Entendendo o mixin

Um mixin é uma diretiva que permite que você defina várias regras com diversos argumentos. Pense nisso como uma função que irá retornar conteúdo CSS ao invés de um valor.

Agora que cobrimos a terminologia, vamos dizer que você encontra algumas declarações que são repetidas várias vezes ao longo da sua folha de estilos. Você que está familiarizado com o conceito de DRY (Don't Repeat Yourself), sabe que a repetição de código é ruim. Para corrigir isso, você pode escrever um mixin para todas aquelas declarações repetidas:

```
@mixin center() {
    display: block;
    margin-left: auto;
    margin-right: auto;
}

.container {
    @include center();
    /* Outros estilos aqui... */
}

/* Outros estilos... */

.image-cover {
    @include center;
}
```

Nota: Se você não passar um argumento para um mixin, você pode omitir os parênteses. Na verdade, você pode até omiti-los na definição do `@mixin`.

Com este mixin recém-criado, você não precisa repetir aquelas três linhas de código cada vez que precisar centralizar um elemento; você simplesmente inclui o mixin. Muito prático, não é?!

Algumas vezes você vai querer um mixin para construir o que você chamaria de *shorthand* para algumas propriedades. Por exemplo, largura e altura. Você não está cansado de escrever as duas linhas várias e várias vezes? Especialmente quando ambas tem o mesmo valor? Bem, vamos lidar com isso usando um mixin!

```
@mixin size($width, $height: $width) {
    width: $width;
    height: $height;
}
```

Muito simples, não é? Note como deixamos o parâmetro `$height` ser opcional e, por padrão assumir o mesmo valor do parâmetro `$width` na assinatura do mixin. Agora, sempre que você precisar definir as dimensões para um elemento, você pode simplesmente fazer isso:

```
.icon {
    @include size(32px);
}

.cover {
    @include size(100%, 10em);
}
```

Nota: Um outro bom exemplo de mixin seria este aqui que eu fiz para evitar de escrever as posições `top`, `left`, `right` e `bottom` toda vez que quiser utilizar um sistema de posicionamento diferente do estático.

Conhecendo seu Placeholder

Placeholders são um tipo de coisa estranha. Eles são classes que não são retornadas quando o seu SCSS é compilado. Você deve então pensar: *"Qual é o sentido disso?"*. Na verdade, o ponto seria minímo senão fosse a expressão `@extend`. Mas vamos por partes. Essa é a forma que você escreve um placeholder:

```
%center {
    display: block;
    margin-left: auto;
    margin-right: auto;
}
```

Nota do editor: Como um placeholder, um mixin é igualmente inútil, a menos que seja referenciado, assim essa seção não está dizendo que eles são diferentes nesse aspecto, mas apenas esclarecendo

que mesmo que se pareça similar com um bloco de declaração CSS, não será gerado por conta própria.

Basicamente você escreve exatamente como uma classe `CSS` exceto pelo símbolo % ao invés do ponto. Além disso, segue as mesmas regras de nomenclatura das classes.

Agora, se você tentar compilar seu SCSS, você não vai ver esse pedaço de código no arquivo gerado. Como eu disse: **placeholders não são compilados**.

Então, por agora, esse placeholder é totalmente inútil. Você não consegue fazer qualquer uso dele a não ser que você veja o `@extend`. Um `@extend` tem como objetivo herdar as propriedades de um seletor CSS / SCSS placeholder. Aqui como usá-lo:

```
.container {
  @extend %center;
}
```

Ao fazer isso, o arquivo SASS vai pegar o conteúdo do placeholder `%center` e aplicá-lo no `.container` (mesmo que isso não aconteça exatamente assim – mas isso não é importante agora). Como eu disse, você também pode *estender* seletores CSS já existentes (além de placeholders SCSS) dessa maneira:

```
.table-zebra {
  @extend .table;

  tr:nth-of-type(even) {
    background: rgba(0,0,0,.5);
  }
}
```

Esse é um caso muito comum para o uso do `@extend`. Nesse caso, pedimos para a classe `.table-zebra` se comportar exatamente como a classe `.table` e então adicionamos as regras específicas da classe `.table-zebra`. *Estender* seletores é bastante conveniente quando você desenvolve seu site ou aplicação em componentes modulares.

Qual utilizar?

Então, a pergunta permanece: o que você deve usar? Bem, como tudo em nossa área: **depende**. Depende do contexto e, em uma outra análise, do que você está querendo fazer.

O melhor conselho seria: se você precisa de variáveis, utilize o mixin. Caso contrário, use o placeholder. Há duas razões para isso:

- Primeiro, você não pode usar variáveis em um placeholder. Na verdade, até pode, mas você não consegue *passar* uma variável em um placeholder para gerar um conteúdo específico de CSS, como você faria em um mixin.
- Segundo, a forma como o SASS lida com os mixins, os torna muito incovenientes quando você os utiliza sem variáveis contextuais. Simplificando: o SASS vai duplicar a saída de um mixin toda vez que você o utilizá-lo, resultando não apenas em CSS duplicado, mas também em uma folha de estilos maior.

Considere o primeiro exemplo desse artigo:

```
@mixin center {
  display: block;
  margin-left: auto;
  margin-right: auto;
}

.container {
  @include center;
}

.image-cover {
  @include center;
}
```

O CSS compilado seria esse:

```
.container {
  display: block;
  margin-left: auto;
  margin-right: auto;
}

.image-cover {
  display: block;
  margin-left: auto;
```

```
  margin-right: auto;
}
```

Observou o CSS duplicado? Ele não é tão prejudicial se forem apenas três linhas duplicadas, mas se você tiver muitos mixins que são usados várias vezes em um projeto, essas três linhas podem facilmente se tornarem 300. E se reformularmos nosso exemplo, só que dessa vez utilizando o placeholder?

```
%center {
  display: block;
  margin-left: auto;
  margin-right: auto;
}

.container {
  @extend %center;
}

.image-cover {
  @extend %center;
}
```

Agora, esse é o CSS gerado:

```
.container, .image-cover {
  display: block;
  margin-left: auto;
  margin-right: auto;
}
```

Muito melhor! A compilação leva vantagem agrupando seletores, sem nenhum estilo repetido. Assim, sempre que você quiser evitar a escrever as mesmas propriedades diversas vezes, sabendo que elas não mudarão, é uma boa idéia *estender* um placeholder. Isso resultará em um código CSS compilado muito mais limpo.

Por outro lado, se você precisa escrever as mesmas propriedades em vários lugares mas com valores diferentes (tamanho, cores, etc), um mixin é o melhor caminho a seguir. Agora se você possui ambos, um grupo de valores fixos e outro de valores variáveis, você deve tentar usar uma combinação dos dois.

```
%center {
  margin-left: auto;
  margin-right: auto;
  display: block;
}

@mixin skin($color, $size) {
  @extend %center;
  background: $color;
  height: $size;
}

a { @include skin(pink, 10em) }
b { @include skin(blue, 90px) }
```

Neste caso, o mixin está *estendendo* o placeholder para os valores fixos em vez de jogá-los diretamente em seu corpo. Isso gera um CSS mais limpo:

```
a, b {
  margin-left: auto;
  margin-right: auto;
  display: block;
}

a {
  background: pink;
  height: 10em;
}

b {
  background: blue;
  height: 90px;
}
```