Regras de Estilo @mixin e @include

O que fazem as Regras de Estilo @mixin e @include?

As regras de estilo @mixin e @include trabalham juntas.

A regra @mixin é usada para gerar uma sub-rotina dentro do nosso Sass, o que seria uma sub-rotina? Seria um trecho de código que usamos com muito frequência, como resets por exemplo, afinal constantemente temos que resetar elementos html zerando suas margens e paddings, podemos criar um trecho que código que faz isso e quando precisarmos utilizar essa formatação chamamos por esse trecho, isso é o @mixin.

Usar a regra de estilo @mixin é uma grande vantagem pois ela evita a criação excessiva de classes não semânticas, gerando um código menos inchado, mais leve e mais semântico. Fazendo com que geremos páginas cada vez mais leves, fáceis de serem encontradas pelos navegadores e menos complicadas durante uma possível manutenção de código.

Outra grande vantagem dos mixins é que podemos encapsular estilos dentro de um mixin, por exemplo, podemos gerar mixins que contém regras aninhadas dentro do próprio mixin, como por exemplo um mixin de um botão, que dentro dele já contém a regra “hover”, fazendo o botão ter um comportamento diferenciado quando o mouse passa por cima dele. O melhor desse encapsulamento é que podemos incluí-lo ou excluí-lo de uma propriedade sem preocupações, uma vez que o mixin estiver dentro de uma propriedade, ele fará parte dela como se estive sido escrito dentro dela, mas uma vez que estiver fora, ele não afetará a propriedade em nada, ela vai continuar a funcionar normalmente.

Se não bastasse todas essas vantagens, o mixin ainda pode receber valores, assim como as funções, por exemplo, imagine que temos um mixin para um botão, mas desejamos mudar a cor do botão de vez enquanto, para isso basta abrir um campo para parâmetros e fazer a mudança da cor do botão sempre que desejarmos.

A regra @include é um complemento da regra @mixin, enquanto os mixins são usados para criar as sub-rotinas, o @include é usado para poder incluir um mixin dentro de um contexto de código, ou seja, para incluir os mixins no contexto local de um seletor.

Como utilizar @minix e @include?

Podemos fazer isso da seguinte maneira:

Criando módulo Sass:

Arquivo _list.scss:

```
@mixin reset {
     margin: 0;
     padding: 0;
     list-style: none;
}

@mixin horizontal {
     li {
          @include reset;
          display: inline-block;
          margin: {
                Left: -2px;
          }
     }
}
```

Perceba que o mixin acima serve para resetar as margens, padding e estilo de uma lista, e temos um mixin mais baixo que recebe todas as funções do mixin de cima e ainda aplica a lista na horizontal por tirar o display block padrão para inline-block;
Perceba que temos 2 mixins, onde o segundo mixin – “horizontal” – chama pelo mixin “reset” com a ajuda da regra @include. Perceba que temos que dar nomes aos mixins, e podemos colocar dentro deles uma formatação de código Sass comum, com o estilo aninhado que estamos acostumados. Esses mixins não possuem parâmetros á receber, mas se quiséssemos receber parâmetros, bastava adicionar os parênteses e as variáveis para receber estes parâmetros.

Carregando módulo dentro do arquivo Sass gerador:

Arquivo style.scss:

```
@use “list”;
```

```
nav ul {
     @include list.horizontal;
}
```

Perceba que quando chamamos o mixin, precisamos de chamar apenas um mixin, e não 2, pois um mixin já estava dentro de outro. E atribuímos o mixin “horizontal” as tags nav e ul.

Resultado no arquivo final:

Arquivo Final:

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

nav ul li {
     display: inline-block;
     margin-left: -2px;
}
```

Perceba que no arquivo final, o Sass soube separar direitinho que formatações deveriam equivaler ao nav, ul e li respectivamente.

Cuidados ao nomear mixins!!!

Quando damos nomes aos nossos mixins devemos tomar o cuidado de não escolher nomes que comecem com números, caracteres especiais ou que tenham o mesmo nome que uma função built-in do Sass, pois, podemos acabar usando um seletor universal para renomear nossos módulos e acabar designando uma função com o mesmo nome de uma função built-in, o que poderia gerar um conflito.

Além disso devemos tomar cuidado com o uso de hífen (-) e underline (_), afinal ambas são entendidas da mesma forma pelo Sass, então se tivermos um mixin cujo nome é: “scale-color” e outra “scale_color”, para o Sass elas serão o mesmo mixin.