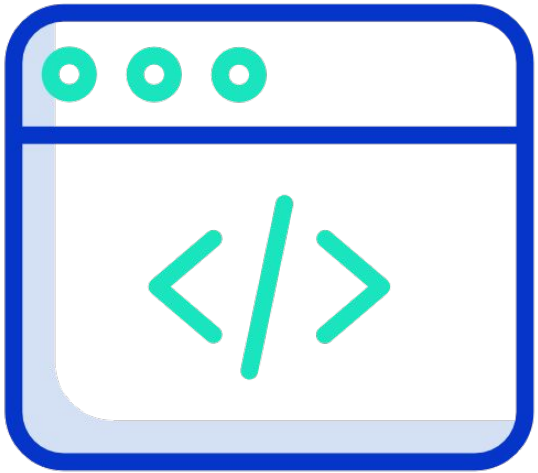
Algo a tener en cuenta con la regla @import es que Sass lo eliminará gradualmente durante los próximos años, y Sass en su documentación oficial, aconseja usar en su lugar la regla **@use**.

¿Qué tiene de malo @import?

- @import hace que todas las variables, mixin o funciones sean accesibles globalmente, esto hace que sea más difícil encontrar dónde se define un valor.
- Lo mismo ocurre con el uso de @extends, ya que esta regla también es global, es difícil predecir cuáles reglas de estilos se extenderán.
- No había forma de definir miembros privados para quitarles el acceso a hojas de estilos que no queremos.

Para conocer más detalles acerca de esto, recomendamos visitar la documentación oficial.

Use



La regla @use ayuda a solucionar los inconvenientes que presenta la regla @import ya que permite seleccionar solo los miembros (funciones, mixin, variables) que queramos usar, y no necesariamente importar todos los estilos de algún archivo.

Esta regla carga otro archivo Sass como un módulo, lo que significa que puede hacer referencia a reglas de estilos provenientes de otro archivo con un espacio de nombres basado en el nombre del archivo.

Use

En este ejemplo usamos **@use 'base';** en el archivo styles.scss.
Su uso es bastante similar al uso de @import, salvo que en el ejemplo anteponemos el nombre del archivo como namespace que queremos usar, y luego la variable, para hacer referencia sólo al archivo base, esto evitará usar alguna otra variable $primary-color que se encuentre en otro archivo..

```scss
// _base.scss

$font-stack: Helvetica, sans-serif;
$primary-color: #333;

body {
    font: 100% $font-stack;
    color: $primary-color;
}
```

```scss
// styles.scss
@use 'base';

.inverse {
    background-color: base.$primary-color;
    color: white;
}
```

{desafío} latam_

/* Manejo de mixins e includes */

Mixin

Los mixin en Sass, permiten hacer grupos de declaraciones CSS que se desea reutilizar en el sitio, estos pueden contener reglas completas, incluso se pueden pasar argumentos al igual que las funciones.

La regla de oro es que si se detecta un grupo de propiedades CSS que están siempre juntas por alguna razón (es decir, no es una coincidencia), puedes crear un mixin en su lugar.

Mixin

Un ejemplo de su uso es un mixin para darle un color, tamaño y peso a un texto, pero debe tener un background color distinto de acuerdo a la sección en la que se encuentre.

No solo hace que el código sea más fácil de escribir, sino que también será más fácil de leer.

```scss
@mixin important-text {
    color: red;
    font-size: 25px;
    font-weight: bold;
}

.danger {
    @include important-text;
    background-color: green;
}

.black {
    @include important-text;
    background-color: black;
}
```

Include

La directiva `@include`, como vimos en el ejemplo anterior, sirve para incluir un mixin dentro de otras reglas de estilos, evitando la duplicidad de código.

Es posible usar @include dentro de un mixin, sin necesidad de incluir 2 mixin, tal como se muestra en este ejemplo de la documentación de Sass.

```
@mixin reset-list {
  margin: 0;
  padding: 0;
  list-style: none;
}

@mixin horizontal-list {
  @include reset-list;

  li {
    display: inline-block;
    margin: {
      left: -2px;
      right: 2em;
    }
  }
}

nav ul {
  @include horizontal-list;
}
```

Fuente: sass-lang.

¿Por qué deberíamos trabajar con la regla @use en lugar de @import?

Resumen

- Una **variable** es un espacio en memoria con un nombre o alias representativo en dónde guardaremos algún valor para poder utilizarlo después.

- La **anidación de Sass** se lleva muy bien con la metodología BEM, ya que podemos anidar los elementos usando &__elemento.

- La regla **@use** ayuda a solucionar los inconvenientes que presenta la regla @import ya que permite seleccionar solo los miembros (funciones, mixin, variables) que queramos usar, y no necesariamente importar todos los estilos de algún archivo.

- Los **mixin** en Sass, permiten hacer grupos de declaraciones CSS que se desea reutilizar en el sitio, estos pueden contener reglas completas, incluso se pueden pasar argumentos al igual que las funciones.

Próxima sesión...

Guía de ejercicios

{desafío}
latam_

{desafío}
latam_

Academia de
talentos digitales