**SASS**

**Almacena datos con variables Sass**

Una característica de Sass que es diferente de CSS es que utiliza variables. Se declara y establece para almacenar datos, de forma similar a JavaScript.

En JavaScript, las variables se definen mediante las palabras clave lety const. En Sass, las variables comienzan con un $seguido del nombre de la variable.

Aquí hay un par de ejemplos:

**$main-fonts: Arial, sans-serif;**

**$headings-color: green;**

**Y para usar las variables:**

**h1 {**

**font-family: $main-fonts;**

**color: $headings-color;**

**}**

Un ejemplo en el que las variables son útiles cuando un número de elementos tiene que ser del mismo color. Si se cambia ese color, el único lugar para editar el código es el valor de la variable.

**<style type='text/scss'>**

**$text-color: red;**

**.header{**

**text-align: center;**

**}**

**.blog-post, h2 {**

**color: $text-color;**

**}**

**</style>**

**Anida CSS con Sass**

Sass permite anidar las reglas CSS, que es una forma útil de organizar una hoja de estilo.

Normalmente, cada elemento está dirigido a una línea diferente para darle estilo, así:

**nav {**

**background-color: red;**

**}**

**nav ul {**

**list-style: none;**

**}**

**nav ul li {**

**display: inline-block;**

**}**

Para un proyecto grande, el archivo CSS tendrá muchas líneas y reglas. Aquí es donde la anidación puede ayudar a organizar tu código colocando reglas de estilo hijo dentro de los elementos separados padres:

**nav {**

**background-color: red;**

**ul {**

**list-style: none;**

**li {**

**display: inline-block;**

**}**

**}**

**}**

**.blog-post {**

**h1 {**

**text-align: center;**

**color: blue;**

**}**

**p {**

**font-size: 20px;**

**}**

**}**

**Crea CSS reutilizable con Mixins**

En Sass, un mixin es un grupo de declaraciones de CSS que pueden reutilizarse a través de la hoja de estilo.

Las nuevas funciones de CSS tardan en ser adoptadas por completo y estar listas para su uso en todos los navegadores. A medida que se agregan funciones a los navegadores, las reglas CSS que las utilizan pueden necesitar prefijos de proveedor. Consideremos box-shadow:

**div {**

**-webkit-box-shadow: 0px 0px 4px #fff;**

**-moz-box-shadow: 0px 0px 4px #fff;**

**-ms-box-shadow: 0px 0px 4px #fff;**

**box-shadow: 0px 0px 4px #fff;**

**}**

Es mucho teclear para reescribir esta regla para todos los elementos que tienen un box-shadow, o para cambiar cada valor para probar diferentes efectos. Mixins son como funciones para CSS. Aqui esta como escribir una:

**@mixin box-shadow($x, $y, $blur, $c){**

**-webkit-box-shadow: $x $y $blur $c;**

**-moz-box-shadow: $x $y $blur $c;**

**-ms-box-shadow: $x $y $blur $c;**

**box-shadow: $x $y $blur $c;**

**}**

La definición empieza con @mixinseguido de un nombre personalizado. Los parámetros ( $x, $y, $blur, y $cen el ejemplo anterior) son opcionales. Ahora cada vez que se necesita una regla box-shadow, una sola línea llamando al mixin reemplaza el tener que escribir todos los prefijos del proveedor. Se llama a un mixin con la directiva @include:

**div {**

**@include box-shadow(0px, 0px, 4px, #fff);**

**}**

**<style type='text/scss'>**

**@mixin border-radius($radius){**

**-webkit-border-radius:$radius;**

**-moz-border-radius:$radius;**

**-ms-border-radius:$radius;**

**border-radius:$radius;**

**}**

**#awesome {**

**width: 150px;**

**height: 150px;**

**background-color: green;**

**@include border-radius(15px);**

**}**

**</style>**

**Usa @if y @else para agregar lógicamente a tus estilos**

La directiva @ifen Sass es útil para probar un caso específico: funciona igual que la sentencia ifen JavaScript.

**@mixin make-bold($bool) {**

**@if $bool == true {**

**font-weight: bold;**

**}**

**}**

Y al igual que en JavaScript, las directrices @else ify @elseprueban más condiciones:

**@mixin text-effect($val) {**

**@if $val == danger {**

**color: red;**

**}**

**@else if $val == alert {**

**color: yellow;**

**}**

**@else if $val == success {**

**color: green;**

**}**

**@else {**

**color: black;**

**}**

**}**

**@mixin border-stroke($val) {**

**@if $val == light {**

**border: 1px solid black;**

**}**

**@else if $val == medium {**

**border: 3px solid black;**

**}**

**@else if $val == heavy {**

**border: 6px solid black;**

**}**

**@else {**

**border: none;**

**}**

**}**

**Usa @for para crear un bucle Sass**

La directiva @foragrega estilos en un bucle, muy similar a un bucle foren JavaScript.

@forse utiliza de dos maneras: "de principio hasta el fin" o "de principio a fin". La principal diferencia es que "de principio a fin" excluye el número final como parte de la cuenta, y "de principio hasta el fin" incluye el número final como parte de la cuenta.

**Aquí hay un ejemplo de principio hasta el fin:**

**@for $i from 1 through 12 {**

**.col-#{$i} { width: 100%/12 \* $i; }**

**}**

La parte #{$i}es la sintaxis para combinar una variable ( i) con texto para hacer una cadena. Cuando el archivo Sass se convierte en CSS, tiene este aspecto:

**.col-1 {**

**width: 8.33333%;**

**}**

**.col-2 {**

**width: 16.66667%;**

**}**

**...**

**.col-12 {**

**width: 100%;**

**}**

Esta es una manera poderosa de crear un diseño de cuadrícula (grid). Ahora tienes doce opciones de ancho de columna disponibles como clases CSS.

**@for $j from 1 through 5 {**

**.text-#{$j} { font-size: 15px \* $j; }**

**}**

**Usa @each para asignar elementos en una lista**

El último desafío mostró cómo la directiva @forutiliza un valor inicial y final para hacer un bucle un determinado número de veces. Sass también ofrece la directiva @eachque hace un bucle sobre cada elemento de una lista o mapa. En cada iteración, la variable se asigna al valor actual de la lista o del mapa.

**@each $color in blue, red, green {**

**.#{$color}-text {color: $color;}**

**}**

Un mapa tiene una sintaxis ligeramente diferente. He aquí un ejemplo:

**$colors: (color1: blue, color2: red, color3: green);**

**@each $key, $color in $colors {**

**.#{$color}-text {color: $color;}**

**}**

Ten en cuenta que la variable $keyes necesaria para hacer referencia a las claves en el mapa. Por el contrario, el CSS compilado tendría color1, color2... en él. Los dos ejemplos anteriores se predefinieron en el siguiente CSS:

**.blue-text {**

**color: blue;**

**}**

**.red-text {**

**color: red;**

**}**

**.green-text {**

**color: green;**

**}**

**<style type='text/scss'>**

**@each $color in blue, black, red {**

**.#{$color}-bg {background-color: $color;}**

**}**

**div {**

**height: 200px;**

**width: 200px;**

**}**

**</style>**

**Aplica un estilo hasta que se cumpla una condición con @while**

La directiva @whilees una opción con funcionalidad similar al bucle whilede JavaScript. Crea reglas CSS hasta que se cumpla una condición.

El desafío @fordio un ejemplo para crear un sistema de cuadrícula (grid) simple. Esto también puede funcionar con @while.

**$x: 1;**

**@while $x < 13 {**

**.col-#{$x} { width: 100%/12 \* $x;}**

**$x: $x + 1;**

**}**

Primero, defina una variable $xy establezca a 1. A continuación, utilice la directiva @whilepara crear el sistema de cuadrícula while $x sea menor que 13. Después de configurar la regla CSS para width, $xse incrementa por 1 para evitar un bucle infinito.

**<style type='text/scss'>**

**$x: 1;**

**@while $x < 11 {**

**.text-#{$x} {**

**font-size: 15px \* $x;**

**}**

**$x: $x + 1;**

**}**

**</style>**

**Divide tus estilos en trozos más pequeños con parciales**

Parciales en Sass son archivos separados que contienen segmentos de código CSS. Estos se importan y son utilizados en otros archivos Sass. Esta es una gran manera de agrupar un código similar en un módulo para mantenerlo organizado.

Los nombres de los parciales comienzan con el carácter de guión bajo ( \_), que le dice a Sass que es un pequeño segmento de CSS y no para convertirlo en un archivo CSS. También, los archivos Sass terminan con la extensión de archivo .scss. Para introducir el código en el parcial en otro archivo Sass, utilice la directiva @import.

Por ejemplo, si todos tus mixins se guardan en un llamado "\_mixins.scss", y son parcialmente necesarios en el archivo "main.scss", es cómo usar el archivo principal:

**@import 'mixins'**

Ten en cuenta que el guión bajo y la extensión del archivo no son necesarios en la declaración import- Sass entiende que es un parcial. Una vez que un parcial es importado en un archivo, todas las variables, mixins y otros códigos están disponibles para usar.

**@import 'variables'**

**Hereda un conjunto de estilos CSS a otro elemento**

Sass tiene una función llamada extendque facilita tomar prestadas las reglas CSS de un elemento y construir sobre ellas en otro.

Por ejemplo, el siguiente bloque de reglas CSS da estilo a la clase .panel. Este tiene un background-color, heighty border.

**.panel{**

**background-color: red;**

**height: 70px;**

**border: 2px solid green;**

**}**

Ahora tienes otro panel llamado .big-panel. Tiene las mismas propiedades base que .panel, pero también necesita widthy font-size. Es posible copiar y pegar las reglas de CSS iniciales de .panel, pero el código se vuelve repetitivo a medida que agrega más tipos de paneles. La directiva extendes una forma simple de reutilizar las reglas escritas para un elemento y luego agregar más para otro:

**.big-panel{**

**@extend .panel;**

**width: 150px;**

**font-size: 2em;**

**}**

El .big-paneltendrá las mismas propiedades que .panelademás de los nuevos estilos.

**<style type='text/scss'>**

**h3{**

**text-align: center;**

**}**

**.info{**

**width: 200px;**

**border: 1px solid black;**

**margin: 0 auto;**

**}**

**.info-important {**

**@extend .info;**

**background-color: magenta;**

**}**

**</style>**

**<h3>Posts</h3>**

**<div class="info-important">**

**<p>This is an important post. It should extend the class ".info" and have its own CSS styles.</p>**

**</div>**

**<div class="info">**

**<p>This is a simple post. It has basic styling and can be extended for other uses.</p>**

**</div>**