

CSS PREPROCESSOR



Preprocesadores CSS... ¿Que son?

- Mozilla.org es quien define los estándares css y estipula:

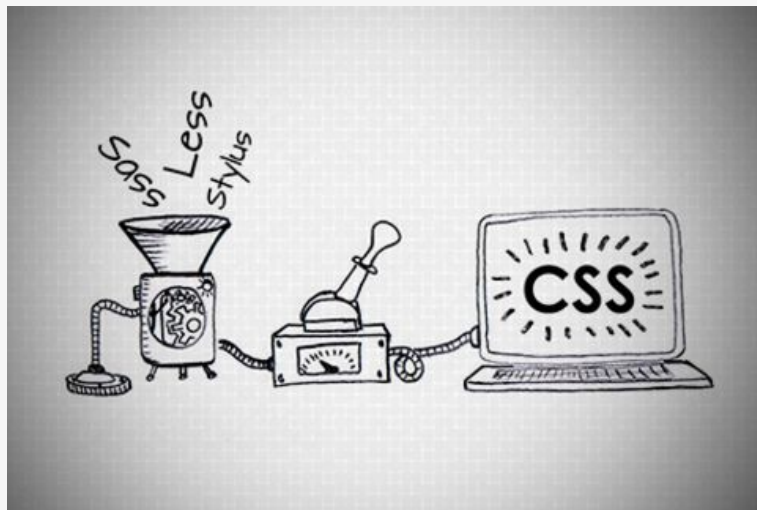
*“Son programas que permiten generar CSS a partir de la **SINTAXIS ESPECÍFICA DE CADA PREPROCESADOR**, pudiendo llegar a tener características de lenguajes de programación”.*

- Y esto... ¿Qué quiere decir?

Que podemos trabajar con funcionalidades adicionales, tales como:

- Variables
- Funciones
- Anidación
- Mixing
- Etc.

Luego, simplemente, compilamos a CSS!



Preprocesadores CSS - OBJETIVOS

- Permite una estructuración del CSS más legible y fácil de mantener a largo plazo y gran escala
- Contiene más funcionalidades (funciones, operadores)
- Reduce líneas de código
- Optimiza calidad de trabajo
- Esto se resume en:

DESARROLLO MÁS RÁPIDO Y ROBUSTO

CÓDIGO MÁS ESTRUCTURADO Y “DRY” (don't repeat yourself)

GRAN ESCALABILIDAD Y MANTENIBILIDAD



ALGUNAS (pocas) DESVENTAJAS:

- “Cierta” cantidad de boilerplate inicial
- Diversidad de sintaxis entre implementaciones
- Código final poco apto para producción (según gente muy quisquillosa)

Preprocesadores CSS

Los Preprocesadores NO son Frameworks...

←

PREPROCESADOR

- Sintaxis Propia
- Funcionalidades extras
- Se compila a CSS
- Ejemplos:
 - SASS
 - LESS
 - STYLUS



→

FRAMEWORK

- Hojas CSS con Layouts, elementos y clases ya creadas y preparadas para producción
- Estilo visual predefinido y consistente
- Menos boilerplate
- Ejemplos:
 - BOOTSTRAP
 - FOUNDATION
 - BULMA
 - TAILWIND

Preprocesadores CSS

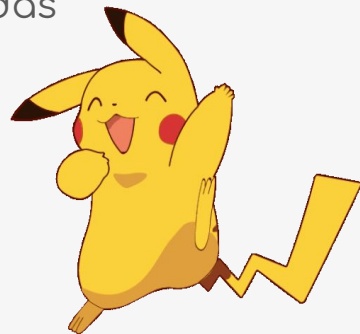
Los Preprocesadores TAMPOCO son Postprocesadores...




PREPROCESADOR

- Sintaxis Propia
- Funcionalidades extras
- Se compila a CSS
- Ejemplos:
 - SASS
 - LESS
 - STYLUS

POSTPROCESADOR

- Da formato “de producción” a CSS
- Autogenera polyfills (*suen a pokemon*)
- Permite usar especificaciones CSS aún no implementadas
- Ejemplos:
 - POSTCSS
 - STYLE COW
 - -PREFIX-FREE



FUNCIONALIDADES			
VARIABLES	✓	✓	✓
NESTING	✓	✓	✓
MIXINS	✓	✓	✓
FUNCTIONS	✓	✓✓	✓
INTERPOLATION	✓	✓	✓
NAMESPACES	✗	✓	✗
@IMPORTS	✓	✓	✓
OPERATORS	✗✓	✓	✓✓



SASS



Syntactically Awesome Style Sheets



- El más usado y longevo (2006)
- Diferentes implementaciones
- Sintaxis doble: SASS y SCSS
- Compatible con TODAS las librerías CSS
- Características similares o *superiores* 😎 al resto de preprocesadores...



HISTORIA E IMPLEMENTACIONES:

- Ruby Sass (2006-2019) 
- LibSass C/C++ (deprecated) 
- Dart library (current) => JS library



SINTAXIS DUAL:

.SASS
Basada en indentación

```
@mixin button-base()  
  @include typography(button)  
  @include ripple-surface  
  @include ripple-radius-bounded  
  
  display: inline-flex  
  position: relative  
  height: $button-height  
  border: none  
  vertical-align: middle  
  
  &:hover  
    cursor: pointer  
  
  &:disabled  
    color: $mdc-button-disabled-ink-color  
    cursor: default  
    pointer-events: none
```

.SCSS
Similar a CSS

```
@mixin button-base() {  
  @include typography(button);  
  @include ripple-surface;  
  @include ripple-radius-bounded;  
  
  display: inline-flex;  
  position: relative;  
  height: $button-height;  
  border: none;  
  vertical-align: middle;  
  
  &:hover { cursor: pointer; }  
  
  &:disabled {  
    color: $mdc-button-disabled-ink-color;  
    cursor: default;  
    pointer-events: none;  
  }  
}
```



LESS

Leaner Style Sheets



- Tan solo añade algunos aspectos al CSS, por lo que es muy sencillo de aprender. (Es *Less* complicado que el CSS...)
- Autocompilación de formato .less a .css
- Puede ejecutarse en Node.js o en el navegador
- Más orientado al uso de funciones que otros preprocesadores.
- Características parecidas a SASS y Stylus





SINTAXIS:

.less

Similar a .css y .scss

```
@link-color: #428bca;  
@link-color-hover:  
  darken(@link-color, 10%);
```

```
a,  
.link {  
  color: @link-color;  
}  
a:hover {  
  color:  
    @link-color-hover;  
}  
.widget {  
  color: #fff;  
  background:  
    @link-color;  
}
```

stylus

STYLUS

Expressive, dynamic, robust CSS

- Sintaxis corta y eficiente
- La estructura se basa en la indentación
- Su diseño está influenciado por Sass y LESS
- Está escrito en JADE y Node.js
- Formato .styl

SINTAXIS:

.styl

Similar a .sass

```
gris = #000
blanco = #fff

border-radius(n)
  -webkit-border-radius n
  -moz-border-radius n
  -ms-border-radius n
  -o-border-radius n
  border-radius n

body
  background blanco
  color gris

.button
  background gris
  color blanco
  display inline-block
```

CARACTERÍSTICAS PRINCIPALES:



VARIABLES

Las variables se utilizan para controlar valores que se vayan a utilizar de manera reiterada

CSS

```
#box {  
  color: #fff;  
  background: #00f;  
}
```

LESS

```
@font-color: #fff;  
@bg-color: #00f  
#box{  
  color: @font-color;  
  background: @bg-color;  
}
```

SASS (.scss)

```
$font-color: #fff;  
$bg-color: #00f;  
#box{  
  color: $font-color;  
  background: $bg-color;  
}
```

STYLUS

```
font-color = #fff  
bg-color = #00f  
#box  
  color font-color  
  background bg-color
```

MIXINS

Los mixins son una forma de agrupar un conjunto de propiedades con el fin de ahorrar líneas de código al reutilizarlas.

CSS

```
.class {  
  color: black;  
  background: white;  
}  
  
.post a {  
  color: red;  
  border-top: dotted 1px  
black;  
  border-bottom: solid  
2px black;  
}
```

LESS

```
.my-mixin {  
  color: black;  
}  
  
.my-other-mixin() {  
  background: white;  
}  
  
.bordered {  
  border-top: dotted 1px black;  
  border-bottom: solid 2px black;  
}  
  
.class {  
  .my-mixin();  
  .my-other-mixin();  
}  
  
.post a {  
  color: red;  
  .bordered();  
}
```

SASS (.scss)

```
@mixin my-mixin {  
  color: black;  
}  
  
@mixin my-other-mixin() {  
  background: white;  
}  
  
.bordered {  
  border-top: dotted 1px black;  
  border-bottom: solid 2px black;  
}  
  
.class {  
  @include my-mixin();  
  @include my-other-mixin();  
}  
  
.post a {  
  color: red;  
  @include bordered();  
}
```

MIXINS

Los mixins son una forma de agrupar un conjunto de propiedades con el fin de ahorrar líneas de código al reutilizarlas.

CSS

```
.class {  
  color: black;  
  background: white;  
}  
.post a {  
  color: red;  
  border-top: dotted 1px black;  
  border-bottom: solid 2px black;  
}
```

STYLUS (.styl)

```
my-mixin()  
  color: black  
my-other-mixin()  
  background: white  
.bordered  
  border-top: dotted 1px black  
  border-bottom: solid 2px black  
.class  
  my-mixin()  
  my-other-mixin()  
.post a  
  color: red  
  bordered()
```

SASS (.sass)

```
@mixin  
  my-mixin  
    color: black  
  my-other-mixin()  
    background: white  
.bordered  
  border-top: dotted 1px black  
  border-bottom: solid 2px  
black  
.class  
  @include my-mixin()  
  @include my-other-mixin()  
.post  
  a  
    color: red  
    @include bordered()
```

NESTING

Nos permite anidar elementos de forma jerárquica.
Especialmente útil al usar media queries en Responsive Design.

CSS

```
.navbar {  
  background-color: orangered;  
  padding: 1rem;  
}  
.navbar ul {  
  list-style: none;  
}  
.navbar li {  
  text-align: center;  
  margin: 1rem;  
}
```

LESS, SASS (.scss)

```
.navbar {  
  background-color: orangered;  
  padding: 1rem;  
  ul {  
    list-style: none;  
  }  
  li {  
    text-align: center;  
    margin: 1rem;  
  }  
}
```

STYLUS, SASS (.sass)

```
.navbar  
  background-color: orangered  
  padding: 1rem  
  ul  
    list-style: none  
  li  
    text-align: center  
    margin: 1rem
```

MAPS

Utilizados sobre todo para los mixins, los maps contienen pares de clave-valor

CSS

```
.button {  
  color: blue;  
  border: 1px solid  
    green;  
}
```

LESS, SASS (.scss)

```
#colors () {  
  primary: blue;  
  secondary: green;  
}  
  
.button {  
  color: #colors[primary];  
  border: 1px solid #colors[secondary];  
}
```

MAPS

Utilizados sobre todo para los mixins, los maps contienen pares de clave-valor

CSS

```
.button {  
  color: blue;  
  border: 1px solid  
green;  
}
```

SASS (.sass)

```
#colors()  
  primary: blue  
  secondary: green
```

```
.button  
  color: #colors[primary]  
  border: 1px solid #colors[secondary]
```

STYLUS (.styl)

```
colors = {  
  primary: blue,  
  secondary: green  
}
```

```
.button  
  color: colors['primary']  
  border: 1px solid colors.secondary
```


FUNCTIONS

Métodos integrados al igual que en los lenguajes de programación

- Relativas a la lógica de programación:
 - Condicionales → if - else, boolean...
 - String → replace, format, escape...
 - List → length, range, each...
 - Math → min, max, floor, round...
 - Type → isnumber, isurl, iscolor, is pixel...
 - Miscelánea → image-size, image-width, convert, get-unit...

FUNCTIONS

Métodos integrados al igual que en los lenguajes de programación

- Relativas al tratamiento del color:
 - Definición → rgb, rgba, hsl, hsv, hsva...
 - Canal → saturation, lightness, luminance...
 - Operaciones → darken, mix, contrast, shade...
 - Combinación → multiply, difference, overlay...

DIRECTIVAS @

En Sass, funciones como el @if, @else o el @for han de especificarse con @ delante en vez de como una función al uso.

SASS (.scss)

```
$type: monster;  
p {  
  @if $type == ocean {color: blue;}  
  @else if $type == matador {color: red;}  
  @else if $type == monster {color: green;}  
  @else {color: black;}}
```

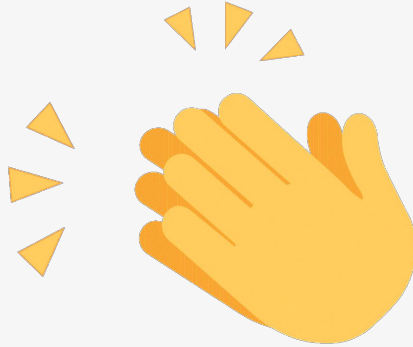
@IMPORT

Este mismo @ se utiliza en los preprocesadores, entre otras muchas cosas, para poder importar hojas de estilo tanto .css como .sass, .scss, .less o .styl, mediante @import

EJEMPLOS DE CÓDIGO:

- [Variables y Nesting con Media Queries en Responsive Web Design](#)
- [Mixins, Maps y Loops en Botones](#)
- [Mixins y Animación sobre Elementos](#)
- [Mixins demasiado grandes](#)
- [Nesting demasiados niveles](#)
- [Una curiosidad: The Mine](#)

¡GRACIAS!



BIBLIOGRAFÍA

- Documentación oficial:
 - [SASS](#)
 - [LESS](#)
 - [Stylus](#)
- Ejemplos de código por:
@jcoulderdesign, @taniarascia, @giana, @elrumordelaluz y @miquel
- Vídeos de YT:
 - [Aprende LESS en 15 MINUTOS](#)
 - [Aprende SASS en MENOS de 15 MINUTOS](#)