UD 7
Preprocesadores de CSS
Sass y Less

DISEÑO DE INTERFACES WEB

Técnico de Grado Superior Desarrollo de Aplicaciones Web

2024-25

J. Mario Rodríguez
jrodper183e@g.educaand.es

Contenidos

- Introducción e instalación.
- Reglas de maquetación de forma anidada.
- Definición de variables.
- Organización de archivos y código.
- Directivas Mixim.

Introducción

Cuando **CSS** comenzó a afianzarse (años 90) su propósito era simple:
proporcionar un lenguaje *básico* para describir el diseño de las páginas web.

Aunque cumplía con esta tarea, pronto quedaron en evidencia varias limitaciones:
los desarrolladores/diseñadores se encontraban con que por aquel entonces CSS…

- No tiene variables "reales"
  En CSS puro, cada valor de color, fuente, o tamaño debe ser escrito una y otra vez.
- No es fácil reutilizarlo
  CSS, sin mecanismos para reutilizar fragmentos de código, hace que sea habitual duplicar estilos.
- No permite "hacer programación"
  Carece de condicionales, funciones o cálculos avanzados y esto limita su flexibilidad.
- No es fácil organizarlo en proyectos grandes
  CSS puede volverse difícil de mantener debido a la falta de organización y estructura jerárquica clara. Además, está diseñado para ser escrito en archivos únicos y esto complica su modularidad.

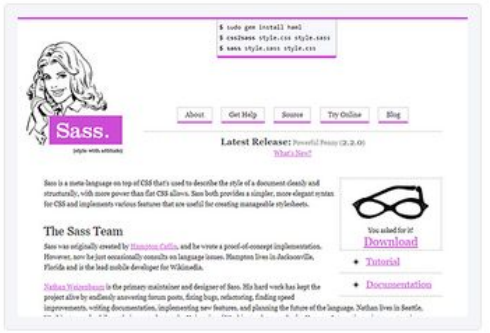
CSS ha evolucionado! características como CSS variables / custom properties, grid, flexbox, la función calc()…

Introducción

2006
October 5th

Sass 0.1.0

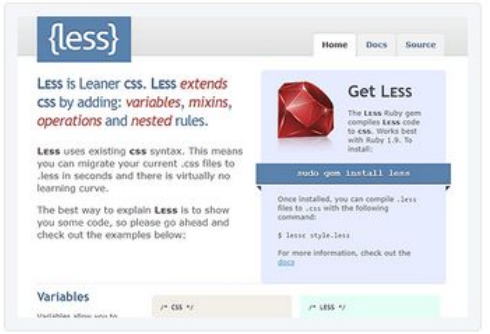
Hampton Catlin and Natalie Weizenbaum designed a **CSS preprocessor called Sass 0.1.0** (Syntactically awesome style sheets). Sass preprocessor is a scripting language interpreted or compiled into cascading styles. To the CSS syntax, Sass adds variables, mixins, selector inheritance, nesting rules, arithmetic operators, and other features.



2009
June

Less 1.0

Alexis Sellier designed a **CSS preprocessor called Less 1.0**, a dynamic styling language for cascading styles. The Less language was influenced by the existing Sass preprocessor. Less adds variables, mixins, arithmetic calculations, nesting rules and functions to the CSS syntax.



Introducción

Los desarrolladores comenzaron a buscar formas de extender las capacidades de CSS

Sass: el pionero (2006)

- Propósito inicial:
  Proporcionar herramientas para facilitar la escritura y mantenimiento de estilos, incluyendo variables, funciones y mixins.
- Introdujo inicialmente una sintaxis basada en indentación, pero más tarde añadió SCSS, lo que aumentó su adopción.

Less: la alternativa simplificada (2009)

- Propósito inicial:
  Hacer aún más amigable la transición para los desarrolladores.
- Menor curva de aprendizaje, al mantener una sintaxis muy parecida a CSS puro, pero con capacidades extendidas.
  Además, el éxito de Bootstrap y su integración con la herramienta favoreció su adopción.

Otros preprocesadores

- Stylus: flexible, con características como una sintaxis opcional más concisa.
- PostCSS: más reciente, se basa en un sistema de plugins y aprovecha las capacidades modernas de CSS.
- Tailwind CSS?

Introducción

Bueno, y ¿qué es un preprocesador de CSS?

es un programa que convierte un código escrito en un lenguaje extendido de hojas de estilo a CSS estándar

Problemas que pretenden resolver:

- Reutilizar el código (con variables y mixins)
- Mantener mejor los proyectos grandes
- Organizar el código, dividirlo en varios archivos y luego combinarlos

Instalación

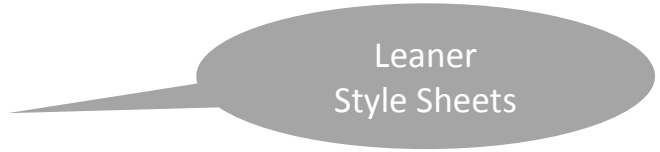
Sass

1. Instalar Node.js: nodejs.org
2. Ejecutar:
   `npm install -g sass`

Compilar un archivo .scss a .css:

```
sass input.scss output.css
```

Less



1. Instalar Node.js: nodejs.org
2. Ejecutar:
   `npm install -g less`

Compilar un archivo .less a .css:

```
lessc input.less output.css
```

Reglas de maquetación de forma anidada

En CSS, las reglas jerárquicas deben escribirse de forma explícita con selectores completos
esto puede resultar repetitivo y propenso a errores.

La anidación permite escribir reglas dentro de otras reglas
esto puede reflejar mejor la jerarquía del HTML en el código CSS
así el código será más legible, organizado y fácil de mantener
especialmente para estructuras complejas como menús de navegación

En Sass, las reglas se pueden anidar directamente.
El preprocesador se encarga de "desanidar" el código en CSS válido durante la compilación.

Reglas de maquetación de forma anidada

```
nav ul {
  margin: 0;
}
nav ul li {
  list-style: none;
}
nav ul li a {
  text-decoration:
none;
}
```

```
nav {
  ul {
    margin: 0;
    li {
      list-style: none;
      a {
        text-decoration: none;
      }
    }
  }
}
```

Reglas de maquetación de forma anidada

```
nav {
  background-color: #333;
}
nav ul {
  margin: 0;
  padding: 0;
}
nav ul li {
  display: inline-block;
}
nav ul li a {
  color: white;
  text-decoration: none;
}
nav ul li a:hover {
  color: #ddd;
}
```

```
nav {
  background-color: #333;
  ul {
    margin: 0;
    padding: 0;
    li {
      display: inline-block;
      a {
        color: white;
        text-decoration: none;
        &:hover {
          color: #ddd;
        }
      }
    }
  }
}
```

Definición de variables

Ventajas que se buscan al utilizar variables

- Evitar duplicaciones de valores frecuentes.
- Permitir cambios globales rápidos.

```
$primary-color: #3498db;
$secondary-color: #2ecc71;

button {
  background-color: $primary-color;
  color: $secondary-color;
}
```

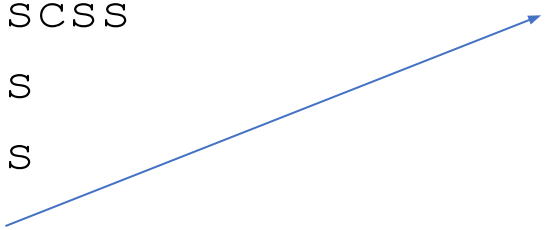
Organización de archivos y código

En proyectos grandes, dividir el código en varios archivos puede mejorar la organización.

Un archivo principal, por ejemplo main.scss, importa los archivos “parciales”.

Estructura de ejemplo:

```
styles/
|-- _variables.scss
|-- _header.scss
|-- _footer.scss
|-- main.scss
```



main.css

```
@import 'variables';
@import 'header';
@import 'footer';
```

Directivas Mixim

Un *Mixin* permite definir estilos que pueden incluirse en varios lugares con parámetros opcionales.

```
@mixin button-style($bg-color, $text-color) {
  background-color: $bg-color;
  color: $text-color;
  padding: 10px 20px;
  border-radius: 5px;
}

button {
  @include button-style(#3498db, #fff);
}
```

Sintaxis Sass
formato – variables – funciones – anidación

```scss
// SCSS
body {
  font-family: Arial, sans-serif;
  color: #333;
}
```

```
// Sass
body
  font-family: Arial, sans-serif
  color: #333
```

basada en indentación

```scss
$primary-color: #3498db;
$font-size: 16px;

button {
  background-color: $primary-color;
  font-size: $font-size;
}
```

```scss
@function calculate-rem($size) {
  @return $size / 16 * 1rem;
}

h1 {
  font-size: calculate-rem(32);
}
```

```scss
nav {
  ul {
    list-style: none;
    li {
      display: inline-block;
      a {
        text-decoration: none;
        color: $primary-color;
      }
    }
  }
}
```

Sintaxis Sass
mixins – condicionales – importación

```scss
@mixin button-styles($bg-color, $text-color: white) {
  background-color: $bg-color;
  color: $text-color;
  padding: 10px 20px;
  border-radius: 5px;
}

button.primary {
  @include button-styles($primary-color);
}

button.secondary {
  @include button-styles(black, $primary-color);
}
```

```scss
@mixin responsive($size) {
  @if $size == small {
    font-size: 12px;
  } @else if $size == large {
    font-size: 24px;
  } @else {
    font-size: 16px;
  }
}

p {
  @include responsive(small);
}
```

```scss
// _variables.scss
$primary-color: #3498db;

// main.scss
@use 'variables';

body {
  background-color: variables.$primary-color;
}
```

Sintaxis Less
variables – anidación - mixins

```
@primary-color: #3498db;
@font-size: 16px;

button {
  background-color: @primary-color;
  font-size: @font-size;
}
```

```
nav {
  ul {
    list-style: none;
    li {
      display: inline-block;
      a {
        text-decoration: none;
        color: @primary-color;
      }
    }
  }
}
```

```
.button-styles(@bg-color, @text-color: white) {
  background-color: @bg-color;
  color: @text-color;
  padding: 10px 20px;
  border-radius: 5px;
}

button.primary {
  .button-styles(@primary-color);
}

button.secondary {
  .button-styles(black, @primary-color);
}
```

Sintaxis Less
funciones – condicionales – importación

```less
@base-size: 16;

h1 {
  font-size: (@base-size * 2); // 32px
}

button {
  background-color: lighten(@primary-color, 20%);
}
```

```less
.responsive(@size) when (@size = small) {
  font-size: 12px;
}
.responsive(@size) when (@size = large) {
  font-size: 24px;
}
.responsive(@size) {
  font-size: 16px;
}

p {
  .responsive(small);
}
```

```less
// variables.less
@primary-color: #3498db;

// main.less
@import "variables.less";

body {
  background-color: @primary-color;
}
```

Diferencias

	Sass	**Less**
Variables	$variable	@variable
Mixins	@mixin y @include	Definición y llamada directa
Condicionales	@if, @else	Guardas (when)
Funciones personalizadas	Sí	No
Importación avanzada	@use, @forward	Solo @import

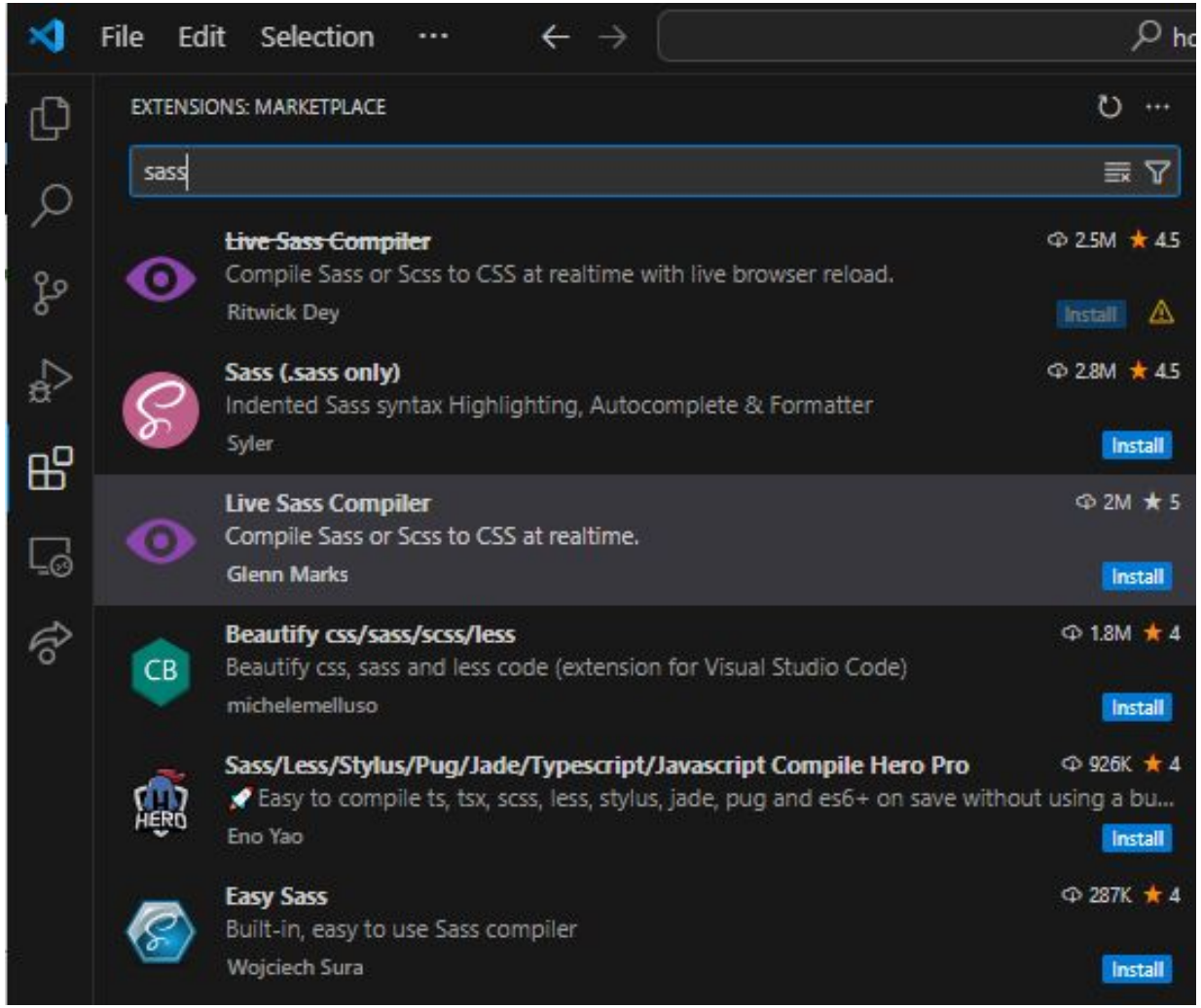
Herramientas relacionadas

You can also watch individual files or directories with the `--watch` flag. The watch flag tells Sass to watch your source files for changes, and re-compile CSS each time you save your Sass. If you wanted to watch (instead of manually build) your `input.scss` file, you'd just add the watch flag to your command, like so:

```
sass --watch input.scss output.css
```

You can watch and output to directories by using folder paths as your input and output, and separating them with a colon. In this example:

```
sass --watch app/sass:public/stylesheets
```



Herramientas relacionadas

Vite

PARCEL

webpack

Browsersync