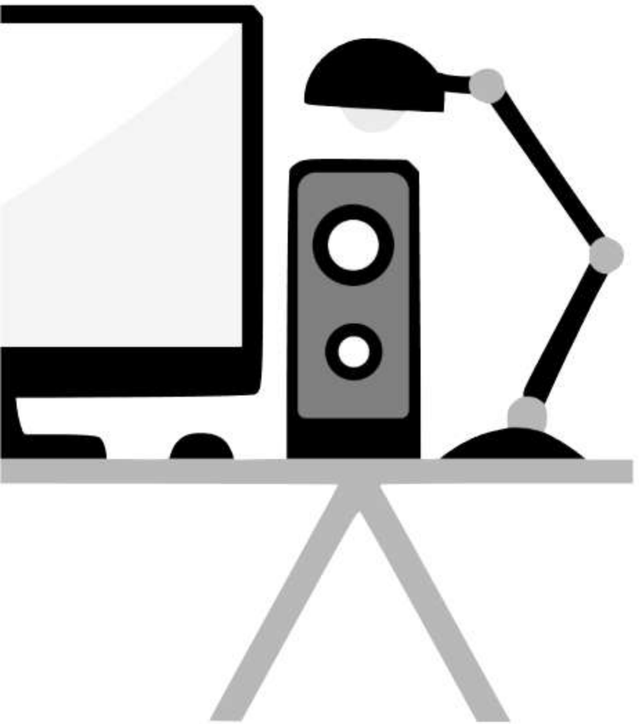
DESARROLLO DE LA INTERFAZ DE USUARIO WEB

Lección 03

sustantiva
CON•SENTIDO

Contenidos



1. El preprocesador SASS
2. Qué es Sass y por qué utilizarlo
3. Conocer Sass y aprovechar las ventajas en el proceso de construcción de un sitio web.
4. Instalar Sass y linters para editores de código
5. Conocer patrón 7-1
6. Conocer sintaxis, flujo de trabajo y buenas prácticas usando Sass
7. Uso de variables para reutilización de código.
8. Elementos anidados y namespaces.
9. Manejo de parciales e imports
10. Manejo de mixins e includes

sustantiva
CON•SENTIDO

El preprocesador SASS

Sass es un metalenguaje de Hojas de Estilo en Cascada (CSS). Es un lenguaje de script que es traducido a CSS, SassScript es el lenguaje de script en sí mismo. Sass consiste en dos sintaxis. La sintaxis original, llamada indented syntax que usa una sintaxis similar al Haml. Este usa la sangría para separar bloques de código y el carácter nueva línea para separar reglas.

La sintaxis más reciente, SCSS, usa el formato de bloques como CSS. Este usa llaves para denotar bloques de código y punto y coma (;) para separar las líneas dentro de un bloque.

La sintaxis indentada y los ficheros SCSS tienen las extensiones .sass y .scss respectivamente.

sustantiva
CON•SENTIDO

Ventajas de SASS

El uso de una herramienta como Sass proporciona una serie de ventajas, como son las siguientes:

- Reduce el tiempo para crear y mantener el CSS.
- Permite tener una organización modular de los estilos, lo cual es vital para proyectos grandes.
- Proporciona estructuras avanzadas propias de los lenguajes de programación, como variables, listas, funciones y estructuras de control.
- Permite generar distintos tipos de salida, comprimida, normal o minimizada, trabajando tanto en desarrollo como en producción, además se hace una forma muy fácil.

sustantiva
CON•SENTIDO

Ventajas de SASS

- Permite vigilar los ficheros, de tal manera que, si ha habido un cambio en la hoja de estilos, se regenera automáticamente (modo watch).

- Tiene muy pocas dependencias, sobre todo la nueva versión, que es dart-sass. En las anteriores versiones se dependía de muchas librerías de Ruby y era un poco engorroso de instalar, pero con la nueva versión, la instalación es muy fácil.

- Existen muchas herramientas asociadas, muchas librerías hechas con Sass y una comunidad muy importante de usuarios.

sustantiva
CON•SENTIDO

Instalar SASS

Si usas Vscode, enhorabuena, VsCode tiene soporte en el mismo editor para .scss, o sea sass. Por ello, no es necesario instalar nada para comenzar a usarlo.

Pese a ello, hay extensiones útiles que se pueden utilizar, como es scss intellisense.

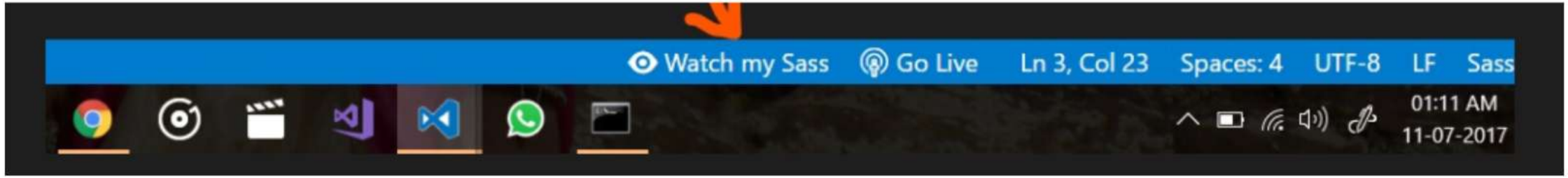
En caso de no estar utilizando vscode, se puede usar sass, mediante el gestor de paquetes npm, se puede hacer de forma global con:

```
npm install -g sass
```

Así de rápido y sencillo

Instalar SASS

Comenzar a usar Sass es muy fácil, solo crea un archivo .scss, luego verás que al seleccionarlo aparece en la barra inferior de vscode la opción watch my sass, dale click y vscode comenzará a compilar los cambios y creará un archivo css para que puedas utilizarlo



Las configuraciones por defecto son bastante buenas, pero si quieres utilizarlo de otra forma, puedes cambiar el comportamiento en Vscode.

sustantiva
CON•SENTIDO

Patrón 7-1

Ahora que ya estamos trabajando con sass, toca conocer el **patrón 7–1, el cual** es una estructura ampliamente estandarizada como forma de organizar proyectos complejos. Consiste en tener todas los *partials* organizados en 7 carpetas y un único archivo en la raíz para gestionar todos los *imports* necesarios.

```
sass/
|
|- utilities/
|   |- _variables.sass
|   |- _functions.sass
|   |- _mixins.sass
|
|- base/
|   |- _reset.sass
|   |- _typo.sass   // Ti
|
|- components
|   |- _buttons.acss
|   |- _jumbo.sass     //
|
|- layout/
|   |- _navigation.sass
|   |- _grid.sass
|   |- _header.sass
|   |- _footer.sass
|   |- _forms.sass
|
|- views/
|   |- _home.sass
|   |- _contact.sass
|
|- themes/
|   |- _theme.sass
|   |- _admin.sass
|
|- vendors/
|   |- _bootstrap.sass
|   |- _jquery-ui.sass
|
|- main.sass
```

sustantiva
CON•SENTIDO

Patrón 7-1

- **utilities** En esta carpeta se encuentran las funciones, variables y *mixins* que emplearemos en el resto de ficheros. Estos archivos no generan ninguna regla CSS al ser compilados sino que añaden funcionalidad extra para ser empleada por los otros *partials.*
- **base .** Contiene el código base de nuestro proyecto como es las reglas de tipografía (y fuentes adicionales) así como los resets necesarios para sobrescribir las reglas por defecto de los navegadores.
- **components .** Contiene los estilos de los distintos componentes que emplearemos dentro de nuestra aplicación como pueden ser botones, jumbos, sliders, carruseles… Separar cada componente en su propio archivo nos permitirá reutilizarlos en otros proyectos.
- **layout .** En esta carpeta aparecen los estilos relacionados con la estructura de nuestra página web, como por ejemplo el sistema de *grid* que estemos empleando así como los estilos de la cabecera, el footer…
- **views .** Contiene los estilos específicos para determinadas vistas (páginas) de nuestro proyecto como puede ser la página de inicio o la de contacto.
- **themes .** Seguramente esta sea la carpeta que más os ha llamado la atención ya que su uso no es muy común. Su razón de ser es la de almacenar los estilos referidos a diferentes *themes* que pueda adoptar nuestro proyecto en función, por ejemplo, del tipo de usuario o la sección que esté visualizándose.
- **vendors .** Acá van librerías de terceros.

sustantiva
CON•SENTIDO

Patrón 7-1

El archivo **main.sass** , situado en la raíz del proyecto, importará todos los archivos situados dentro de cada carpeta sin contener código sass adicional. Por ejemplo:

```
@import 'utilities/variables';
@import 'utilities/functions';
@import 'utilities/mixins';

@import 'utilities/bootstrap';
@import 'vendors/jquery-ui';

@import 'base/reset';
@import 'base/typo';

@import 'layout/navigation';
@import 'layout/grid';
@import 'layout/header';
@import 'layout/footer';
@import 'layout/forms';

@import 'components/buttons';
@import 'components/jumbo';

@import 'pages/home';
@import 'pages/contact';

@import 'themes/theme';
@import 'themes/admin';
```

sustantiva
CON•SENTIDO

Sintaxis flujo de trabajo y buenas prácticas

Partimos con el formato, con 4 reglas sencillas:

- dos (2) espacios en blanco, en lugar de tabulaciones.
- idealmente, líneas de 80 caracteres.
- reglas CSS multilínea correctamente escritas.
- buen uso de los espacios en blanco.

```
// Yep
.foo {
  display: block;
  overflow: hidden;
  padding: 0 1em;
}

// Nope
.foo {
    display: block; overflow: hidden;

    padding: 0 1em;
}
```

sustantiva
CON•SENTIDO

Sintaxis flujo de trabajo y buenas prácticas

Y ahora agregamos otras reglas sencillas y generales:

- Los strings siempre deben ir entre comillas simples
- Si en un string se usan muchas veces las comillas simples, es mejor envolverlas en comillas dobles.

```
// Yep
$direction: 'left';

// Nope
$direction: left;
```

```
// Okay
@warn 'You can\'t do that.';

// Okay
@warn "You can't do that.";
```

sustantiva
CON•SENTIDO

Sintaxis flujo de trabajo y buenas prácticas

Siempre se deben mostrar los ceros a la izquierda antes de un valor decimal menor que uno. Nunca mostrar los ceros finales.

Cuando se trata de longitudes, el 0 nunca debe llevar el nombre de la unidad.

```
// Yep
.foo {
  padding: 2em;
  opacity: 0.5;
}

// Nope
.foo {
  padding: 2.0em;
  opacity: .5;
}
```

```
// Yep
$length: 0;

// Nope
$length: 0em;
```

sustantiva
CON•SENTIDO

Sintaxis flujo de trabajo y buenas prácticas

Los cálculos numéricos de nivel superior deben ir siempre entre paréntesis

En cuanto a colores, idealmente usar hsl o rgb, evitar los valores hexadecimales

```
// Yep
.foo {
  width: (100% / 3);
}

// Nope
.foo {
  width: 100% / 3;
}
```

```
// Yep
.foo {
  color: hsl(0, 100%, 50%);
}

// Also yep
.foo {
  color: rgb(255, 0, 0);
}

// Meh
.foo {
  color: #f00;
}

// Nope
.foo {
  color: #FF0000;
}

// Nope
.foo {
  color: red;
}
```

sustantiva
CON•SENTIDO

Variables

Las variables son la esencia de cualquier lenguaje de programación. Nos permiten reutilizar valores sin tener que copiarlos una y otra vez. Y lo más importante, permiten actualizar cualquier valor de manera sencilla. No más buscar y reemplazar valores de manera manual.

```
// La variable del desarrollador
$baseline: 2em;
```

Sin embargo CSS no es más que una cesta enorme que contiene todos nuestros elementos. A diferencia de muchos lenguajes, no hay scopes reales en CSS.

Debido a esto, debemos prestar especial atención cuando añadimos variables ya que pueden existir conflictos.

sustantiva
CON•SENTIDO

Variables

Una nueva variable debe crearse solo cuando se cumplen los siguientes criterios:

- el valor se repite al menos dos veces;
- es probable que el valor se actualice al menos una vez;
- todas las ocurrencias del valor están vinculadas a la variable (es decir, no por casualidad).

sustantiva
CON•SENTIDO

Variables

Scoping

En Sass, el ámbito (*scoping*) de las variables ha cambiado a lo largo de los años. Hasta hace muy poco, las declaraciones de variables dentro de los conjuntos de reglas y otros ámbitos eran locales por defecto. Sin embargo cuando ya había una variable global con el mismo nombre, la asignación local cambiaría dicha variable global. Desde la versión 3.4, Sass aborda correctamente el concepto de ámbitos y crea una nueva variable local en su lugar.

Los documentos hablan de *ocultar o sombrear la variable global*. Cuando se declara una variable que ya existe en el marco global dentro de un ámbito interno (selector, función, *mixin*…), se dice que la variable local esta *sombreando* a la variable global. Básicamente, la sobrescribe solo en el ámbito local.

Variables

Ejemplo de scoping

```
// Inicializar una variable global a nivel raiz.
$variable: 'valor inicial';

// Crear un *mixin* que sobrescribe la variable global.
@mixin global-variable-overriding {
  $variable: 'mixin value' !global;
}

.local-scope::before {
  // Crear una variable local que oculte la variable global.
  $variable: 'local value';

  // Incluir el *mixin*: sobrescribe la variable global.
  @include global-variable-overriding;

  // Imprimir el valor de la variable.
  // Es la variable **local** puesto que sobrescribe la global.
  content: $variable;
}

// Imprime la variable en otro selector que no la está sombreando.
// Es la variable **global**, como se esperaba.
.other-local-scope::before {
  content: $variable;
}
```

sustantiva
CON•SENTIDO

Mixins / Includes

Mixins

Los *mixins* son una de las características más utilizadas dentro de todo el lenguaje Sass. Son la clave para la reutilización y los componentes DRY.

Y por una buena razón: los *mixins* permiten a los autores definir estilos CSS que pueden volver a usarse a lo largo de toda la hoja de estilo sin necesidad de recurrir a las clases no semánticas, como por ejemplo .float-left.

Pueden contener reglas CSS completas y casi todo lo que se permite en cualquier parte de un documento Sass. Incluso pueden pasarse argumentos, al igual que en las funciones. Sobra decir que las posibilidades son infinitas.

sustantiva
CON•SENTIDO

Mixins / Includes

La regla de oro es que si detectas 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.

Un ejemplo simple y muy útil de un mixin.

```
@mixin clearfix {
  &::after {
    content: '';
    display: table;
    clear: both;
  }
}
```

sustantiva
CON•SENTIDO

Mixins / Include

Otro ejemplo válido sería un mixin para para darle tamaño a un elemento, definiendo tanto width como height al mismo tiempo. No solo hace que el código sea más fácil de escribir, sino que también será más fácil de leer.

```
/// Asigna un tamaño a un elemento
/// @author Kitty Giraudel
/// @param {Length} $width
/// @param {Length} $height
@mixin size($width, $height: $width) {
  width: $width;
  height: $height;
}
```

sustantiva
CON•SENTIDO

Mixins / Include

Para incluir el mixin en otro elemento usamos @include seguido del nombre del mixin, es así de fácil.

Ejemplo de esto:

```
@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;
}
```

sustantiva
CON•SENTIDO

Partials / imports

Como vimos en el patrón 7-1, en sass podemos manejar nuestro código en archivos parciales. Para esto, le damos un underscore antes del nombre del archivo, _partials.scss por ejemplo:

```scss
// _base.scss
$font-stack: Helvetica, sans-serif;
$primary-color: #333;

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

Creamos el partial
_base.scss

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

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

Lo llamamos con @use ‘base’ sin necesidad de indicar
la extensión o el underscore, sass sabe que nos referimos a ese archivo.

sustantiva
CON•SENTIDO

Partials / imports

Respecto a los imports, ojo que sass no recomienda su uso, privilegiando @use, como hicimos con los partials. De todas maneras te dejo un ejemplo de import

```scss
// foundation/_code.scss
code {
  padding: .25em;
  line-height: 0;
}
```

```scss
// foundation/_lists.scss
ul, ol {
  text-align: left;

  & & {
    padding: {
      bottom: 0;
      left: 0;
    }
  }
}
```

```scss
// style.scss
@import 'foundation/code', 'foundation/lists';
```

Nesting

Esta es una de las grandes ventajas de sass, permite generar una jerarquía de elementos y que se hereden cosas, así en este ejemplo ul, li y a, esos selectores, están nesteados y por tanto en css se verán como van ul nav li y nav a

SASS

```sass
nav {
  ul {
    margin: 0;
    padding: 0;
    list-style: none;
  }

  li { display: inline-block; }

  a {
    display: block;
    padding: 6px 12px;
    text-decoration: none;
  }
}
```

```css
nav ul {
  margin: 0;
  padding: 0;
  list-style: none;
}
nav li {
  display: inline-block;
}
nav a {
  display: block;
  padding: 6px 12px;
  text-decoration: none;
}
```

CSS


sustantiva
CON•SENTIDO