{less}

css menos código

helder da rocha

(18 de março 2014)

{apresentação}

Pré-processadores CSS como Less e SASS/SCSS são hoje ferramentas essenciais para o desenvolvimento Web. Eles estendem o CSS com recursos que tornam o desenvolvimento mais eficiente, mais fácil de manter e reutilizar. Less é um dos mais populares pré-processadores CSS. É usado em bibliotecas de temas, aplicações e em frameworks populares de design responsivo como o Twitter Bootstrap.

Este curso explora a tecnologia Less em detalhes, oferecendo uma introdução abrangente e cobrindo tópicos mais avançados através de exemplos. Você deve conhecer HTML e CSS, uma vez que Less é uma extensão do CSS e também é importante entender o CSS que é gerado quando se usa Less. Não é preciso conhecer JavaScript, mas se você souber usar a linguagem poderá explorar alguns recursos não-essenciais de Less que requerem JavaScript. O uso de Less com JavaScript é abordado em um dos tópicos do capítulo 9.

Less gera CSS, então é mais importante observar o código gerado que a aparência do efeito do CSS na página. Ao depurar código Less é essencial ter acesso ao CSS gerado. Neste curso, para demonstrar determinados conceitos são usados exemplos de Less que geram CSS que não necessariamente servem para uso em páginas Web, ou mesmo CSS incorreto para ilustrar diferentes efeitos do processamento de funções ou operações do Less.

Para acompanhar este tutorial você deve ter um computador com um editor de textos e um compilador Less instalado. Algumas opções de editores e compiladores serão apresentadas no primeiro capítulo. O capítulo 10 inclui referências usados na pesquisa deste material, tutoriais, artigos e outras informações que serão úteis durante o treinamento.

{arquivos}

O código mostrado nos exemplos deste livro pode ser baixado da Internet no site: http://github.com/argonavisbr/LessCourse em formato ZIP ou através de GIT *clone*.

Todo o código foi testado e compilado usando a versão 1.7 do Less (fevereiro de 2014) durante a geração do texto deste livro em XML. O texto foi escrito em Markdown e posteriormente transformado em HTML e XML, que foi usado para gerar o documento em RTF. As fontes `.less` apresentadas no livro e os resultados `.css` resultantes da compilação estão todos disponíveis na pasta `examples`, em sub-pastas correspondentes a cada capítulo. Alguns exemplos utilizam arquivos HTML e imagens que estão disponíveis nas pastas `html` e `images`, respectivamente.

{distribuição e termos de uso}

{sobre o autor}

Helder da Rocha é programador, escritor, professor, ator, músico e artista plástico. Trabalha com tecnologias Web e Java desde 1994. É autor de um livro sobre HTML e CSS publicado em 1996, e sobre JavaScript publicado em 1998, e de diversos tutoriais disponibilizados online sobre Java, XML e tecnologias relacionadas. Ele pode ser contactado pelo email *helder@argonavis.com.br*.

{conteúdo resumido}

{conteúdo em detalhes}

{1: introdução }

1.1 O que é Less?

Less é um *pré-processador* CSS. Ele *gera* CSS. É também uma *extensão* do CSS: um documento CSS é um documento Less válido. Less acrescenta vários recursos que tornam mais eficientes a criação e manutenção de folhas de estilo do CSS.

1.2 Por que usar less?

Less não é necessário, mas Less também não quebra os padrões existentes (não depende de suporte de browser) e torna o CSS mais fácil de manter.

Documentos Less são bem mais curtos que documentos CSS e geralmente têm muito menos repetição de código. Alguns dos recursos que Less acrescenta ao CSS incluem funções, variáveis, mixins, condicionais, escopo, aninhamento de blocos e outros recursos comuns em linguagens de programação. Less também facilita a criação de folhas de estilo independentes de browser e plataforma.

1.2.1 Variáveis

Less suporta constantes (que são chamadas de *variáveis*). Através delas é possível declarar um valor (ex: uma cor, url ou fonte) e reusar esse valor várias vezes na folha de estilos, evitando a repetição e facilitando a alteração. Por exemplo:

```
@cor-base: #f74020;
.sec1 {
     color: @cor-base;
     border-color: @cor-base;
}
```

1.2.2 Mixins

Com Less é possível utilizar uma coleção de propriedades declaradas para um seletor várias vezes, simplesmente incluindo o nome do seletor dentro de outro bloco. Esses seletores são chamados de *mixins*.

Pode-se ainda criar mixins que recebem parâmetros, mixins que definem variáveis de retorno, mixins condicionais e mixins recursivos que funcionam como loops.

O primeiro seletor abaixo está sendo usado no segundo como um mixin.

```
.girar-90-graus {
    transform: rotate(90deg);
    -webkit-transform: rotate(90deg);
    -moz-transform: rotate(90deg);
    -o-transform: rotate(90deg);
    -ms-transform: rotate(90deg);
}

.titulo-lateral {
     font-weight: 900;
     .girar-90-graus; // props de .gerar-90-graus serão inseridas aqui
}
```

As duas barras `//` representam um bloco de comentário em Less.

1.2.3 Aninhamento e escopo

Para aplicar um estilo em um elemento de acordo com seu contexto, CSS define uma sintaxe compacta:

```
.secao > div .item {
     color: blue;
}
```

Mas que causa repetição de código se também for necessário aplicar estilos nos outros elementos do contexto:

```
.secao > div {
     width: 560px;
}

.secao {
     position: absolute;
}
```

Less acrescenta uma segunda sintaxe que evita a repetição de código em situações como esta, e também cria um escopo onde podem ser usadas variáveis locais.

Os três blocos acima poderiam ser reduzidos a um só bloco usando declarações aninhadas em Less:

```
.secao {
     position: absolute;
     > div {
         width: 560px;
         .item {
            color: blue;
         }
     }
}
```

1.2.4 Extensão

Suponha que você tenha o seguinte CSS:

```
.secao1, .lateral {
    background: url(images/bg.svg);
    color: #f789a1;
}
.tabela td td > p {
     font-weight: bold;
     text-decoration: underline;
}
.secao2 {
     border: solid blue 1px;
}
```

Agora você necessita que um novo seletor `.secao2` tenha todos os estilos de `.lateral` e `.tabela td td > p`. Isto pode ser feito através da adição de `.secao2` nas outras duas declarações acima:

```
.secao2, .secao1, .lateral {
    background: url(images/bg.svg);
    color: #f789a1;
}
.secao2, .tabela td td > p {
     font-weight: bold;
     text-decoration: underline;
}
.secao2 {
     border: solid blue 1px;
}
```

Less faz esse tipo de operação automaticamente com o pseudo-seletor `:extend`, gerando o CSS acima:

```
.secao2:extend(.lateral, .tabela td td > p) {
     border: solid blue 1px;
}
```

1.2.5 Operadores e funções

Muitas vezes precisamos fazer contas em CSS para alterar posicionamentos e cores. Less facilita isto acrescentando ao CSS a possibilidade de se usar operações aritméticas, blocos condicionais, blocos de repetição, funções matemáticas e funções de manipulação de cores. Combinado com o uso de

variáveis e mixins pode-se automatizar diversas tarefas repetitivas e evitar usar outras linguagens de programação para gerar cores e dimensões.

```
@largura-total: 1024px;
@largura-bloco: 960px;
@margem-esquerda: (@largura-total - @largura-bloco) / 2;
@cor-base: #800080;
.secao3 {
   left: @margem-esquerda;
   background: darken(@cor-base + #008000, 20%);
   color: spin(@cor-base, 180deg); // inverte a cor girando a matiz
}
```

O código acima gera o seguinte CSS:

```
.secao3 {
   left: 32px;
   background: #4d4d4d;
   color: #008000;
}
```

1.3 Como usar

O desenvolvimento com Less possui duas etapas:

1. Criação do arquivo `.less`
2. Compilação do arquivo `.less` (geração de um arquivo CSS)

Normalmente Less é usado apenas em desenvolvimento. É possível carregar um documento `.less` diretamente no browser que irá gerar um CSS durante a carga da página, porém isto costuma ser ineficiente e pode fazer com que a renderização da página demore.

Um documento Less pode ser criado em qualquer editor de textos. Várias ferramentas de desenvolvimento Web populares suportam Less ou de forma nativa ou via plugin. Elas fazem a geração do CSS automaticamente.

Less *é* CSS, então para criar um documento `.less` pode-se simplesmente usar um documento CSS e alterar a sua extensão para .less.

Há duas opções para gerar CSS através de um documento Less:

1. Usar o compilador `lessc` (linha de comando)
2. Usar compiladores automáticos (`lessc` com opção `--watch`, compiladores gráficos, plugins)

1.4 Compilação

1.4.1 Linha de comando

Em sistemas Windows, Mac ou Linux, o Less é instalado como aplicação JavaScript (Node.js) através do *Node Package Manager* (NPM). Se o Less estiver instalado no seu sistema, abra uma janela do terminal de comandos e digite:

```
lessc -version
```

Na tela será impressa a versão do Less que está instalada no seu sistema. Este tutorial assume que você está usando pelo menos a versão 1.7.

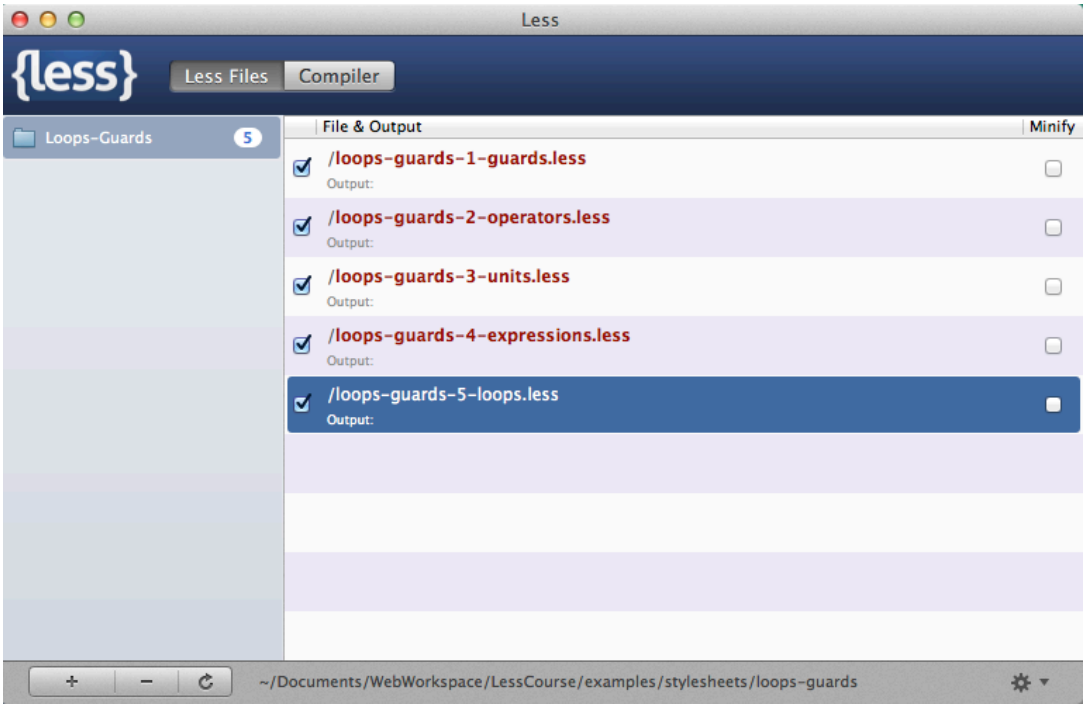
Para gerar um arquivo CSS a partir de um documento Less, a sintaxe é:

```
lessc arquivo.less > arquivo.css
```

O compilador `lessc` possui várias opções documentadas no site oficial do Less em *http://lesscss.org*. Uma das opções permite configurar a aplicação para monitorar documentos `.less` ou pastas, e gerar o `.css` automaticamente a cada alteração.

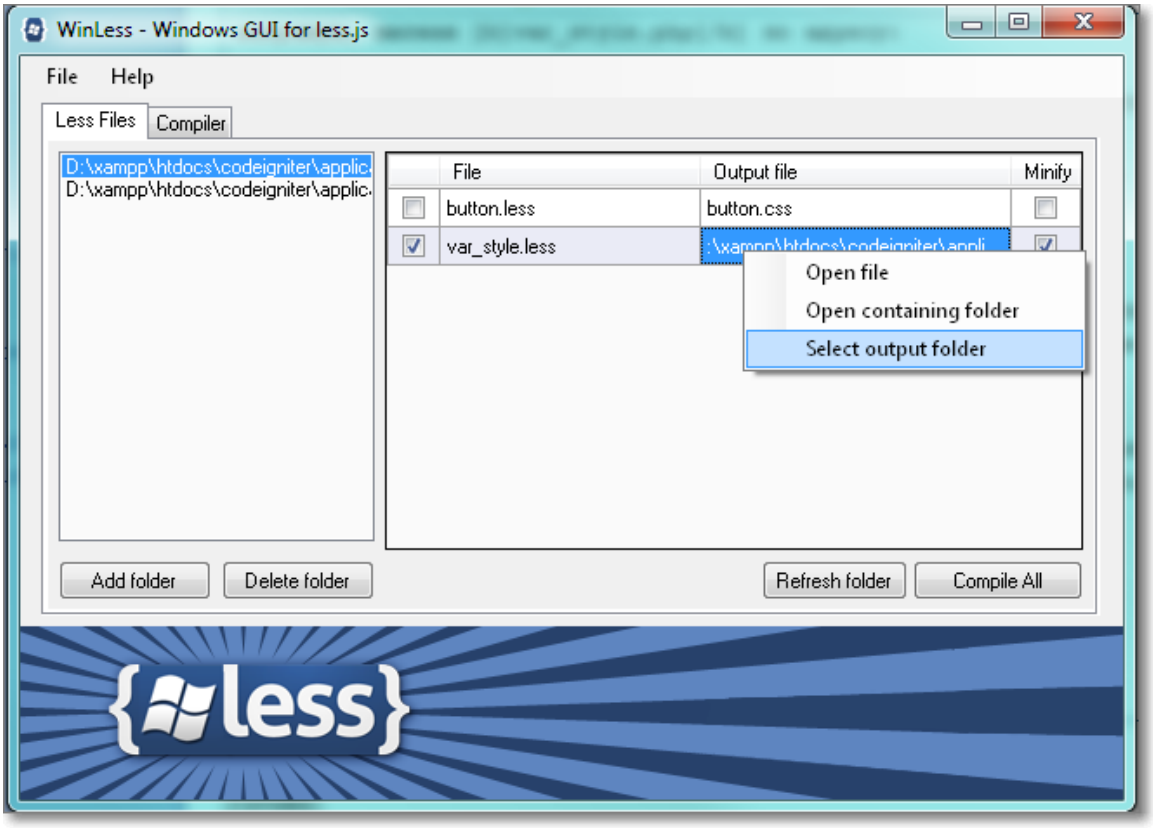
1.4.2 Ferramentas gráficas

Se você não se sente muito à vontade usando o terminal, existe a opção de utilizar um compilador gráfico como *less.app* (*Mac OS)* ou *WinLess* (*Windows*). Eles permitem configurar um ambiente onde arquivo serão compilados e recompilados sempre que forem alterados, e não precisam do terminal.



O *less.app* (imagem acima) para MacOS permite associar uma pasta ou documentos individuais que serão monitorados A cada alteração eles serão compilados. Pode-se configurar o diretório onde os documentos CSS resultantes da compilação serão guardados.

O *WinLess* é similar ao *less.app* para sistemas Windows.



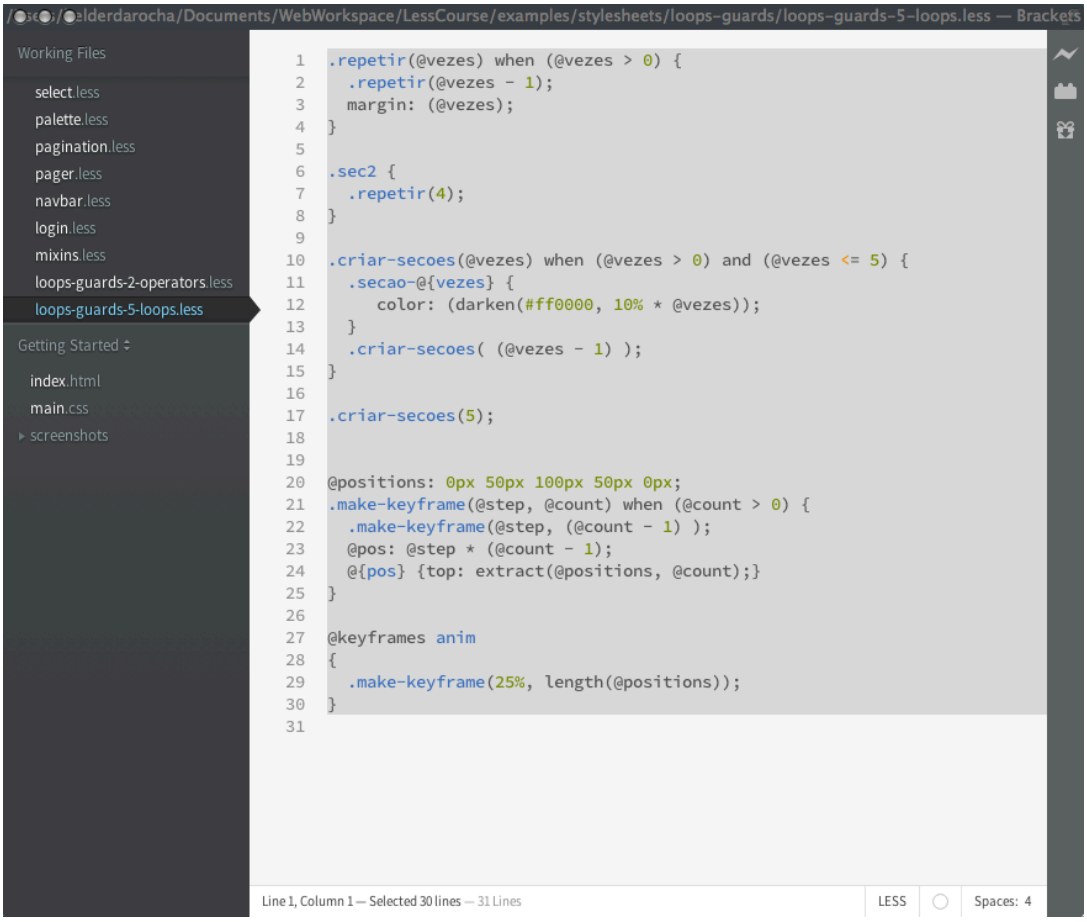
Também existem ferramentas para outras plataformas como Linux.

1.5 Ambientes de desenvolvimento

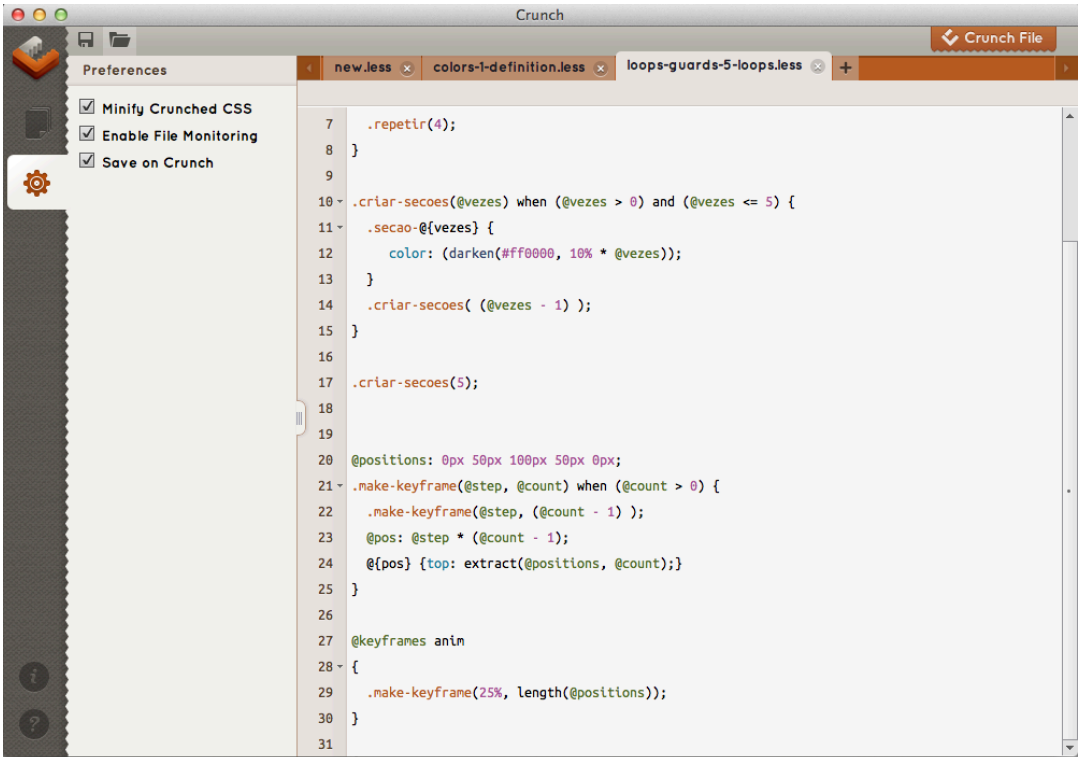
Você pode integrar o Less ao seu ambiente de desenvolvimento. No site oficial *lesscss.org* há uma lista de IDEs e ferramentas de Web Design que suportam Less, e vários outros que têm plugins que fazem a compilação automática.

Um ambiente integrado pode ser mais produtivo por ter recursos como busca e substituição, edição de HTML, depuração de JavaScript, que muitas vezes fazem parte de aplicações que usam Less.

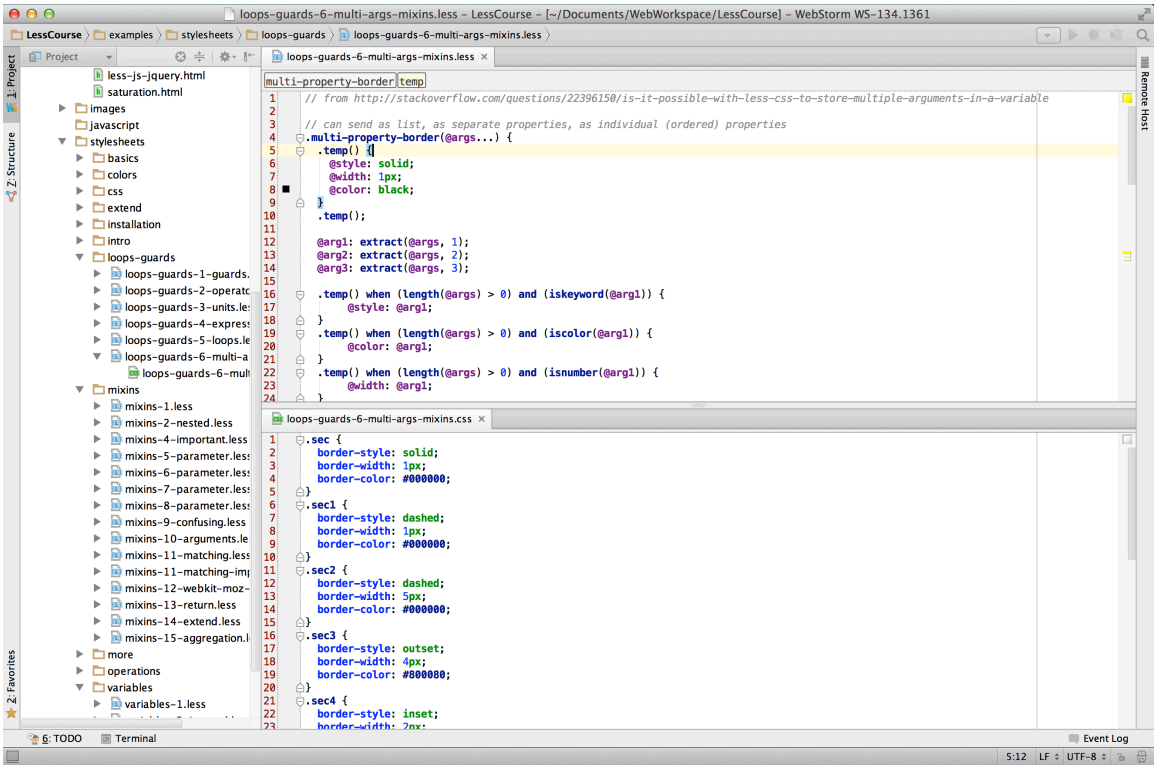
Um editor popular que tem suporte a Less é o *Adobe Brackets*, que é usado para aplicações Web em geral e é gratuito:



O *Crunch* é uma ferramenta completa que permite criar e editar documentos Less e também realiza a geração automática e compressão de CSS:



A imagem abaixo mostra uma tela do *JetBrains WebStorm* com código Less em uma janela superior, e código CSS sendo gerado em uma janela inferior.



Finalmente, se você precisa de uma ferramenta accessível de qualquer lugar pode usar o site *lesstester.com*, onde você pode colar o conteúdo de um arquivo `.less` e obter o resultado da compilação em CSS.

1.5.1 Teste do ambiente

Escolha o seu ambiente de desenvolvimento preferido, e teste a sua instalação criando o arquivo de testes abaixo:

```
@cor-principal: #FF8800;
#secao {
    color: @cor-principal;
    border-color: @cor-principal;
    background: lighten(@cor-principal, 50%);
    a:hover {
       color: darken(@cor-principal, 30%);
    }
}
```

Grave o arquivo em uma pasta e use uma das ferramentas listadas acima para gerar o CSS. O resultado deve ser semelhante ao CSS abaixo:

```
#secao {
   color: #ff8800;
   border-color: #ff8800;
   background: #ffffff;
}

#secao a:hover {
   color: #663600;
}
```

1.6 Compilação em tempo real

O programa `lessc` e as ferramentas acima listadas são pré-processadores estáticos. Isto quer dizer que eles compilam o Less gerando um arquivo CSS estático que deve ser carregado em suas páginas Web.

Você também pode usar folhas de estilo Less diretamente no HTML para processamento pelo browser. Como já foi mencionado, essa alternativa é lenta e ineficiente para uso em produção, mas pode ser útil em desenvolvimento.

Para usar Less no browser, é preciso carregar o script `less.js`, que pode ser baixado de https://github.com/less:

```
<script src="less.js" type="text/javascript"></script>
```

Pode-se também usar o `less.js` disponível em um repositório de código na nuvem (CDN):

```
<script
    src="http://cdnjs.cloudflare.com/ajax/libs/less.js/1.7.0/less.js">
</script>
```

Agora pode-se incluir folhas de estilos Less diretamente usando:

```
<link rel="stylesheet/less" type="text/css" href="teste.less" />
```

Mas é importante que as folhas de estilo Less sejam carregadas **antes** do script.

Pode-se carregar vários documentos Less. Cada um será compilado individualmente, portanto não é possível compartilhar variáveis e outros

recursos do Less entre eles (apenas recursos do CSS). O CSS resultante seguirá as mesmas regras de cascade, na ordem em que os estilos Less foram declarados.

1.7 Versões e alternativas

Este curso foi preparado e testado com a versão 1.7 do Less. O Less não possui uma especificação formal e toda a documentação está disponível no site e no GitHub do projeto. Muitos detalhes da linguagem não estão documentados. Se você estiver usando uma versão anterior do Less, vários exemplos poderão não funcionar. Se estiver usando uma versão mais nova, alguns comandos poderão ser diferentes ou até mais simples.

SASS e *Stylus* são outros dois pré-processadores de CSS que têm objetivos semelhantes a Less. Ambos têm sintaxes que são extensões do CSS como o Less, mas também suportam sintaxes mais compactas, eliminando chaves e ponto-e-vírgula. SASS é o mais popular dos dois e a sua sintaxe compatível com CSS (SCSS) tem muitas semelhanças com Less.

{2: aninhamento }

Nesta seção abordaremos um aspecto da sintaxe do Less que o distingue do CSS: a possibilidade de aninhar declarações.

2.1 Less é CSS, CSS não é Less

Os recursos do Less são *todos* opcionais. Less é uma *extensão* do CSS. Sempre pode-se usar CSS puro em uma página Less (e isto deve ser feito, se for mais simples e eficiente). Tudo que é CSS também é Less.

Por outro lado, CSS não é Less. Less é usado para gerar um CSS novo, que poderá ficar maior e mais repetitivo que o documento original Less.

Pode-se criar uma folha de estilos Less partindo-se de uma folha de estilos CSS existente. A forma mais simples é mudar a extensão do arquivo para `.less`. Quando o arquivo for lido pelo processador Less será gerado um CSS funcionalmente idêntico. Ele talvez substitua algumas representações por outras equivalentes (por exemplo, `rgb(255,0,0)` será substituído por `#ff0000`), mas o arquivo gerado será equivalente.

A partir de um documento Less contendo CSS puro, pode-se fazer alterações gradualmente, aplicando recursos do Less como *aninhamento de declarações*, variáveis, funções, etc. Este processo é chamado de refatoração: ele altera a forma de se expressar regras de estilo sem alterar o resultado. A melhor maneira de fazer refatoração é usando um editor que gere automaticamente o CSS, como os que foram apresentados no capítulo anterior.

2.2 Aninhamento de declarações

Less permite que declarações CSS sejam *aninhadas*. É uma forma alternativa de expressar relacionamentos contextuais entre seletores. Por exemplo, as duas declarações a seguir em CSS:

```
.artigo {
   border: solid red 1px;
   font-size: 12pt;
}
.artigo .secao1 {
   background-color: black;
   color: white;
}
.artigo .secao2 {
   background-color: white;
   color: black;
}
```

Podem ser combinadas em uma única declaração aninhada em Less, que especifica seus relacionamentos contextuais através do aninhamento:

```
.artigo {
   border: solid red 1px;
   font-size: 12pt;
   .secao1 {
     background-color: black;
     color: white;
   }
   .secao2 {
     background-color: white;
     color: black;
   }
}
```

Não há limites para o nível de aninhamento:

```
.livro {
   font-size: 10pt;
   .capitulo {
     border: solid gray 1px;
     .secao {
       background-color: #eee;
       .titulo {
         font-size: 120%;
       }
       .corpo {
         text-align: justify;
       }
     }
   }
}
```

A declaração aninhada acima resulta em cinco declarações:

```
.livro {
   font-size: 10pt;
}
.livro .capitulo {
   border: solid gray 1px;
}
.livro .capitulo .secao {
   background-color: #eee;
}
.livro .capitulo .secao .titulo {
   font-size: 120%;
```

```
}
.livro .capitulo .secao .corpo {
   text-align: justify;
}
```

O(s) seletor(es) declarados em um bloco externo são copiados antes de cada seletor de um bloco interno. Por exemplo, considere o bloco abaixo:

```
.t1 p {
   .t2, .t3 li {
     color: blue;
   }
}
```

Os seletores `.t2` e `.t3 li` serão ambos reescritos em CSS como seletores contextuais descendentes de `.t1 p`:

```
.t1 p .t2, .t1 p .t3 li {
   color: blue;
}
```

Qualquer tipo de contexto hierárquico pode ser representado, como filho (*child selector)* `>`, irmão adjacente (*adjacent sibling selector*) (`+`), etc. como na regra abaixo:

```
.v1 {
    + .v2 {
    > .v3 {
       font: 12pt;
     }
   }
}
```

que gera este CSS:

```
.v1 + .v2 > .v3 {
   font: 12pt;
}
```

O uso de declarações aninhadas é *opcional*. Geralmente elas aumentam a legibilidade e facilitam a manutenção mantendo juntas as regras aplicadas em seletores que têm relacionamentos hierárquicos que refletem a estrutura do HTML. Mas não devem ser usadas se uma declaração CSS comum expressar a aplicação de um estilo de forma *mais clara e concisa*. Por exemplo, se um relacionamento contextual longo aparece uma única vez em CSS.

A declaração CSS:

```
div .notes table .footnotes p {
   color: red;
}
```

é bem mais concisa e clara que:

```
div {
   .notes {
     table {
       .footnotes {
         p {
           color: red;
         }
       }
     }
   }
}
```

Mas se houver necessidade de aplicar regras a seletores intermediários da hierarquia, o uso de seletores aninhados pode ser mais vantajoso. O importante é sempre avaliar os benefícios de usar ou não declarações aninhadas em Less.

2.3 Comentários

Less é CSS, logo suporta comentários de bloco CSS, no formato `/* ... */`:

```
/* Este bloco de texto e as duas últimas propriedades
   do ruleset abaixo serão ignoradas pelo browser */
.header {
       color: cyan;
   /*  background: navy;
       font-size: 16pt;   */
}
```

Mas também suporta *comentários de linha*, usando duas barras `//`. Este tipo de comentário permite omitir trechos de código da geração do CSS final, e não contribui para aumentar o tamanho do arquivo CSS. O comentário inicia em `//`, afeta apenas a linha em que foi definido a partir desse ponto, e só termina no fim da linha. Não é possível terminá-lo antes.

```
// No CSS, o bloco a seguir terá apenas uma declaração de estilo
// Também não terá esta linha, nem a anterior
/* Mas estas linhas estarão presentes no CSS,
   embora sejam ignoradas pelo browser */
.header {
   color: cyan;
   // background: navy;
}
```

Apenas os comentários de bloco são preservados no CSS final:

```
/* Mas estas linhas estarão presentes no CSS,
   embora sejam ignoradas pelo browser */
.header {
   color: cyan;
}
```

2.4 O símbolo '&' (seletor pai)

Vimos que o aninhamento cria seletores contextuais ao posicionar os seletores de um bloco interno como *descendentes* dos seletores do bloco externo. O símbolo `&` é usado para representar o acumulado de seletores do bloco externo.

Considere a regra abaixo:

```
.t1 {
   & .t2 {
      & .t3 {
         font: 12pt;
      }
   }
}
```

Ela irá gerar o seguinte CSS:

```
.t1 .t2 .t3 {
   font: 12pt;
}
```

Que é o mesmo gerado se o `&` não estivesse presente.

```
.t1 {
   .t2 {
     .t3 {
       font: 12pt;
     }
   }
}
```

Mas podemos usar o `&` para *posicionar* os seletores acumulados nos blocos externos em outro lugar.

Na regra abaixo invertemos a ordem de contexto:

```
.s1 {
   .s2 & {
      .s3 & {
       font: 12pt;
     }
   }
}
```

E obtemos o seguinte resultado:

```
.s3 .s2 .s1 {
   font: 12pt;
}
```

Se o símbolo `&` é usado uma vez, ele *substitui* a posição dos seletores herdados, mas se ele aparecer *mais de uma vez*, ele *repete* esses seletores, por exemplo:

```
.r1 {
   & & .r2 {
     font: 12pt;
   }
}
```

que tem como resultado:

```
.r1 .r1 .r2 .r1 {
   font: 12pt;
}
```

2.4.1 Aninhamento com pseudo-seletores

O aninhamento não se limita a seletores contextuais. É possível usar o símbolo `&` em classes, pseudo-classes e pseudo-elementos.

O exemplo abaixo aplica aplica dois pseudo-elementos em `.tag`

```
.tag {
   &:before {Content: '&lt;'}
   &:after {Content: '/&gt;'}
}
```

Resultado em CSS:

```
.tag:before {
   Content: '&lt;';
}

.tag:after {
   Content: '/&gt;';
}
```

A regra abaixo aplica uma pseudo-classe no seletor `div p` e uma classe (`.last`) no seletor `div`:

```
div {
   p {
     &:first-child {color: red;}
   }
   &.last {color: blue;}
}
```

Produzindo este resultado em CSS:

```
div p:first-child {
   color: red;
}

div.last {
   color: blue;
}
```

2.4.2 Seletores com nomes semelhantes

O seletor `&` pode ser usado para criar novos seletores que repetem parte do nome, como por exemplo em seções numeradas:

```
.secao {
  &-1 {
    background-color: lightgreen;
  }

  &-2 {
    background-color: lightblue;
  }
}
```

O resultado em CSS do código acima é:

```
.secao-1 {
  background-color: lightgreen;
}

.secao-2 {
  background-color: lightblue;
}
```

2.4.3 Concatenação com &

Usando `&` copia o seletor, mas se houver muitos seletores separados por vírgula, como se comporta o seletor `&`?

```
.a, .b, .c {
  & > & {
    color: darkgreen;
  }

  & + & {
    color: olive;
  }
}
```

Ele se multiplica! Aplica o bloco de regras a todas as combinações possíveis entre os seletores do bloco externo. O código acima gera este CSS:

```
.a > .a, .a > .b, .a > .c,
.b > .a, .b > .b, .b > .c,
.c > .a, .c > .b, .c > .c {
  color: darkgreen;
}

.a + .a, .a + .b, .a + .c,
.b + .a, .b + .b, .b + .c,
.c + .a, .c + .b, .c + .c {
  color: olive;
}
```

Pode-se também aplicar classes aos seletores:

```
.a, .b {
  && & {
    color: darkred;
  }
}
```

O código acima produz oito combinações diferentes envolvendo contexto e classe, como mostra o resultado em CSS:

```
.a.a .a, .a.a .b, .a.b .a,
.a.b .b, .b.a .a, .b.a .b,
.b.b .a, .b.b .b {
  color: darkred;
}
```

{3: variáveis }

Variáveis permitem armazenar valores que serão reusados muitas vezes. É muito comum que certos valores como cores, fontes, dimensões, etc. sejam repetidas várias vezes em uma folha de estilo CSS. As variáveis do Less permitem eliminar essa repetição. É possível ainda realizar *operações* com variáveis, passá-las como argumentos para funções, etc.

Variáveis devem ter a seguinte sintaxe:

```
@nome: valores;
```

O `nome` pode ser qualquer nome que não seja uma diretiva do CSS (por exemplo, não pode ser `import` ou `media`). Os `valores` podem quaisquer valores ou listas de valores válidas em CSS ou mesmo expressões sobre valores ou variáveis.

3.1 Variáveis globais

Uma variável pode ser *global* ou ter o escopo limitado a um *bloco*. Variávels globais devem ser declaradas *fora* de blocos, em uma linha própria. Veja alguns exemplos:

```
@tipologia: "Times New Roman", "Times", serif;
@largura: 15em;
@cor-base: rgb(255, 255, 255);
@altura: @largura * 2;
```

Uma vez declarada, uma variável global pode ser usada no lugar de valores de propriedades CSS em qualquer lugar do arquivo:

```
h1 {
    font-family: @tipologia;
    height: @altura;
    width: @largura;
}
```

O CSS gerado será

```
h1 {
   font-family: "Times New Roman", "Times", serif;
   height: 30em;
   width: 15em;
   color: #ffffff;
}
```

As variáveis globais podem ser declaradas em qualquer parte do documento Less, mesmo *depois* dos blocos que as utilizam. A ordem em que são declaradas é irrelevante. Mas é uma boa prática adotar uma ordem seqüencial e mantê-las no início do documento para facilitar a legibilidade e manutenção do código.

3.2 Escopo e variáveis locais

As chaves que delimitam blocos limitam o escopo das variáveis. Se uma variável é definida dentro de um bloco ela é *local* ao bloco. O valor dela vale para todo o bloco e para os blocos que estejam aninhados *em um nível mais baixo*. O valor definido para uma variável declarada dentro de um bloco não é visível em blocos externos ou no contexto global.

No exemplo abaixo há variáveis em três níveis de escopo. `@global` pode ser usada em qualquer lugar do documento. `@local-1` foi definida dentro do contexto do bloco `header`, então só pode ser usada *dentro* desse bloco e em blocos aninhados (como o `div`). `@global` pode ser usada em qualquer bloco aninhado. Finalmente em `div` há uma variável `@local-2` definida, que não existe fora desse bloco.

```
@global: 'Global';
header {
   @local-1: 'Local ao header';
   content: @global, @local-1;
   div {
     @local-2: 'Local ao div';
     content:@global, @local-1, @local-2;
   }   // aqui não existe @local-2
} // aqui não existe local-1 ou local-2
footer {
   content: @global;
}
```

O CSS resultante desse documento Less é:

```
header {
   content: 'Global', 'Local ao header';
}
header div {
   content: 'Global', 'Local ao header', 'Local ao div';
}
footer {
   content: 'Global';
}
```

O processador Less exibe uma mensagem de erro e não gera o CSS se uma variável for usada fora do contexto onde ela é declarada. Mas é possível *declarar* variáveis em um contexto global com um valor qualquer, e *redefinir* essa variável em um contexto local declarando-a novamente (mesmo nome) com outro valor. Nesse caso, o novo *valor* da variável será válido apenas nesse contexto. Quando o bloco onde o valor foi redefinido terminar, a variável volta para o seu valor antigo:

```
@global: 'Global';
article {
    @global: "Local ao article";
    content: @global;
    section {
      content: @global;
    }
}
aside {
   content: @global;
}
```

No exemplo acima, a variável `@global` foi redefinida no bloco `article`, e o novo valor vale não apenas para todas as vezes que a variável for usada dentro do bloco, mas também em qualquer bloco *aninhado*, como `section`. O bloco `aside` está fora do bloco portanto vê o valor original de `@global`, pois a redefinição tem escopo limitado pelo bloco.

```
article {
   content: "Local ao article";
}
article section {
   content: "Local ao article";
}
aside {
   content: 'Global';
}
```

Assim como ocorre no documento com variáveis globais, a *ordem* das declarações dentro de cada bloco é irrelevante. Você pode declarar uma variável no final de um bloco, que o valor dela valerá para o bloco inteiro e todos os sub-blocos, mesmo que tenham sido declarados antes. Somente o *aninhamento* dos blocos limita o escopo de variáveis. A ordem no mesmo bloco não afeta o escopo.

O exemplo anterior poderia ter sido escrito da forma abaixo e geraria o mesmo CSS:

```
article {
    content: @global;
    section {
      content: @global;
    }
   @global: "Local ao article";
}
```

```
aside {
   Content: @global;
}

@global: 'Global';
```

É importante entender esse funcionamento para não ter surpresas. É igualmente importante evitar escrever documentos Less assim, pois eles ficam menos legíveis para quem precisa mantê-los.

3.3 Variáveis interpoladas

Variáveis não servem apenas para guardar valores de propriedades. Podem ser usadas para conter outros textos, tais como nomes de propriedades, seletores, URLs, etc. Neste caso a *declaração* continua igual, mas para *usar* a variável é preciso usar uma sintaxe diferente com o nome da variável (sem o `@`) entre chaves da forma `@{variavel}`.

O exemplo abaixo declara quatro variáveis que são usadas de diversas maneiras diferentes usando interpolação:

- Como seletor: variável `@classe`;
- Como nome de uma propriedade CSS: variável `@transparencia`;
- Como parte do nome de uma propriedade: variável `@prop`;
- Como parte do string usado em uma URL: variável `@diretorio`;

```
@transparencia: opacity;
@classe: coluna;
@diretorio: "../images";
@prop: color;

.@{classe} {
   @{transparencia}: 0.5;
   border-@{prop}: red;
   background: url('@{diretorio}/paisagem.png');
}
```

Observe que o uso de variáveis interpoladas em URLs deve ser *dentro* das aspas (ou apóstrofes). O resultado em CSS do Less acima é:

```
.coluna {
   opacity: 0.5;
   border-color: red;
   background: url('../images/paisagem.png');
}
```

Uma variável também pode conter o nome de outra variável simplesmente incluindo mais um `@` antes do nome (que ficará `@@nome`):

```
@the-end: "this is the end";
@end: "the-end";
```

```
.my-friend {
   content: @@end;
}
```

O resultado será:

```
.my-friend {
    content: "this is the end";
}
```

Não se pode incluir mais que dois `@` seguidos.

{4: extend }

4.1 :extend

`:extend` é uma pseudo-classe exclusiva do Less que copia o conjunto de regras do seletor no qual é aplicado, aos conjuntos de regras de *outros seletores* passados como parâmetro. Evita que seletores sejam repetidos muitas vezes.

Vejamos um exemplo simples. Considere a folha de estilos abaixo, em CSS puro:

```
.capitulo {
   font: 12pt;
   margin: 2px;
}
.secao1 {
   content: 'Secao 1';
}
```

Para que os estilos de `capitulo` tambem sejam aplicados em `.secao1`, usando CSS podemos estender o seletor `.secao1` com as regras de `capitulo` desta forma:

```
.capitulo, .secao1 {
   font: 12pt;
   margin: 2px;
}
```

A pseudo-classe `:extend` do Less oferece uma forma alternativa de obter esse resultado, incluindo essa informação em um seletor que herda as declarações de estilo dos seletores passados como parâmetro.

Em Less para obter o resultado acima pode-se fazer:

```
.secao1:extend(.capitulo) {
   content: 'Secao 1';
}
```

O resultado prático para a classe `.secao1` será:

```
.secao1 {
   content: 'Secao 1';
   font: 12pt;
   margin: 2px;
}
```

Mas a geração do CSS inteiro separará esse resultado em duas declarações, para evitar a repetição das regras de `.capitulo`:

```
.capitulo, .secao1 {
   font: 12pt;
   margin: 2px;
}

.secao1 {
   content: 'Secao 1';
}
```

4.2 Extensão múltipla

`:extend` pode ser aplicada a vários seletores, e incluirá regras de todos eles.

No seguinte exemplo `secao1`, `secao2` e `livro2` tem um conjunto de regras *vazio*. O seletor `secao1` será estendido com as mesmas regras de `.capitulo`, `secao2` será estendido com as regras que foram definidas para `secao1` mais as de `.capitulo`. `livro` terá a combinação das regras de `div` e `header`:

```
div, .capitulo, #copy {
   font: 12pt;
   margin: 2px;
}

.prefacio, header {
   color: #111;
   background: #eee;
}

.livro {
   padding: 2px;
}

.secao1:extend(.capitulo) {}
.secao2:extend(.secao1, .capitulo) {}
.livro:extend(div, header) {}
```

A regra de extensão aplicada a `secao2` também pode ser escrita como:

```
.secao2:extend(.secao1):extend(.capitulo) {}
```

Ainda haverá muita repetição se for necessário aplicar a mesma regra de extensão a vários seletores:

```
.secao1:extend(.capitulo, .prefacio),
.secao2:extend(.capitulo, .prefacio),
.secao3:extend(.capitulo, .prefacio) {}
```

Less resolve esse problema usando o símbolo `&`, que representa o conjunto dos seletores acumulados no bloco externo. Assim é possível aplicar a mesma regra de extensão a vários seletores usando um único `:extend`, obtendo o mesmo resultado do bloco acima:

```
.secao1, .secao2, .secao3 {
   &:extend(.capitulo, .prefacio);
}
```

O bloco acima também pode ser escrito da forma abaixo:

```
.secao1, .secao2, .secao3 {
   &:extend(.prefacio);
   &:extend(.capitulo);
}
```

Que tem como resultado em CSS:

```
.capitulo, .secao1, .secao2, .secao3 {
   font: 12pt;
   margin: 2px;
}
.prefacio, .secao1, .secao2, .secao3 {
   color: #111;
   background: #eee;
}
```

4.3 Extensão com seletores aninhados

Mais um exemplo com `:extend` usando seletores contextuais e aninhamento. O código abaixo:

```
s1 d1, s2 {
    &:extend(s3, s4 d2)
}
```

Pode também ser escrito como:

```
s1 d1:extend(s3, s4 d2), s2:extend(s3, s4 d2) {}
```

ou:

```
s1 d1:extend(s3):extend(s4 d2), s2:extend(s3):extend(s4 d2) {}
```

ou ainda:

```
s1 d1:extend(s3) {}
s1 d1:extend(s4 d2) {}
s2:extend(s3) {}
s2:extend(s4 d2) {}
```

Supondo que também existam as seguintes declarações, nas quais aparecem os seletores `s3` e `s4 d2` (observe que o contexto `s4 d2` revela-se no aninhamento):

```
t1, t3, s3, t4 {color: blue}
n1, s4 {
   content: 'nothing';
   d2 {
     font-size: 12pt
   }
}
```

O CSS resultante será

```
t1, t3, s3, t4, s1 d1, s2 {
   color: #0000ff;
}
n1 d2, s4 d2, s1 d1, s2 {
   font-size: 12pt;
}
n1, s4 {
   content: 'nothing';
}
```

Os blocos vazios, se não tiverem propriedades, são removidos.

4.4 Sobreposição de propriedades

`:extend` não impede a criação de propriedades duplicadas. Elas podem ocorrer no mesmo bloco ou em blocos diferentes aplicadas ao mesmo seletor. As regras sobre qual terá precedência são as mesmas do CSS: dentro de um mesmo bloco, se houver duas declarações afetando a mesma propriedade, vale a que foi definida por último; dentro do mesmo documento, propriedades aplicadas a seletores idênticos em blocos diferentes, também vale a última.

No exemplo abaixo o seletor `.n7` tem `background: orange`, mas também tem dois `:extend` que irão fazer com que o seletor `.n7` seja repetido nos blocos `.n1` e `.n2`. Não faz diferença a ordem em que os `&:extend` aparecem no primeiro bloco, mas a posição dos blocos `.n2` e `.n1`. O `&:extend(.n2)` foi chamado após `&:extend(.n1)`, porém no documento, o bloco em que aparece `.n1` é o último.

```
.n7 {
   &:extend(.n1);
   background: orange;
   &:extend(.n2);
}
.n2 {
   background: green;
}
.n1 {
   color: yellow;
   background: red;
```

```
}
```

`.n7` será estendido com o conteúdo de `.n1` e `.n2` da seguinte forma no CSS gerado:

```
.n7 {
   background: orange;
}
.n2, .n7 {
   background: green;
}
.n1, .n7 {
   color: yellow;
   background: red;
}
```

Como todos os três blocos definem a propriedade background para o mesmo seletor `.n7`, apenas a última definição será preservada, e `.n7` terá a propriedade `background: red`.

Less também não impede ou detecta duplicação se seletores forem estendidos múltiplas vezes. No terceiro bloco abaixo, `.mast-head` estende `.head` e `.title`, mas como `.head` já estendia `.title`, `.mast-head` será copiado duas vezes:

```
.title {
   color: black;
}

.head {
   &:extend(.title);
}

.mast-head:extend(.head, .title) {}
```

Este é o resultado:

```
.title, .head, .mast-head, .mast-head {
   color: black;
}
```

Isto não altera o funcionamento do CSS, embora seja uma duplicação desnecessária.

4.5 Pseudo-elementos e variáveis

Um seletor pode ter outras pseudo-classes além de `:extend` , mas `:extend` tem sempre que ser a *última*:

```
pre:nth-child(odd):extend(div span) {}  // OK!
a:hover:extend(div):extend(section) {} // OK!
a:extend(div):hover {} // ILEGAL - causa erro no processador!
```

`:extend` pode ser anexado a um seletor representado por uma variável interpolada:

```
@var: .secao; @{var}:extend(div) {}
```

Mas `:extend` não suporta (até o Less 1.6) variáveis como argumentos. Elas são ignoradas já que o processador não consegue achar o seletor representado pela variável. Não é mostrada mensagem de erro e a falha ocorre silenciosamente.

```
@var: .secao; div:extend(@{var}) {} // ISTO NAO FUNCIONA!
```

4.6 Correspondência exata

Para aplicar as extensões, `:extend` precisa localizar seletores que combinem com os que foram passados como argumentos. Por *default*, essa correspondência precisa ser *exata*. A forma importa. Dois seletores diferentes que têm o mesmo efeito ou significado em CSS. Por exemplo `p:before:hover` e `p:hover:before` *não são considerados iguais* para `:extend`. A única exceção é conteúdo entre aspas em predicados:

`[nome=abc]`, `[nome='abc']` e `[nome="abc"]`

são considerados equivalentes.

Tipos de expressões com seletores que são equivalentes em CSS mas não são consideradas correspondências equivalentes em parâmetros de `:extend` incluem:

- Seletores equivalentes que usam ou omitem o seletor universal: `*.classe` e `.classe`, `*:before` e `:before` ou [`nome=abc`] e `*[nome=abc]`
- Seletores com pseudo-elementos equivalentes com ordem diferente: `a:before:hover` e `a:hover:before`
- Seletores do tipo `nth-` equivalentes com argumentos expressos de forma textualmente diferente: `a:nth-child(n+1)`, `a:nth-child(1n+1)`, `a:nth-child(odd)` e `a:nth-child(n + 1)`

Mesmo tendo usando como argumento quatro seletores equivalentes, `div` não será estendido nos blocos abaixo:

```
a:before:hover, p:nth-child(n + 1), *.secao, *[nome] {
   text-decoration: underline;
}

div:extend(a:hover:before, p:nth-child(n+1), .secao, [nome]) {
   color: purple;
}
```

e o resultado será:

```
a:before:hover, p:nth-child(n + 1), *.secao, *[nome] {
   text-decoration: underline;
}

div {
   color: purple;
}
```

Mas se for adicionado um espaço antes e outro depois do `+`, a combinação exata de `p:nth-child(n + 1)` será suficiente para fazer `div` aparecer no primeiro bloco:

```
a:before:hover, p:nth-child(n+1), *.secao, *[nome], div {
   text-decoration: underline;
}
div {
   color: purple;
}
```

4.7 Correspondência parcial

Se um seletor do tipo `.a` for usado como argumento de `:extend`, ele encontrará correspondência apenas com seletores idênticos `.a`. Seletores `c.a` ou `.a.b` não serão considerados equivalentes.

O seletor `.new-section abaixo` não será estendido pois `.sec` não tem correspondência *exata* com nenhum dos seletores do primeiro bloco:

```
div.sec, .sec.subsec, .sec:before {
   color: pink;
}

.new-section:extend(.sec) {}
```

Resultado:

```
div.sec, .sec.subsec, .sec:before {
   color: pink;
}
```

Mas Less irá aceitar essas correspondências se o argumento de `:extend` vier seguido da palavra `all`:

```
.new-section:extend(.sec all) {}
```

Assim ele vai gerar o CSS abaixo:

```
div.sec, .sec.subsec, .sec:before, div.new-section, .new-section.subsec,
.new-section:before {
   color: pink;
}
```

4.8 Escopo

O escopo de `:extend` vale para toda a página e blocos aninhados. Se for usado dentro de um bloco `@media`, ele só poderá estender seletores declarados dentro do mesmo bloco ou em blocos aninhados.

O seletor `body:extend(.page)` irá estender `body` em todos os blocos abaixo, mas `div:extend(.page)` usado dentro do primeiro bloco `@media` irá estender `div` apenas no bloco `.page` contido no mesmo bloco `@media`:

```
.page {
   font-size: 20px;
   width: 100%;
   height: 100%;
}

@media (min-width: 48rem) {
   .page {
     font-size: 18px;
   }
   div:extend(.page) {}
}

@media print {
   .page {
     font-size: 12px;
   }
}

body:extend(.page) {};
```

Resultado em CSS:

```
.page, body {
   font-size: 20px;
   width: 100%;
   height: 100%;
}

@media (min-width: 48rem) {
   .page, div, body {
     font-size: 18px;
   }
}

@media print {
   .page, body {
     font-size: 12px;
   }
}
```

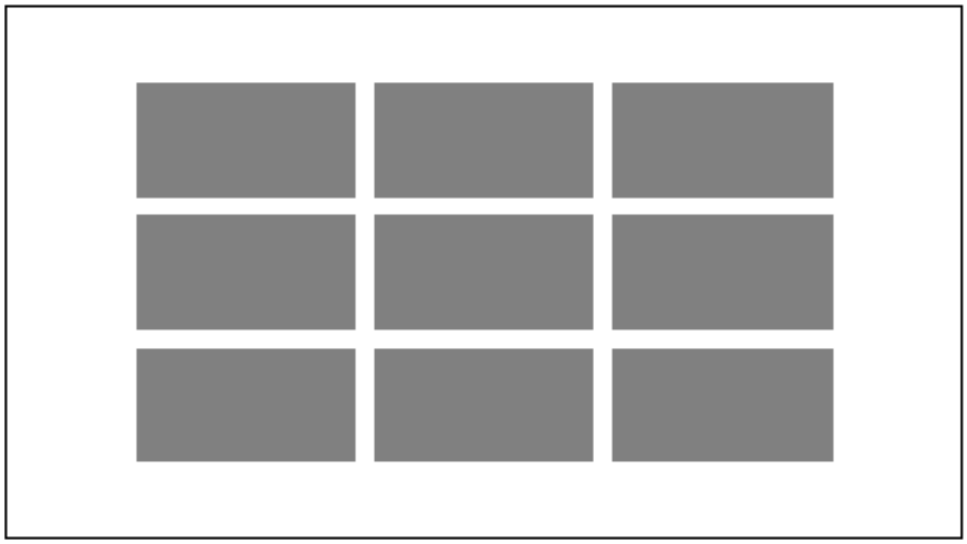
4.9 Quando usar :extend

Com `:extend` pode-se evitar a criação de uma classe no HTML somente para aplicar propriedades gerais que devem ser usadas por um grupo de elementos.

Por exemplo, suponha que você tenha uma folha de estilos que defina um seletor para posicionamento *default* centralizado (`.abs`) e outro para construir retângulos 100x50 cinzas (`.box`) como mostrado abaixo.

```
.abs {
   position: absolute;
   top: 0; left: 0; bottom: 0; right: 0;
   margin: auto;
}
.box {
   background-color: gray;
   height: 50px;
   width: 100px;
}
```

Você tem 9 `div`s dentro de uma section e pretende renderizar cada `div` como um `.box` e posicionar da forma abaixo:



Então você pode utilizar esta folha de estilos:

```
@import url('abs-box.less');
@up: 50%;
@down: -@up;
@right: 50%;
@left: -@right;

section {
   width: 450px;
   height: 250px;
   border: solid black 1px;
   position: relative;
}
```

```
.top {
   top: @up;
}
.bottom {
   top: @down;
}
.left {
   left: @left;
}
.right {
   left: @right;
}
```

E aplicar as classes nos seus `div`s de acordo com a posição desejada para cada retângulo:

```
<section>
    <div class="box abs top left"></div>
    <div class="box abs top"></div>
    <div class="box abs top right"></div>
    <div class="box abs left" ></div>
    <div class="box abs"></div>
    <div class="box abs right"></div>
    <div class="box abs bottom left" ></div>
    <div class="box abs bottom"></div>
    <div class="box abs bottom right"></div>
</section>
```

Há muita repetição desnecessária. Se cada classe `.top`, `.right`, etc. sempre vai ser um `.box` e um `.abs`, podemos herdar os estilos desses seletores e evitar repetir `"box abs"` em cada classe.

Então acrescentamos o bloco seguinte, estendendo com as propriedades de `.box` e `.abs` os quatro seletores de posicionamento, e mais um com bloco vazio que simplesmente herda os estilos (`.center`):

```
.right, .top, .bottom, .left, .center {
   &:extend(.abs, .box);
}
```

Com isto, não precisamos mais repetir as classes `.box` e `.abs` em cada `div`, mantendo apenas as classes de posicionamento e deixando o código mais limpo:

```
<section>
    <div class="top left"></div>
    <div class="top"></div>
    <div class="top right"></div>
    <div class="left" ></div>
    <div class="center"></div>
    <div class="right"></div>
    <div class="bottom left" ></div>
    <div class="bottom"></div>
    <div class="bottom right"></div>
</section>
```

{5: mixins }

5.1 Mixins

Variáveis permitem armazenar valores. Mixins permitem armazenar conjuntos inteiros de regras, que podem ser reusadas em outros blocos.

A declaração de um mixin não difere em nada de uma declaração de um seletor CSS de classe ou id. A diferença é que o mixin foi criado para ser usado apenas na folha de estilos. Além disso, mixins podem ter parâmetros.

O mixin abaixo agrupa várias regras usadas em diferentes seções de um documento:

```
.text-content-set {
   padding: 2px;
   margin: 0 2px 0 2px;
   border: solid rgb(225,225,225);
   background-color: rgb(128,128,128);
   color: rgb(240,240,240);
}
```

Expresso desta forma, o CSS gerado é praticamente o mesmo (talvez mude a forma de representação de cores, dependendo do processador). O que faz esse bloco um mixin é a *forma* como ele é usado. Agora podemos aplicá-lo em vários outros blocos, e evitar repetir um monte de declarações:

```
.section {
    .text-content-set;
    h1 {
      color: #00008b;
    }
}
#footer {
   .text-content-set;  // este é o mixin
   background-color: white;
}
```

O resultado será:

```
.text-content-set {
   padding: 2px;
   margin: 0 2px 0 2px;
   border: solid #e1e1e1;
   background-color: #808080;
   color: #f0f0f0;
}
.section {
   padding: 2px;
   margin: 0 2px 0 2px;
   border: solid #e1e1e1;
   background-color: #808080;
   color: #f0f0f0;
}
.section h1 {
   color: #00008b;
}
#footer {
   padding: 2px;
   margin: 0 2px 0 2px;
   border: solid #e1e1e1;
   background-color: #808080;
   color: #f0f0f0;
   background-color: white;
}
```

Como não pretendemos que o mixin seja usado na página, podemos ocultá-lo na geração final do CSS declarando-o com parênteses:

```
.text-content-set() { ... }
```

Assim apenas os elementos que usam o mixin fazem parte do CSS final:

```
.section {
   padding: 2px;
   margin: 0 2px 0 2px;
   border: solid #e1e1e1;
   background-color: #808080;
   color: #f0f0f0;
}
.section h1 {
   color: #00008b;
}
#footer {
   padding: 2px;
   margin: 0 2px 0 2px;
   border: solid #e1e1e1;
   background-color: #808080;
   color: #f0f0f0;
   background-color: white;
}
```

É uma boa prática sempre usar parênteses na declaração de mixins, mesmo que eles não tenham parâmetros.

Mixins podem também usar seletores de id (`#nome`) em vez de seletores de classe (`.nome`). Não faz nenhuma diferença, mas a convenção é usar seletores de classes para mixins, e seletores de id para namespaces (que veremos mais adiante).

5.2 Mixins com blocos aninhados

Mixins podem conter blocos aninhados, por exemplo:

```
.mixin1() {
   .abc {
     color: black;
   }
   .xyz {
     color: gray;
   }
}
```

Isto vai criar incluir .abc e .xyz no contexto de qualquer seletor que usar o mixin:

```
.container {
   .mixin1();
}
```

Que resulta no CSS:

```
.container .abc {
   color: black;
}
.container .xyz {
   color: gray;
}
```

Pode-se criar outros tipos de relacionamento entre seletores, mas não é permitido usar `&` antes ou depois da *chamada* a um mixin:

```
.container {
   &.mixin1;  // ILEGAL!
   .mixin1 &;  // ILEGAL!
}
```

No entanto, é possível usar o símbolo & dentro de um mixin, permitindo que o seletor do bloco externo seja usado como parâmetro dentro do mixin:

```
.mixin2() {
   &.abc {
      color: blue;
   }
   &.xyz {
     color: navy;
   }
}
```

Desta vez a chamada:

```
.component {
   .mixin2();
}
```

irá passar o(s) seletor(es) do bloco onde o mixin foi chamado como argumento no lugar do &, adicionando um seletor de classe em cada um:

```
.component.abc {
   color: blue;
}
.component.xyz {
   color: navy;
}
```

5.3 !important

A palavra-chave `!important` é usada em CSS para sobrepor as regras de precedência do cascade e forçar a aplicação de uma declaração de estilo em um seletor. Se for aplicada em um mixin, todas as declarações de estilo contidas no mixin irão ser marcadas como `!important`.

Considere o mixin abaixo:

```
.mask-circle() {
   -webkit-mask-image: url(circle.svg);
   -webkit-mask-origin: padding;
   -webkit-mask-position: 450px 150px;
   -webkit-transform: scale(2);
}
```

Se ele for usado desta forma:

```
.photo {
   border: none;
   .mask-circle() !important;
}
```

Produzirá este resultado em CSS:

```
.photo {
   border: none;
   -webkit-mask-image: url(circle.svg) !important;
   -webkit-mask-origin: padding !important;
   -webkit-mask-position: 450px 150px !important;
   -webkit-transform: scale(2) !important;
}
```

5.4 Mixins com parâmetros

Mixins podem receber parâmetros na forma de variáveis com escopo limitado ao bloco. As variáveis têm seus valores definidos quando o mixin é usado. Este tipo de mixin é muito bom para reusar definições com valores diferentes.

No exemplo abaixo, um mixin para redimensionamento que recebe três parâmetros:

```
.scale-2D-2(@amount) {
   -webkit-transform-origin: top left;
   -webkit-transform: scale(@amount);
   -moz-transform-origin: top left;
   -moz-transform: scale(@amount);
   -ms-transform-origin: top left;
   -ms-transform: scale(@amount);
   -o-transform-origin: top left;
   -o-transform: scale(@amount);
   transform-origin: top left;
   transform: scale(@amount);
}
```

O mixin acima pode ser usado da forma:

```
.image2 {
   .scale-2D-2(.75);
}
```

e irá gerar o CSS abaixo:

```
.image2 {
   -webkit-transform-origin: top left;
   -webkit-transform: scale(0.75);
   -moz-transform-origin: top left;
   -moz-transform: scale(0.75);
   -ms-transform-origin: top left;
   -ms-transform: scale(0.75);
   -o-transform-origin: top left;
   -o-transform: scale(0.75);
   transform-origin: top left;
   transform: scale(0.75);
}
```

Um mixin pode ter múltiplos parâmetros separados por *ponto-e-vírgula* (vírgulas também podem ser usadas mas podem causar problemas em alguns casos). Os parâmetros são selecionados pela sua posição.

```
.pular(@duration; @count) {
   -webkit-animation-name: roteiro-pulo;
   -webkit-animation-duration: @duration;
   -webkit-animation-iteration-count: @count;
}
```

Uso do mixin:

```
.anim-pulando {
   .pular(4s; 3);
}
```

Resultado:

```
.anim-pulando {
   -webkit-animation-name: roteiro-pulo;
   -webkit-animation-duration: 4s;
   -webkit-animation-iteration-count: 3;
}
```

Pode-se definir o mixin com um valor *default* para cada variável, que será usado caso os parâmetros não sejam definidos no uso.

No exemplo abaixo o segundo parâmetro do mixin `.pulo-vertical`, `@timing-function` possui um valor *default* `ease-out` que será usado se o segundo parâmetro não estiver presente.

```
.pulo-vertical(@height; @timing-function: ease-out) {
   -webkit-transform: translateY(@height);
   -webkit-animation-timing-function: @timing-function;
}
```

O mixin é chamado três vezes. Na primeira e última o segundo parâmetro é omitido e o valor *default* é usado.

```
@-webkit-keyframes roteiro-pulo {
   0%   {.pulo-vertical(100px);}
   50%  {.pulo-vertical(50px; ease-in);}
   100% {.pulo-vertical(100px);}
}
```

Resultado em CSS:

```
@-webkit-keyframes roteiro-pulo {
   0% {
     -webkit-transform: translateY(100px);
     -webkit-animation-timing-function: ease-out;
   }
   50% {
     -webkit-transform: translateY(50px);
     -webkit-animation-timing-function: ease-in;
   }
   100% {
     -webkit-transform: translateY(100px);
     -webkit-animation-timing-function: ease-out;
   }
}
```

Mixins também podem ter parâmetros identificados por nomes. Os nomes sobrepõem a ordem dos elementos. Isto permite que mixins sejam chamados com *menos* parâmetros, já que os parâmetros são *identificados*.

O mixin abaixo possui três parâmetros com valores *default*:

```
.reflexo(@begin:left top; @end: left bottom; @final-color:white) {
   -webkit-box-reflect:
        below
        10px
       -webkit-gradient(linear, @begin, @end,
          from(transparent),
          color-stop(0.5, transparent),
          to(@final-color));
}
```

Pode-se chamar o mixin sem parâmetros ou com menos de três parâmetros, desde que os parâmetros sejam os primeiros. Para chamar o mixin com apenas o último parâmetro, é preciso identificá-lo:

```
.image3 {
   .reflexo(@final-color: black;);
}
```

Desta forma a ordem dos parâmetros não importa. O resultado em CSS será:

```
.image3 {
   -webkit-box-reflect:
           below
           10px
           -webkit-gradient(linear, left top, left bottom,
                  from(transparent),
                  color-stop(0.5, transparent),
                  to(#000000));
}
```

É uma boa prática usar mixins com parâmetros identificados, já que são mais legíveis.

5.5 Sintaxes alternativas (e problemáticas)

O Less também permite que se use *vírgula* para separar os parâmetros, mas ele se confunde quando os argumentos passados também contém vírgulas.

Considere o mixin abaixo:

```
.font-mix(@family, @size: 10pt) {
   font-family: @family;
   font-size: @size;
}
```

Se ele for chamado desta forma:

```
.section1 {
   .font-mix('Times', 12pt);
}
```

O compilador Less irá gerar o CSS abaixo, como esperado:

```
.section1 {
   font-family: 'Times';   font-size: 12pt;
}
```

Como o segundo argumento possui um valor *default*, é possível omiti-lo caso o valor de `10pt` seja desejado:

```
.section2 {
   .font-mix('Times');
}
```

E o resultado também será o esperado:

```
.section2 {
   font-family: 'Times';
   font-size: 10pt;
}
```

Mas, e se o mixin for chamado com uma lista de fontes?

```
.section3 {
   .font-mix('Times', sans-serif);
}
```

O compilador Less não vai reclamar, no entanto vai gerar o CSS incorreto:

```
.section3 {
    font-family: 'Times';
    font-size: sans-serif;
}
```

Se forem passados três ou mais parâmetros, o compilador não gera o CSS e exibe uma mensagem de erro, mas se o número de parâmetros for compatível, não acontece erro e o CSS é gerado incorretamente.

A solução é usar *ponto-e-vírgula* para separar e terminar os parâmetros nas chamadas. A chamada do mixin em `.section3` acima pode ser corrigida adicionando um *ponto-e-vírgula* no final:

```
.section3 {
   .font-mix('Times', sans-serif;);
}
```

Usando *ponto-e-vírgula*, agora pode-se passar mais parâmetros de fonte sem causar erro no compilador e obtendo o resultado esperado:

```
.section4 {
   .font-mix('Times', 'Times New Roman', sans-serif; 12pt;);
}
```

5.6 Mixins com número variável de argumentos

O número de parâmetros aceitos no uso de um mixin depende de como ele foi declarado. A tabela abaixo ilustra algumas situações. Os argumentos `@arg`, `@arg1` e `@argn` representam variáveis quaisquer e reticências "*...*" (representadas abaixo em caracteres proporcionais) significam zero ou mais argumentos.

Por outro lado, a variável `@arguments`, a variável `...` (três pontos, representados aqui em caracteres fixos) e `@`*variavel*`...` (nome de variável seguido de três pontos) são variáveis especiais do Less que serão abordadas nesta seção.

Sintaxe usada na declaraçao do mixin	No. args	Variáveis criadas
`.mixin` ou `.mixin()`	0	Nenhuma
`.mixin(@arg)`	1	`@arg` e `@arguments`
`.mixin(@arg1;` ... `; @argn)`	n	`@arg1`, ..., `@argn` e `@arguments`
`.mixin(@arg:` *valor*`)`	0..1	`@arg` e `@arguments`
`.mixin(@arg1:` *valor*; ... `;@argn:` *valor*`;)`	0..n	`@arg1`, ..., `@argn` e `@arguments`
`.mixin(...)`	0..*	`@arguments`
`.mixin(@arg; ...)`	1..*	`@arg` e `@arguments`
`.mixin(@arg1;` ... `; @argn; ...)`	n..*	`@arg1`, ..., `@argn` e `@arguments`
`.mixin(@arg:` *valor*`; ... )`	0..*	`@arg` e `@arguments`
`.mixin(@arg1; @arg2... )`	1..*	`@arg1` + `@arg2` = `@arguments`

Se o número de argumentos passados para um mixin for incompatível com o número esperado de argumentos (segundo a tabela acima), o compilador Less não irá gerar o CSS e exibirá uma mensagem de erro. Mas não há verificação de tipo de dados, portanto se os argumentos forem enviados na ordem errada, eles serão atribuídos a variáveis incorretas e não irão gerar o CSS esperado. Por essa razão é uma boa prática usar argumentos com nome, e valores *default* nos mixins com muitos argumentos.

5.6.1 A variável `@arguments`

`@arguments` é uma variável que existe em todos os mixins que aceitam argumentos, e contém *todos* os argumentos recebidos. Dentro do mixin pode-se tanto usar as variáveis individuais como todas juntas na ordem em que foram recebidas. As variáveis de `@arguments` podem ser usadas em loops ou diretamente nas propriedades. No CSS resultante elas serão *separadas por espaços*.

```
.borda (@estilo: solid; @cor: black; @espessura: 1px) {
   border: @arguments;
   background: lighter(@cor, 50%);
}
```

As duas classes abaixo usam este mixin. A primeira não passa parâmetros e usa os valores *default*. A segunda altera apenas um parâmetro:

```
.sec1 {
   .borda();
}
.sec2 {
   .borda(@cor: red);
}
```

Este é o resultado em CSS:

```
.sec1 {
   border: solid #000000 1px;
   background: #808080;
}

.sec2 {
   border: solid #ff0000 1px;
   background: #ffffff;
}
```

Este outro mixin pre-define vários argumentos para uma margem, inicializados com um valor default (`2px`):

```
.margem(@top: 2px; @right: 2px; @bottom: 2px; @right: 2px) {
   margin: @arguments;
}
```

Ele pode ser chamado sem argumentos para usar os valores *default*, ou pode escolher qualquer um ou mais parâmetros para redefinir. O seletor `.sec3` abaixo mudou apenas o terceiro argumento para zero:

```
.sec3 {
   .margem(@bottom: 0);
}
```

O mixin garante que a substituição em CSS ocorrerá no lugar correto:

```
.sec3 {
   margin: 2px 2px 0 2px;
}
```

5.6.2 A variável `...` (três pontos)

Mixins que recebem um número variável de argumentos podem ser declarados com reticências (três pontos) dentro dos parênteses:

```
.padding-e-margem(...) {
   margin:@arguments;
   padding: @arguments;
}
```

A chamada pode conter de zero a muitos argumentos:

```
.sec4 {
   .padding-e-margem(1;2);
}
```

O CSS resultante contém os argumentos na ordem em que foram enviados:

```
.sec4 {
    margin: 1 2;
    padding: 1 2;
}
```

Um problema desse mixin é que ele aceita que *não* sejam passados argumentos. Se ele for chamado desta forma:

```
.sec4 {
   .padding-e-margem;
}
```

O resultado será um CSS incorreto:

```
.sec4 {
   margin: ;
   padding: ;
}
```

Para evitar isto, pode-se declarar um ou mais parâmetros e usar `...` como *último* argumento. A variável `@arguments` irá conter *todos* os argumentos, inclusive os que têm variáveis declaradas.

O mixin abaixo garante que haverá *ao menos um* argumento válido:

```
.transmix(@transform: scale(1); ...) {
   -webkit-transform: @arguments;
   -moz-transform: @arguments;
   -o-transform: @arguments;
   -ms-transform: @arguments;
   transform: @arguments;
}
```

Se for chamado sem argumentos:

```
.sec5 {
   .transmix;
}
```

irá produzir:

```
.sec5 {
   -webkit-transform: scale(1);
   -moz-transform: scale(1);
   -o-transform: scale(1);
   -ms-transform: scale(1);
   transform: scale(1);
}
```

Mas se forem passados argumentos, eles terão precedência:

```
.sec6 {
   .transmix(scale(0.5); translate(3); rotateX(25deg));
}
```

Resultado:

```
.sec6 {
   -webkit-transform: scale(0.5) translate(3) rotateX(25deg);
   -moz-transform: scale(0.5) translate(3) rotateX(25deg);
   -o-transform: scale(0.5) translate(3) rotateX(25deg);
   -ms-transform: scale(0.5) translate(3) rotateX(25deg);
   transform: scale(0.5) translate(3) rotateX(25deg);
}
```

O número mínimo de argumentos é definido pelas variáveis explícitas antes dos três pontos. Cada uma receberá o valor passado na posição correspondente ou pelo seu nome.

5.6.3 A variável @*nome* . . .

`@arguments` sempre contém *todos* os argumentos. Mas é possível ter em um mesmo mixin argumentos "anônimos" e declarados.

Se em vez de `...` como último argumento, o último argumento for uma *variável* seguida de `...` sem espaço (`@args...`, por exemplo), Less permite que ela receba um número ilimitado de argumentos seja atribuída a ela. Assim é possível o acesso à lista contendo apenas os argumentos que *não* foram explicitamente declarados em variáveis. Dentro do mixin, a lista pode ser obtida através de referência à variável declarada sem as reticências (por exemplo `@args`).

O mixin abaixo usa o primeiro argumento para definir a propriedade `border`. Os argumentos restantes (ilimitados) são usados para definir `margin`:

```
.borda-com-margem(@borda; @outros-args...) {
   .border: @borda;
   margin: @outros-args;
}
```

Abaixo ele está sendo chamado com cinco argumentos:

```
.sec7 {
   .borda-com-margem(solid black 1px; 5; 10; 15; 5;);
}
```

E o resultado em CSS será:

```
.sec7 {
   border: solid #000000 1px;
   margin: 5 10 15 5;
}
```

5.7 Sobrecarga de mixins

Mixins podem ter nomes iguais e serem selecionados com base na *quantidade de argumentos* declarados.

```
.mancha(@alpha: 1){            // 0 ou 1 argumento
   background: black;
   color: white;
}

.mancha(@gray; @alpha: 1) {    // 1 ou 2 argumentos
   @yarg: 255 - @gray;
   background: rgb(@gray,@gray,@gray);
   color: rgb(@yarg, @yarg, @yarg);
}

.mancha(@red; @green; @blue; @alpha: 1) {    // 3 ou 4 argumentos
   @der   : 255 - @red;
   @neerg : 255 - @green;
   @eulb  : 255 - @blue;
   background: rgb(@red, @green, @blue);
   color: rgb(@der, @neerg, @eulb);
}
```

Os mixins serão executados de acordo com a correspondência do número de argumentos com os que forem passados na chamada. Se houver ambiguidade, o compilador não irá gerar o CSS e exibirá uma mensagem de erro.

```
.sec3 {
     .mancha;           // chama o primeiro mixin
}
.sec4 {
     .mancha(32;0.5)    // chama o segundo mixin
}
.sec5 {
     .mancha(128;64;32) // chama o terceiro mixin
}
```

Haveria ambiguidade se o mixin fosse chamado com apenas um argumento, pois o compilador não saberia distinguir entre o primeiro ou o segundo, já que ambos são compatíveis com um argumento.

Resultado em CSS:

```
.sec3 {
   background: black;
   color: white;
}
.sec4 {
   background: #202020;
   color: #dfdfdf;
}
.sec5 {
   background: #804020;
   color: #7fbfdf;
}
```

Pode-se também selecionar mixins de mesmo nome com *parâmetros fixos*.

Os mixins `.caixa()` abaixo têm o primeiro parâmetro fixo que é usado para distingui-los. O segundo parâmetro é variável e pode ser omitido. O último mixin `.caixa()` recebe dois parâmetros quaisquer, e o segundo é opcional. Qualquer chamada a mixins `.caixa()` com um ou dois argumentos irá selecioná-los. Se a chamada contiver no primeiro argumento algum dos parâmetros fixos, este também será selecionado e as propriedades serão *misturadas* (sem nenhuma garantia que não haverá repetição).

```
.caixa-base(@espessura: 1px; @cor:black) {
   margin: @espessura;
   padding: @espessura;
   border: solid @cor @espessura;
}

.caixa(grande; @cor:black) {
   .caixa-base(4px; @cor);
}

.caixa(media; @cor:black) {
   .caixa-base(2px; @cor);
}

.caixa(pequena; @cor:black) {
   .caixa-base(1px; @cor);
}

.caixa(zero; @cor:black) {
   .caixa-base(0; @cor);
}

.caixa(@todas; @todas: xyz) {
   position: absolute;
   top: 0; left: 0;
}
```

Pode-se então selecionar o mixin usando o nome do argumento fixo como parâmetro.

```
.sec2 {
   .caixa(pequena;red);
}
```

Além do mixin que contém o primeiro argumento “pequena”, a chamada acima irá executar *também* o mixin

```
.caixa(@todas, @todas: xyz)
```

que combina com *qualquer chamada de um ou dois argumentos*.

O resultado será o CSS abaixo:

```
.sec2 {
   margin: 1px;
   padding: 1px;
   border: solid #ff0000 1px;
   position: absolute;
   top: 0;
   left: 0;
}
```

Se os mixins ficarem em um documento global importado por outros documentos locais, o parâmetro fixo pode ser armazenado em uma variável e ser definido em um único lugar no documento:

```
@import url(mixins-11-matching.less);
@tipo-caixa: grande; // tamanho default para esta folha de estilos

.sec1 {
   .caixa(@tipo-caixa);
}
```

Resultado em CSS:

```
.sec1 {
   margin: 4px;
   padding: 4px;
   border: solid #000000 4px;
   position: absolute;
   top: 0;
   left: 0;
}
```

5.8 Mixins para CSS multi-browser

Mixins são ótimos para encapsular propriedades dependentes de browser. Essas propriedades têm um prefixo proprietário e geralmente recebem os mesmos dados. É possível encapsular em um mixin as propriedades relativas a diversos fabricantes diferentes, e chamar a propriedade pelo nome do mixin.

O mixin abaixo gera a propriedade do CSS3 *transform* com um número mínimo de argumentos para quatro browsers diferentes:

```
.transform(@transform:scale(1); ...) {
   -webkit-transform: @arguments;
   -moz-transform: @arguments;
   -o-transform: @arguments;
   -ms-transform: @arguments;
   transform: @arguments;
}
```

Exemplo de uso:

```
.secao {
   .transform(
     rotate(45deg);
     scale(0.5);
     translate(100px,100px);
     skewY(10deg);
   )
}
```

Resultado da geração do CSS:

```
.secao {
   -webkit-transform: rotate(45deg) scale(0.5) translate(100px, 100px) skewY(10deg);
   -moz-transform: rotate(45deg) scale(0.5) translate(100px, 100px) skewY(10deg);
   -o-transform: rotate(45deg) scale(0.5) translate(100px, 100px) skewY(10deg);
   -ms-transform: rotate(45deg) scale(0.5) translate(100px, 100px) skewY(10deg);
   transform: rotate(45deg) scale(0.5) translate(100px, 100px) skewY(10deg);
}
```

5.9 Mixins que retornam valores

Variáveis definidas no corpo de mixins têm seu escopo propagado no bloco que faz a chamada, ou seja, os valores dessas variáveis podem ser lidas no bloco destino.

Por exemplo, o mixin abaixo define sete variáveis. Quatro como parâmetros e quatro no corpo do mixin.

```
.mancha(@red; @green; @blue; @alpha: 1) { // 3 ou 4 argumentos
   @der   : 255 - @red;
   @neerg : 255 - @green;
   @eulb  : 255 - @blue;
   @opacity: @alpha;
   background: rgb(@red, @green, @blue);
   color: rgb(@der, @neerg, @eulb);
}
```

As quatro variáveis que foram declaradas no corpo são *visíveis* no bloco que chama o mixin. O seletor abaixo utiliza `@opacity` e `@neerg`:

```
.sec8 {
   .mancha(255,128,64);
   opacity: @opacity;
   border-color: @neerg;
}
```

Resultado em CSS:

```
.sec8 {
   background: #ff8040;
   color: #007fbf;
   opacity: 1;
   border-color: 127;
}
```

As variáveis dentro de mixins podem ser usadas para transformar os valores recebidos. Assim o mixin se comporta como uma *função*.

Exemplo: dois mixins para conversão de coordenadas cartesianas e polares:

```
.polar(@x, @y) {
   @r: sqrt(@x*@x + @y*@y);
   @theta: convert(atan(@y / @x), deg);
}

.cartesian(@r, @theta) {
   @x: @r * cos(@theta);
   @y: @r * sin(@theta);
}
```

Usando os mixins para fazer conversões:

```
.sec6:before {
   .polar(3, 4);
   content: 'r: @{r} theta: @{theta}';
}

.sec7:before {
   .cartesian(5, convert(53.13010235deg, rad));
   content: 'x: @{x} y: @{y}';
}
```

Resultado em CSS:

```
.sec6:before {
   content: 'r: 5 theta: 53.13010235415598deg';
}

.sec7:before {
   content: 'x: 3.0000000002901417 y: 3.9999999997823936';
}
```

Estes mixins utilizam funções matemáticas e de conversão que fazem parte do Less e serão abordadas mais adiante.

5.10 Mixins com :extend

O pseudo-elemento `:extend` pode ser usado dentro de mixins, permitindo que o mixin estenda os seletores do bloco onde é declarado com propriedades de outros seletores. Por exemplo:

```
.label:before > input:hover {
   content: 'hovering';
}

.list-style() {
   &:extend(.label:before > input:hover);
}
```

A chamada do mixin estende os seletores do bloco onde é usado com as propriedades dos seletores correspondentes:

```
.menu-item, .nav-item {
   .list-style();
}
```

Resultado em CSS:

```
.label:before > input:hover, .menu-item, .nav-item {
   content: 'hovering';
}
```

5.11 Agregação de valores

Less permite que valores de propriedades definidas em um mixin sejam *agregadas* aos valores existentes de propriedades dos seletores onde o mixin é usado, em uma *lista separada por vírgulas*. Isto é útil para propriedades que aceitam valores separados por vírgulas como `font-family` e outros.

Para usar, é necessário incluir um sinal `+` tanto na propriedade declarada no mixin, quanto na propriedade declarada no seletor afetado.

Por exemplo, o mixin abaixo define uma transição multi-plataforma sobre a propriedade `transform` do CSS3:

```
.transform-transition(@arg) {
    @duration: @arg;

    -webkit-transition-property+: -webkit-transform,
                                  -moz-transform,
                                  transform;
    -webkit-transition-duration+: @duration, @duration, @duration;
}
```

Aqui ele é usado como base para *adicionar* mais um atributo à transição:

```
.secao1 {
   .transform-transition(2s);
   -webkit-transition-property+: opacity;
   -webkit-transition-duration+: @duration;
}
```

O resultado em CSS acrescenta os atributos adicionados no fim da lista separada por vírgulas:

```
.secao1 {
  -webkit-transition-property: -webkit-transform,
                               -moz-transform,
                               transform,
                               opacity;
  -webkit-transition-duration: 2s, 2s, 2s, 2s;
}
```

{6: operações e funções }

Neste capítulo serão explorados recursos do Less que permitem realizar operações matemáticas e de transformação de dados, operações condicionais e loops.

6.1 Operações aritméticas

Less permite realizar operações matemáticas entre valores numéricos e cores, levando em conta as unidades em que são declaradas e fazendo conversão automática quando possível.

As expressões geram CSS *estático*, que difere da função `calc()` do CSS3 (que também é capaz de realizar operações matemáticas), ou das operações em JavaScript que podem operar dinamicamente. Less normalmente não tem acesso à estrutura DOM da página que irá receber o estilo (é possível fazer isto com Less sendo processado no browser). Normalmente Less é apenas um processador que *gera* CSS estático, portanto toda a matemática irá servir para gerar um string estático. Ainda assim é bastante útil para realizar alinhamentos, dimensionamento, manipulação de cores, temporização de animações, distribuição de gradientes, calculo do tamanho de fontes, etc.

As expressões podem ser usadas diretamente na atribuição de valores a propriedades, na definição de variáveis e podem incluir outras variáveis. Para evitar ambigüidades, é recomendável escrever as expressões entre parênteses e de preferência realizá-las em variáveis. Esta recomendação também garantirá o funcionamento em versões futuras de Less.

Os operadores que podem ser usados em expressões matemáticas são os mesmos operadores básicos encontrados nas linguagens de programação:

- `*` multiplicação
- `/` divisão
- `+` soma
- `-` subtração

O exemplo abaixo ilustra o uso desses operadores:

```
@parte1: 10px;
@parte2: 20px;
@borda: 2px;
@margem: ((((@parte1 + @parte2) / (2 * @borda)) - 0.5);

.sec1 {
   margin: @margem @margem;
   color: (#777 + #333);
   width: 100% / 4;
}
```

O resultado do processamento do código acima em CSS é:

```
.sec1 {
   margin: 7px 7px;
   color: #AAAAAA;
   width: 25%;
}
```

As operações também têm a mesma precedência das linguagens de programação: calculados da esquerda para a direita, mas com multiplicação, resto e divisão tendo prioridade sobre adição e subtração. A precedência pode ser alterada com o uso de parênteses. Removendo os parênteses do exemplo acima, o resultado é outro:

```
@margem: (@parte1 + @parte2 / 2 * @borda - 0.5);
```

Resultado:

```
.sec1 {
   margin: 29.5px 29.5px;
}
```

Erros de matemática *não são detectados* pelo pré-processador Less. A divisão por zero, por exemplo, resulta em `Infinity` (um tipo de dados do JavaScript). Outros podem gerar CSS incorreto, como por exemplo tamanhos de fonte negativas ou `NaN` (“Not a Number” – outro tipo de dados do JavaScript). O código Less abaixo:

```
.sec3 {
   @tam: (@parte2 - 2 * @parte1);
   font-size: (12pt - @parte2);
   margin: @tam / @parte2 @parte2 / @tam - @parte1;
}
```

Resulta neste CSS incorreto:

```
.sec3 {
   font-size: -8pt;   margin: 0px Infinitypx;
}
```

6.2 Operações usando unidades

Operações aritméticas podem usar e geralmente utilizam unidades. Se em uma expressão matemática nenhum dos números tiver uma unidade, o resultado final será mantido sem unidade:

```
.sec4 {
   font-size: 4 + 2 + 16;
}
```

CSS:

```
.sec4 {
   font-size: 22;
}
```

Se um dos termos possuir uma unidade, essa unidade será usada no resultado final:

```
.sec5 {
   font-size: 4 + 2px + 16;
}
.sec5 {
   font-size: 22px;
}
```

Se os números de uma expressão tiverem mais de uma unidade, e as unidades forem diferentes, elas poderão ser convertidas automaticamente *se forem compatíveis* e *se as operações forem de soma* (+) *e subtração* (-). A *primeira* unidade usada na expressão será a unidade do número resultante. A tabela abaixo lista os grupos compatíveis:

Unidades	Tipo
`px, m, cm, mm, in, pt, px`	Comprimentos
`s, ms`	Tempo
`deg, rad, grad, turn`	Ângulos

Unidades do CSS não listadas na tabela acima (`px`, `em`, `rem`, etc.) *não podem ser misturadas*. Podem ser usadas em expressões contendo números da mesma unidade ou com números sem unidade.

Observe o efeito da conversão automática de unidades nas expressões abaixo.

```
.sec6 {
   padding: (1cm + 1mm) (1mm + 1cm) (1pt + 1pc) (1m + 1in);
   duration: (2s + 2ms);
   margin: (1px + 1rem); // incompatíveis
   transform: rotate(1deg + 1rad) rotate(1rad + 1deg)
              rotate(1deg + 1grad) rotate(1deg + 1turn);
}
```

E o CSS resultante:

```
.sec6 {
   padding: 1.1cm 11mm 13pt 1.0254m;
   duration: 2.002s;
   margin: 2px;
   transform: rotate(58.29577951deg) rotate(1.01745329rad)
              rotate(1.9deg) rotate(361deg); }
```

Outras combinações produzirão resultados incorretos e devem ser evitadas. Se forem realizadas operações entre grupos não compatíveis, ou operações de multiplicação ou divisão envolvendo números com unidades diferentes, os valores serão multiplicados *sem considerar as unidades*, e a unidade gerada no final pode ser tanto a primeira (em divisão) como a última (em multiplicação).

```
.calculos-incorretos {
   duration: (2s * 2ms);
   margin: (1px + 1em) (1em + 1rem) (1mm * 1cm) (1m / 100cm);
   transform: rotate(3turn / 1080deg) rotate(360deg * 3turn);
}
```

CSS contendo resultados incorretos:

```
.calculos-incorretos {
   duration: 4ms;
   margin: 2px 2em 1cm 0.01m;
   transform: rotate(0.00277778turn) rotate(1080deg);
}
```

6.3 Arredondamento e percentagem

A tabela abaixo relaciona as funções `ceil()` (teto), `floor()` (piso) e `round()` (arredonda) que eliminam a parte decimal de um número de ponto-flutuante segundo diferentes critérios, além de `percentage()` que converte o valor em percentagem.

Função	Recebe	Retorna
`ceil(n)`	número	Próximo inteiro. `ceil(5.1)` gera 6
`floor(n)`	número	Inteiro anterior. `floor(5.9)` gera 5
`round(n)`	número	Inteiro arredondado. `round(5.4)` gera 5, `round(5.5)` gera 6.
`percentage(n)`	número	Número multiplicado por 100 + unidade %

Exemplo:

```
.secao {
   margin: ceil(5.2px) floor(5.7px) round(5.5px) round(5.4px);
   width: percentage(0.5);
}
```

CSS:

```
.secao {
   margin: 6px 5px 6px 5px;
   width: 50%;
}
```

6.4 Conversão de unidades

Nem sempre é possível ou recomendado usar transformação automática de unidades. Às vezes é desejável acrescentar uma unidade em um número após a realização de cálculos, ou ainda trocar a unidade sem fazer conversão. Duas funções servem a esse propósito:

Função	Recebe	Retorna
`convert(n, u)`	`n` = número (com ou sem unidade), `u` = unidade compatível	Número convertido na nova unidade: `convert(1in, cm)` gera `2.54cm`
`unit(n, u)`	`n` = número (com ou sem unidade), `u` = unidade	Número recebido qualificado com a unidade passada como parâmetro. `unit(1in, cm)` gera `1cm`

Se as unidades passadas para `convert()` não forem compatíveis ele funcionará igual a `unit()` simplesmente copiando o mesmo valor e trocando a unidade. A função `convert()` *não suporta* pixels (`px`), `em` ou `rem`.

Os exemplos abaixo ilustram o uso das funções `convert()` e `unit()` em várias situações e comparadas à conversões automáticas incorretas em expressões:

```
@height: 2cm;
@width: 10px;
@length: 2in;

.secao {
   margin: (@width / @height) unit(@width / @height, cm);
   padding: (@height * @length) (@height * convert(@length, cm));
   font-size: convert(1in, cm);
   duration: unit(floor(100 * 12 / 142), s);
}
```

Resultado em CSS:

```
.secao {
   margin: 5px 5cm;
   padding: 4cm 10.16cm;
   font-size: 2.54cm;
   duration: 8s;
}
```

6.5 Funções matemáticas e trigonométricas

Less possui várias funções matemáticas e trigonométricas que são úteis para calcular valores usados em alinhamentos, parâmetros para animações 2D e 3D, transições, conversão de coordenadas, métricas de fontes, filtragem e composição de cores, etc. evitando a necessidade de usar JavaScript quando precisa-se apenas de valores estáticos.

As funções estão relacionadas na tabela abaixo.

Os valores passados podem ser números, variáveis que contenham números ou outras funções que retornem números.

A unidade *default* para ângulos é radianos (`rad`) – é a unidade dos valores retornados pelas funções trigonométricas. Mas na chamada de funções, os parâmetros podem ser qualificados com uma unidade. Se os valores retornados forem comprimentos eles serão números (sem unidades).

Função	Recebe	Retorna
`sin(a)`	ângulo	seno do ângulo
`asin(c)`	comprimento	ângulo (inverso do seno)

Função	Recebe	Retorna
`cos(a)`	ângulo	cosseno do ângulo
`acos(c)`	comprimento	ângulo (inverso do cosseno)
`tan(a)`	ângulo	tangente do ângulo
`atan(c)`	comprimento	ângulo (inverso da tangente)
`pi()`		3.14159265
`pow(n,p)`	`n` = número, `p` = potência	número elevado à potência
`sqrt(n)`	número	raiz quadrada
`mod(n,d)`	`n` = nominador, `d` = denominador	módulo (resto)
`abs(n)`	número	valor absoluto

Veja alguns exemplos.

Ângulos e arcos:

```
#senos {
   transform: translateY(sin(45deg)) rotate(convert(asin(0.5px), deg));
}

#co-senos {
   transform: translateX(cos(45deg)) rotate(convert(acos(0.5px), deg));
}

#tangentes {
   transform: skewX(tan(45deg)) rotate(convert(atan(1px), deg));
}
```

Resultado em CSS:

```
#senos {
   transform: translateY(0.70710678) rotate(30deg);
}

#co-senos {
   transform: translateX(0.70710678) rotate(60deg);
}

#tangentes {
   transform: skewX(1) rotate(45deg);
}
```

Um mixin que calcula áreas e circunferências:

```
.circle-mixin(@radius, @x, @y) {
   @area: pi() * pow(@radius, 2);
   @circunference: 2 * pi() * @radius;
}
```

```
#trigonometria {
   .circle-mixin(5, 0, 0);
   transform: rotate(acos(0.5px)*180deg/pi());
   content:  'area: @{area} circunference: @{circunference}';
}
```

CSS:

```
#trigonometria {
   transform: rotate(60deg);
   content: 'area: 78.53981633974483 circunference: 31.41592653589793';
}
```

Cálculo de potência, raiz quadrada, restos, números absolutos:

```
#quadrados {
   margin: pow(5px,2);
   padding:sqrt(25cm);
}
#modulos {
   length: mod(11px,3);
   width: abs(-10cm);
   @component: pow(2,7); // 50%
   color: rgb(@component,@component,@component);
}
```

CSS:

```
#quadrados {
   margin: 25px;
   padding: 5cm;
}
#modulos {
   length: 2px;
   width: 10cm;
   color: #808080;
}
```

6.6 Funções que operam sobre coleções

Less possui algumas funções que operam sobre *listas*. Uma lista é um conjunto de valores separada por vírgulas ou espaços. Muitas propriedades do CSS e seletores podem ser tratados como listas, e elas podem ser usadas em loops.

Função	Recebe	Retorna
`length(a)`	`a` = lista de valores	o número de elementos da lista
`extract(a, p)`	`a` = lista de valores, `p` = posição	o valor da lista contido na posição informada

Função	Recebe	Retorna
`min(c)`	`c` = lista de números separada por vírgulas	o menor número da lista
`max(c)`	`c` = lista de números separada por vírgulas	o maior número da lista

O exemplo abaixo ilustra o uso de várias funções listadas acima:

```
@comp: 255 127 64;
@tamanhos: 36pt 24pt 18pt 16pt 12pt 10pt;
@fontes: 'Garamond', 'Times New Roman', Times, serif;
@steps: 0% 20% 40% 60% 80% 100%;

.paragrafo {
   font-size: min(@tamanhos);
   duration: unit(length(@steps), s);
   font-family: @fontes;
}
.titulo {
   font-size: max(@tamanhos);
   color: rgb(extract(@comp,1), extract(@comp, 2), extract(@comp, 3));
   font-family: extract(@fontes, 1), extract(@fontes, length(@fontes));
}
```

Resultado em CSS:

```
.paragrafo {
   font-size: 10pt;
   duration: 6s;
   font-family: 'Garamond', 'Times New Roman', Times, serif;
}
.titulo {
   font-size: 36pt;
   color: #ff7f40;
   font-family: 'Garamond', serif;
}
```

6.7 Transformação e formatação de texto

Less oferece algumas funções que auxiliam na construção de strings de texto formatado e conversão de texto em números, cores, urls, etc. Através deles é possível suprimir várias limitações do Less e gerar CSS literalmente.

Suponha que temos as seguintes variáveis:

```
@image-name: 'pattern 5.png';
@docroot: 'http://www.test.com/app';
```

E queremos usá-las para construir o valor de uma propriedade `background` da forma:

```
background: url(http://www.test.com/app/images/pattern%205.png);
```

Poderíamos tentar concatenar as variáveis assim:

```
.sec8 {
   background: url(@docroot '/images/' @image-name);
}
```

Mas o processador Less reclama e não permite. Precisamos usar interpolação de variáveis e colocá-las dentro do string:

```
.sec8 {
   background: url('@{docroot}/images/@{image-name}');
}
```

Com isto obtemos o resultado:

```
.sec8 {
   background: url('http://www.test.com/app/images/pattern 5.png');
}
```

que não é exatamente o que queremos. Precisamos converter o espaço no nome do arquivo em `%20` e remover as aspas.

A remoção das aspas pode ser feita com a função `e()` ou pelo operador `~`. São sinônimos. Tanto faz usar um como outro. Esta função converte o string recebido em uma representação nativa, que na prática significa remover as aspas do resultado final gerado em CSS: A conversão pode ser feita da forma abaixo:

```
.sec8 {
   background: e('url(@{docroot}/images/@{image-name})');
}
```

ou

```
.sec8 {
   background: url(~'@{docroot}/images/@{image-name}');
}
```

O operador `~` é mais versátil já que a função `e()` não é aceita em qualquer lugar. O resultado para qualquer uma das alternativas acima é a mesma:

```
.sec8 {
   background: url(http://www.test.com/app/images/pattern 5.png);
}
```

Ainda falta converter o espaço do nome da imagem no formato *url-encoded*: `%20`. Isto pode ser feito com a função `escape`, que requer um string:

```
.sec8 {
   @image-escaped: escape(@image-name);
   background: url(~'@{docroot}/images/@{image-escaped}');
}
```

O resultado final é:

```
.sec8 {
   background: url(http://www.test.com/app/images/pattern%205.png);
}
```

Há outras formas de obter o mesmo resultado.

Na tabela abaixo estão listadas as funções e operadores de manipulação de string disponíveis em Less:

Função ou operador	O que recebe	O que faz
`e('s')` ou `~'s'`		

	`'s'` = string	Converte string em representação nativa do CSS (remove as aspas do CSS gerado). Quaisquer variáveis interpoladas dentro do string são processadas.
		Sem o uso deste operador, o resultado em CSS mantém as aspas.
		Exemplo:
		`~"12px"` ou `~'12px'` ou `e("12px")`
		resulta em CSS contendo
		`12px`

`escape('s')`		

	`'s'` = string	Codifica string em formato *url-encoded*, que atribui códigos unicode no formato `%hh` para caracteres que são reservados em URLs, onde `h` é um dígito hexadecimal entre `0` e `f`.
		URL-encoding *não* substitui:
		`,`, `/`,`?`, `@`, `&`, `+`, `'`, `~`, `!`, `$`.
		Exemplo:
		`encode('São Paulo')`
		gera
		`S%C3%A3o%20Paulo`

Função ou operador	O que recebe	O que faz
`%('s', args...)`		

	`'s'` = string com parâmetros de substituição,
	`args...` = valores de substituição
	`%a`,`%A`, `%s`, `%S`,`%d`, `%D`
		Permite a criação de strings formatados com parâmetros de substituição (uma versão ultra-simplificada do `printf` do C).
		Os parâmetros de substituição maiúsculos fazem `escape()` automático.
		`%a`/`%A` e`%d`/`%D` são equivalentes (Less 1.7) mas `%s`/`%S` retornam `undefined` se o valor for uma cor.
		Exemplo:
		`%(~'url(%a) url(%A)', a b, a b)`
		gera
		`url(a b) url(a%20b)`

`replace(s1, p, s2, f)`		

	`s1` = string,
	`p` = regexp,
	`s2` = substituição,
	`f` = flags
		Substitui um padrão de expressão regular (`p`) encontrado em um string (`s1`) por outro (`s2`) com flags opcionais (`g` = global, `i` = ignore case).
		Exemplo:
		`replace(~"w.a.b", "a\.b", "x.k")`
		gera
		`w.x.k`

`color('s')`		

	`s` = string contendo a representação de uma cor	Retorna uma cor (sem as aspas). *Exemplos*:
		`color('#ff0000')` gera `#ff0000`
		`color('red')` gera `#ff0000`.
		É possível obter o mesmo resultado usando a função `e()` ou o operador `~`:
		`~'#ff0000'` gera `#ff0000`

Mais exemplos usando `%()`:

```
@estilo: italic;
@peso: 900;
@mintam: 10;
@maxtam: 36;
@fam: Arial, Helvetica, sans-serif;
@image-name: 'patt 5.png';
@docroot: 'http://www.test.com/app';

.sec7 {
    font: %(~"%d %d %dpt/%dpt %d",@estilo,@peso,@mintam,@maxtam,@fam);
    background: %(~"url(%d/images?name=%D)", @docroot, @image-name);
}
```

CSS:

```
.sec7 {
  font: italic 900 10pt/36pt Arial, Helvetica, sans-serif;
  background:url('http://www.test.com/app'/images?name='patt%205.png');
}
```

Mais exemplos usando `replace()`:

```
@old-url: ~"url(http://abc.com/logo-ABC.COM.jpg)";
.sec9 {
   background: replace(@old-url, "abc\.com", "xyz.com.br", "gi");
}
```

CSS:

```
.sec9 {
   background: url(http://xyz.com.br/logo-xyz.com.br.jpg);
}
```

6.8 Sintaxes problemáticas

Vimos que o uso de expressões aritméticas usando unidades incompatíveis (ou expressões de divisão e multiplicação com unidades diferentes) geram resultados incorretos e devem, portanto, ser evitados. Outros problemas surgem na ambigüidade devido ao significado duplo da *barra* `/` e do *espaço*. Alguns desses problemas podem ser resolvidos com o uso de parênteses.

O trecho de código abaixo inclui várias propriedades que necessitam preservar a *barra* no CSS. O Less identifica essas propriedades e não realiza a operação, copiando os valores intactos para o CSS:

```
@media screen and (aspect-ratio: 16/9) and (min-height: 960px/2 ) {
   .section {
     margin-top: 16/10;
     font: 12pt/36pt;
     font-size: 12pt/36pt;
   }
}
```

O compilador Less gera o CSS abaixo para o código acima:

```
@media screen and (aspect-ratio: 16/9) and (min-height: 960px/2) {
   .section {
      margin-top: 1.6;
      font: 12pt/36pt;
      font-size: 0.33333333pt;
   }
}
```

Observe que o `aspect-ratio` não foi tratado como uma divisão, corretamente, mas o `min-height` também não, embora o valor aceito por ele seja número.

Dentro do bloco `.section` a propriedade `margin-top` teve o valor 16/10 transformado em 1.6, `font-size` dividiu `12pt` por `36pt` mas `font`, que aceita no CSS o valor `12pt/36pt` não realizou a divisão.

E se você realmente quisesse realizar a operação de divisão em `aspect-ratio` e `font`? E também a subtração de `min-height`? Neste caso, basta envolver a expressão entre parênteses que o Less irá entender que ela deve ser tratada como expressão matemática:

```
@media screen and (aspect-ratio: (16/9)) and (min-height: (960px/2) ) {
   .section {
      margin-top: 16/10;
      font: (12pt/36pt);
      font-size: 12pt/36pt;
   }
}
```

E agora o resultado do CSS é:

```
@media screen and (aspect-ratio: 1.77777778) and (min-height: 480px) {
   .section {
       margin-top: 1.6;
       font: 0.33333333pt;
       font-size: 0.33333333pt;
   }
}
```

Quanto ao significado do espaço ele também pode ser resolvido mas não com parênteses. É preciso usar uma notação que não seja ambígua. Se um valor de propriedade aceita quatro números separados por espaços, o Less só irá tratar os valores como expressão matemática *se não houver outra forma de interpretar os dados*. Veja o resultado dos valores de `margin` e `padding` abaixo:

```
header, footer {
   padding: -5 -10 +5 -20;
   margin: -5 - 10 +5- 20;
}
```

Os valores de `padding` podem ser interpretados como quatro valores, então Less irá tratá-los assim. Já os valores de `margin` não podem ser tratados como valores,

pois um `-` não vale como valor, e `+5-` é ilegal, mas ainda é possível encontrar *dois* valores, então o resultado é:

```
header, footer {
   padding: -5 -10 5 -20;
   margin: -15 -15;
}
```

Se a intenção for a realização da expressão aritmética, o sinal negativo não pode estar junto do número se houver um espaço antes. Tem que ficar claro que é uma expressão:

```
header, footer {
   padding: -5 - 10 + 5 - 20;
   margin: -5-10+5-20;
}
```

Resultado em CSS:

```
header, footer {
   padding: -30;
   margin: -30;
}
```

{7: loops.guards }

Neste capítulo serão explorados recursos do Less que permitem realizar operações condicionais e loops.

7.1 Mixins condicionais (mixin guards)

Pode-se criar mixins parametrizados que definem regras de estilo de acordo com operações condicionais realizadas sobre os valores recebidos. São chamados de *mixin guards*. Através deles é possível tomar decisões de forma similar a expressões *if* e *if/else* existentes em linguagens de programação.

A expressão condicional é representada por um *teste* seguindo a palavra-chave `when` ('quando'). Assim pode-se representar expressões do tipo "defina o mixin x *quando* a expressão y for verdadeira" usando a seguinte sintaxe:

```
.mixin(@parametros) when (condição sobre @parametros) {
     regras;
}
```

Veja um exemplo:

```
.page (@fundo) when (@fundo = black) {
    background: @fundo;
    color: white;
}
.page (@fundo) when (@fundo = white) {
   background: @fundo;
   color: black;
}
```

Uso do mixin condicional:

```
.noite {
    .page(rgb(0,0,0));
}
.dia {
   .page(rgb(255,255,255));
}
```

Resultado em CSS:

```
.noite {
   background: #000000;
   color: white;
}
.dia {
   background: #ffffff;
   color: black;
}
```

Vale lembrar que os mixins condicionais geram *CSS estático*. Em Web sites responsivos muitas vezes o que se deseja é um comportamento dinâmico. Usando *media queries* é possível controlar o efeito do CSS em diferentes dispositivos, resoluções e dimensões de tela. Portanto, um mixin condicional como:

```
.skin(@width) when (@width >= 500px) {
   column-count:2;
}
.skin(@width) when (@width < 500px) {
   column-count:1;
}
```

que *parece* ser variável, é na verdade aplicado com um valor estático *antes* da compilação do Less:

```
.sec {
    .skin(600px);
}
```

e gera o CSS abaixo, *invariável*.

```
.sec {
   column-count: 2;
}
```

Para um Web site responsivo, o ideal é definir o número de colunas de acordo com a largura de tela do dispositivo em tempo real, que é possível através de media queries do CSS:

```
@media screen and (min-width: 500px) {
   .sec  {
     column-count:2;
   }
}

@media screen and (max-width: 500px) {
   .sec {
     column-count:1;
   }
}
```

7.1.1 Operadores condicionais

Para testar condições pode-se se usar os seguintes operadores: `>`, `>=`, `<`, `<=`, `=`.

```
@max-duration: 10s;
@max-repeats: 2;
.anim (@duration) when (@duration >= @max-duration) {
   animation-duration: @duration;
   animation-iteration-count:1;
}
.anim (@duration) when (@duration < @max-duration) {
   animation-duration: @duration;
   animation-iteration-count:@max-repeats;
}
.symbol {
   .anim(15s);
}
```

Resultado em CSS:

```
.symbol {
   animation-duration: 15s;
   animation-iteration-count: 1;
}
```

7.1.2 Inversão de uma condição

Para *inverter* o resultado de um teste, pode-se usar **`when not`**, que inverte o resultado da expressão condicional.

No exemplo abaixo, o mixin testa se a variável recebida é ou não um número e chama o mixin adjust que define uma propriedade se o valor retornado *não* for falso (as palavras `true` e `false` são retornadas por funções como `isnumber` mas elas não têm um significado especial em Less):

```
@weight: bold; .adjust(@test) when not (false) {
    font-weight:@weight;
}
.section {
   font-size: 12pt;
   .adjust(isnumber(@weight));
}
```

Em `.section` o mixin é chamado com o resultado do teste de `isnumber`, que retorna `false` já que `bold` não é número. O resultado em CSS é:

```
.section {
   font-size: 12pt;
   font-weight: bold;
}
```

7.1.3 Detecção de tipos e unidades

Além de `isnumber` existem quatro outras funções usadas para testar o tipo de um valor:

Função	Recebe	Retorna
`iscolor(s)`	string	`true` se valor for uma cor, ou seja, se o string contiver `#rrggbb`, `#rrggbbaa`, `rgb(%,%,%)`, `rgba(r,g,b,a)`, *nome-de-cor*, etc. `false` caso contrário.
`iskeyword(s)`	string	Quaisquer nomes, sejam de seletores, variáveis, comandos do less, retornam `true`. Retorna `false` se valor for string, cor, número ou url; `true` caso contrário.
`isnumber(s)`	string	`true` se valor for um número: 12, 12.3, 12%, 12cm, 12/36. `false` se não for.
`isstring(s)`	string	`true` se o valor estiver entre aspas ou apóstrofes. `false` se não for string.
`isurl(s)`	string	`true` se valor for uma url. `false` se não for.

Alguns exemplos usando as funções acima:

```
.istest {
   content: '1. string'  isstring('12345');
   content: '2. string'  isstring(12345);
   content: '3. keyword' iskeyword(flowerblue);
   content: '4. keyword' iskeyword(cornflowerblue);
   content: '5. number'  isnumber(12.77);
   content: '6. number'  isnumber(~'123');
   content: '7. color'   iscolor(cornflowerblue);
   content: '8. color'   iscolor(~'rgba(100%,100%,100%,1)');
   content: '9. url'     isurl(url(image.jpg));
   content: '10.url'     isurl(~'/image/jpg');
}
```

CSS resultante:

```
.istest {
   content: '1. string'  true;
   content: '2. string'  false;
   content: '3. keyword' true;
   content: '4. keyword' false;
   content: '5. number'  true;
   content: '6. number'  false;
   content: '7. color'   true;
   content: '8. color'   false;
   content: '9. url'     true;
   content: '10.url'     false;
}
```

Para *números* existem ainda quatro funções que retornam `true` ou `false` de acordo com as unidades que os qualificam:

Função	Recebe	Retorna
`ispixel(n)`	número	`true` se unidade é `px`; `false` caso contrário.
`isem(n)`	número	`true` se unidade é `em`; `false` caso contrário.
`ispercentage(n)`	número	`true` se unidade é `%`; `false` caso contrário.
`isunit(n,u)`	n = número, u = unidade	`true` se o número está na unidade especificada; `false` caso contrário.

Exemplos:

```
@pixvalue: 1px * 30 / 2;
@cmvalue: 10cm + 10pt + .01m;
.istest2 {
   content: '1. pixel'  ispixel(@pixvalue);
   content: '2. pixel'  ispixel(convert(@pixvalue, 'pt'));
   content: '3. em' isem(12em + 12pt);
   content: '4. em' isem(12pt + 12em);
   content: '5. percentage'  ispercentage(100%);
   content: '6. percentage'  ispercentage(0.5);
   content: '7. unit'  isunit(@cmvalue, 'cm');
   content: '8. unit'  isunit(10ms, 's');
}
```

CSS:

```
.istest2 {
   content: '1. pixel' true;
   content: '2. pixel' false;
   content: '3. em' true;
   content: '4. em' false;
   content: '5. percentage' true;
   content: '6. percentage' false;
   content: '7. unit' true;
   content: '8. unit' false;
}
```

7.1.4 Combinação de expressões condicionais

Várias expressões condicionais podem ser combinadas com a vírgula e têm efeito somatório *OR* (*'ou' lógico*), ou seja, se qualquer uma das expressões resultar em verdadeira, a expressão inteira é aceita.

```
skin(@cor, @tambase) when (lightness(@cor) > 50%), (@tambase > 20) {
   font-size: convert(@tambase, "pt");
   color: darken(@cor, 80%);
   background: @cor;
}
```

```
.sec1 {
    font-family: serif;
   .skin(#d0d0d0, 2); // true OU false = true
}
.sec2 {
   font-family: serif;
   .skin(#101010, 21); //  false OU true = true
}
.sec3 {
   font-family: serif;
   .skin(#101010, 19);  // false OU false = false
}
```

Resultado em CSS:

```
.sec1 {
  font-family: serif;
  font-size: 2;
  color: #040404;
  background: #d0d0d0;
}
.sec2 {
  font-family: serif;
  font-size: 21;
  color: #000000;
  background: #101010;
}
.sec3 {
  font-family: serif;
}
```

O efeito multiplicador *AND* (*'e' lógico*) na combinação de expressões é obtido com a palavra-chave `and`:

```
.mix(@um, @dois) when (iscolor(@um)) and (ispixel(@dois)) {
   margin: @dois;
   color: @um;
}

.sec4 {
   font-family: monospace;
   .mix(red, 2px); // true E true = true
}
.sec5 {
   font-family: monospace;
   .mix(red, 2pt); // true E false = false
}
```

Resultado em CSS:

```
.sec4 {
   font-family: monospace;
   margin: 2px;
   color: #ff0000;
}
.sec5 {
   font-family: monospace;
}
```

A função `default()` é usada como cláusula *else* de um mixin guard. Representa uma condição *default* que será executada *se nenhum outro* mixin condicional de mesmo nome/parâmetros resultar em `true`.

```
.mix(@um, @dois) when (iscolor(@um)) and (ispixel(@dois)) {
   margin: @dois;
   color: @um;
}
.mix(@um, @dois) when(default()) { // caso nenhum mixin selecionado
   margin: 0;
   color: black;
}
```

Aplicando agora o mixin `.mix` nos seletores abaixo:

```
.sec4 {
   font-family: monospace;
   .mix(red, 2px); // true E true = true
}
.sec5 {
   font-family: monospace;
   .mix(red, 2pt); // true E false = false
}
```

Obtém-se o CSS:

```
.sec4 {
   font-family: monospace;
   margin: 2px;
   color: #ff0000;
}
.sec5 {
   font-family: monospace;
   margin: 0;
   color: black;
}
```

7.2 Seletores condicionais

Expressões condicionais não são exclusividade dos mixins. Elas podem ser aplicadas diretamente nos seletores CSS.

```
@debug: true;

div when (@debug) {
    border: solid red 1px;
}
```

Quando `@debug` for true bordas vermelhas serão colocadas em volta de cada `div`:

```
div {
   border: solid red 1px;
}
```

Esse tipo de expressão só pode ser aplicada em um único seletor de cada vez. Para aplicar a vários seletores, deve-se usar o `&`:

```
& when (@debug) {
   section, footer, aside, div {
     border: solid red 1px;
   }
   p:before {
      content:'[P]';
   }
   p:after {
      content:'[/P]';
   }
}
```

Resultado em CSS:

```
section, footer, aside, div {
   border: solid red 1px;
}
p:before {
   content: '[P]';
}
p:after {
   content: '[/P]';
}
```

7.3 Mixins recursivos (loops)

Loops em LESS são chamadas *recursivas* de mixins, ou seja, o mixin chama ele mesmo.

Para que um loop não seja infinito, é necessário que a cada repetição uma condição seja testada e que o valor testado *mude* de forma que em algum momento a condição seja falsa, e o loop tenha fim.

O loop abaixo repete um número de vezes definida quando o mixin for chamado. A cada repetição, o contador é comparado com zero, e se for maior que zero, seu valor é diminuído de um e o corpo do mixin é executado. Quando o valor for zero, o mixin termina.

```
.repetir(@vezes) when (@vezes > 0) {
   .repetir(@vezes - 1);
   margin: (@vezes);
}
```

No bloco abaixo o mixin é chamado para executar quatro vezes:

```
.sec2 {
   .repetir(4);
}
```

E o resultado do CSS é:

```
.sec2 {
   margin: 1;
   margin: 2;
   margin: 3;
   margin: 4;
}
```

Loops também podem ser usados para gerar blocos inteiros de seletores. O mixin abaixo permite gerar até 5 seções, cada uma com uma cor de fundo mais escura.

```
.criar-secoes(@vezes) when (@vezes > 0) and (@vezes <= 5) {
   .secao-@{vezes} {
      color: (darken(#ff0000, 10% * @vezes));
   }
   .criar-secoes( (@vezes - 1) );
}

.criar-secoes(5);
```

Ele é chamado logo em seguida para repetir 5 vezes e gera o seguinte CSS:

```
.secao-5 {
   color: #000000;
}
.secao-4 {
   color: #330000;
}
.secao-3 {
   color: #660000;
}
.secao-2 {
   color: #990000;
}
.secao-1 {
   color: #cc0000;
}
```

Este exemplo ilustra o uso de loops para gerar uma seqüência de keyframes para animação:

```
@positions: 0px 50px 100px 50px 0px;

.make-keyframe(@step, @count) when (@count > 0) {
   .make-keyframe(@step, (@count - 1) );

   @pos: @step * (@count - 1);

   @{pos} {
      top: extract(@positions, @count);
   }
}
```

Uso do mixin:

```
@keyframes anim {
   .make-keyframe(25%, length(@positions));
}
```

Resultado em CSS:

```
@keyframes anim {
   0% {
     top: 0px;
   }
   25% {
     top: 50px;
   }
   50% {
     top: 100px;
   }
   75% {
     top: 50px;
   }
   100% {
     top: 0px;
   }
}
```

{8: cores }

Less possui várias funções para geração, combinação, filtragem e mesclagem de cores. A maioria gera como resultado uma representação de cor em CSS no formato hexadecimal `#rrggbb`.

8.1 Definição de cores

Várias das funções de definição de cores já existem em CSS (ex: `rgb(r,g,b)`), mas algumas só existem desde versões mais recentes e não têm suporte em browsers antigos (`hsl`, `hsla`, etc.). O Less converte essas funções em formatos hexadecimais `#rrggbb` ou na função `rgba(r,g,b,a)` do CSS, quando a cor tiver um componente alfa.

Por exemplo, o código em Less abaixo:

```
@componente: 128;
.secao2 {
   color: rgb(@componente,255,255);
}
```

irá gerar o seguinte CSS:

```
.secao2 {
   color: #80ffff;
}
```

Se você quiser *manter* a função `rgb` no CSS (em vez de converter para `#hhhhhh`), é necessário incluir o valor *entre aspas* e depois removê-las do CSS gerado através do operador `~`:

```
@componente: 128;
.secao1 {
   color: ~'rgb(@{componente},255,255)';
}
```

que irá preservar o string no CSS da forma como ele foi declarado:

```
.secao1 {
   color: rgb(128,255,255);
}
```

8.1.1 Réplicas de funções do CSS

As funções abaixo são réplicas das funções suportadas por CSS1 e 2 (`rgb`, `rgba`) e CSS3 (`hsl`, `hsla`). Todas geram código em formato `#hhhhhh` ou `rgba(r,g,b,a)`.

As funções `rgb` e `rgba` geram uma cor a partir de componentes de luz *vermelha*, *verde* e *azul* (números de 0 a 255, ou 0 a 100%), mais um componente *alfa* (transparência) entre 0 e 1 (0 a 100%).

RGB é um espaço de cores dependente de hardware. HSL é mais intuitivo: escolhe-se a *matiz* (tonalidade) da cor, depois a *saturação* e intensidade ou brilho da *luz*. Uma cor é representada em HSL por

- **hue**: um ângulo que representa a matiz (0 ou 360deg = vermelho);
- **saturation**: um número entre 0 e 1 (0 - 100%) que representa a saturação da cor;
- **lightness**: um número entre 0 e 1 (0 - 100%) que representa a quantidade de luz.

Uma cor 100% saturada como `#ff0000` é representada em HSL como H=0, S=1, L=0.5 (L = 0 é nenhuma luz, preto, L = 1 é branco). A matiz é um círculo contínuo que começa e termina em vermelho, assim pode-se ultrapassar o valor máximo:

- 0deg, 360deg, 720deg, ... = vermelho
- 120deg, 480deg, ... = verde
- 240deg, 600deg, ... = azul

Função	Recebe	Retorna
`rgb(r,g,b)`	números (0-255) ou percentagem	Cor em hexadecimal. *Exemplos*: `rgb(255,0,0)` gera`#ff0000`, `rgb(100%,0,0)` gera `#ff0000`
`rgba(r,g,b,a)`	`r,g,b` = números (0-255) ou %, `a` = número entre 0 e 1 ou percentagem	A mesma função em CSS. *Exemplos*: `rgba(255,255,255,0.5)` gera `rgba(255,255,255,0.5)`
`hsl(h,s,l)`	`h` = ângulo, `s,l` = números entre 0 e 1 ou %	Cor em hexadecimal. *Exemplo*: `hsl(360,1,0.5)` gera `#ff0000`

Função	Recebe	Retorna
`hsla(h,s,l,a)`	`h` = ângulo, `s,l,a` = números entre 0 e 1 ou percentagem	Função rgba do CSS correspondente. *Exemplos*: `hsla(0,1,0.5,0.5)` gera `rgba(255, 0, 0, 0.5)`

8.1.2 Funções HSV

O espaço de cores HSV tem como principal diferença em relação ao HSL o componente de intensidade de luz, que é chamado de *Value* (ou Brilho/Brightness) e é definido como o valor do *maior* componente da cor, em contraste com o HSL que define lightness como um *valor médio*. Portanto, L=0.5 em HSL equivale a V=1 em HSV.

Estas funções não existem no CSS, portanto são convertidas em `#hhhhhh` ou `rgba` pelo processador Less.

Função	Recebe	Retorna
`hsv(h,s,v)`	`h` = ângulo, `s,v` = números entre 0 e 1 ou percentagem	cor em hexadecimal. ex: `hsv(360deg,1,1)` gera `#ff0000`
`hsva(h,s,v,a)`	`h` = ângulo, `s,v,a` = números entre 0 e 1 ou %	função rgba do CSS correspondente: `hsva(0,1,1,0.5)` gera `rgba(255, 0, 0, 0.5)`

Exemplos:

```
@color1: rgb(100%,50%,25%);
@color2: rgba(128,255,64,0.5);

@color3: hsl(120deg,50%,25%);
@color4: hsla(240deg,100%,75%,75%);

@color5: hsv(120deg,50%,50%);
@color6: hsva(240deg,50%,50%,50%);

.colors {
   border-top-color: @color1;
   border-right-color: @color2;
   border-bottom-color: @color3;
   border-left-color: @color4;
   background-color: @color5;
   color: @color6;
}
```

Resultado em CSS:

```
.colors {
   border-top-color: #ff8040;
   border-right-color: rgba(128, 255, 64, 0.5);
   border-bottom-color: #206020;
   border-left-color: rgba(128, 128, 255, 0.75);
   background-color: #408040;
   color: rgba(64, 64, 128, 0.5);
}
```

8.2 Extração de componentes de cores

Estas funções recebem cores e extraem seus componentes individuais nos espaços RGB, RGBA, HSL, HSLA, HSV e HSVA. Os componentes são retornados como números decimais.

8.2.1 Componentes RGBA

Função	Recebe	Retorna
`red(c)`	cor	valor do componente vermelho (`r`): `red(#ff8040)` gera 255 (hex ff)
`green(c)`	cor	valor do componente verde (`g`): `red(#ff8040)` gera 128 (hex 80)
`blue(c)`	cor	valor do componente azul (`b`): `red(#ff8040)` gera 64 (hex 40)
`alpha(c)`	número	valor do componente alfa (transparência) (`a`): `alpha(#ff8844)` gera 1, `alpha(fade(@cor, 50%))` gera 0.5

8.2.2 Componentes HSL e HSV

Função	Recebe	Retorna
`hue(c)`	cor	número inteiro com o ângulo da matiz do espaço HSL. `hue( #8080ff )` gera `240`
`saturation(c)`	cor	percentagem de saturação no espaço HSL. `saturation(#8080ff)` gera `100%`
`lightness(c)`	cor	percentagem de luminosidade no espao HSL. `lightness(#8080ff)` gera `75%`

Função	Recebe	Retorna
`hsvhue(c)`	cor	número inteiro com o ângulo da matiz no espaço HSV. `hsvhue(#8080ff)` gera `240`
`hsvsaturation(c)`	cor	percentagem de saturação no espaço HSV. `hsvsaturation(#8080ff)` gera `50%`
`hsvvalue(c)`	cor	percentagem de brilho no espaço HSV. `hsvvalue(#8080ff)` gera `100%`

Exemplos:

```
@color2: rgba(64,255,64,0.5);
@color4: hsla(240deg,100%,75%,75%);
@color6: hsva(240deg,50%,50%,50%);

.color-components-rgba:after {
  content: red(@color2), green(@color4), blue(@color6), alpha(@color4);
}

.color-components-hsla:after {
  content: hue(@color4), saturation(@color4), lightness(@color6);
}

.color-components-hsva:after {
  content: hsvhue(@color2), hsvsaturation(@color4), hsvvalue(@color6);
}
```

Resultado em CSS:

```
.color-components-rgba:after {
   content: 64, 127.5, 127.5, 0.75;
}

.color-components-hsla:after {
   content: 240, 100%, 38%;
}

.color-components-hsva:after {
   content: 120, 50%, 50%;
}
```

8.2.3 Luma

A função luma retorna um percentual que indica a luminância relativo de uma cor (brilho perceptível) de acordo com a recomendações W3C de accessibilidade.

Função	Recebe	Retorna
`luma(c)`	cor	uma percentagem do brilho perceptível (luma) da cor. Ex: `luma(red)` gera `21%`

Exemplo:

```
.sec:after {
   content: luma(white), luma(yellow), luma(cyan), luma(magenta);
   content: luma(red),  luma(green),  luma(blue), luma(black);
}
```

Em CSS:

```
.sec:after {
   content: 100%, 93%, 79%, 28%;
   content: 21%, 15%, 7%, 0%;
}
```

8.3 Operações sobre cores individuais

Estas funções alteram os componentes de cores individuais e retornam uma cor modificada, mais clara ou escura, mais ou menos saturada, com a matiz ou transparência alteradas.

8.3.1 Saturação

Estas funções alteram a saturação e usam o espaço de cores HSL.

Função	Recebe	Retorna

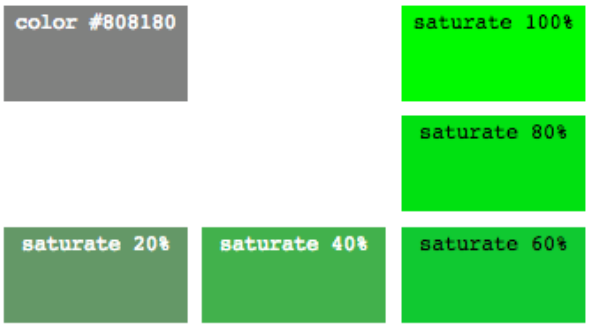
`saturate(c,p)`	`c` = cor,
	`p` = percentagem
		Cor com a saturação aumentada na percentagem indicada (até a saturação máxima).
		Exemplos:
		`saturate(#808180,20%)` gera `#679a67`
		`saturate(#808180,100%)` gera `#02ff02`

`desaturate(c,p)`	`c` = cor,
	`p` = percentagem
		cor com a saturação diminuída na percentagem indicada (até a saturação mínima).
		Exemplos:
		`desaturate(#0000ff, 60%)` gera `#4d4db3`
		`desaturate(#0000ff, 100%)` gera `#808080`

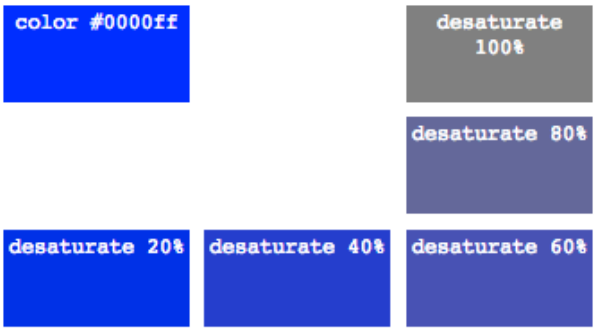
`greyscale(c)`	cor	cor completamente de-saturada. Mesmo que `desaturate(c, 100%)`. *Exemplo*: `greyscale(#ffff00)` gera `#808080`

Exemplos:

```
saturate(#808180, 0 a 100%):
```




```
desaturate(#0000ff, 0 a 100%):
```




```
greyscale(#bf2020),
greyscale(#ffff00),
greyscale(#0000ff):
```

8.3.2 Luminância

Estas funções tornam a cor mais clara ou mais escura.

Função	Recebe	Retorna
`lighten(c,p)`	`c` = cor, `p` = percentagem	Cor com a luminância aumentada na percentagem indicada (até a máxima).
`darken(c,p)`	`c` = cor, `p` = percentagem	Cor com a luminância diminuída na percentagem indicada (até a mínima).

Exemplos de `darken` e `lighten`:

```
 @color: #ff0000;
.light {
   background-color: lighten(@color, 25%);
}
.dark {
   background-color: darken(@color, 25%);
}
```

CSS:

```
.light {
   background-color: #ff8080;
}
.dark {
   background-color: #800000;
}
```

8.3.3 Matiz

A função `spin` recebe um ângulo que é aplicado sobre o círculo ou cone de cores, variando a matiz.

Função	Recebe	Retorna
`spin(c,a)`	`c` = cor, `a` = ângulo	Cor correspondente ao ângulo. Vermelho = 0 = 360. Verde = 120, Azul = -120 ou 240. Valores múltiplos de 360 terão valores iguais.

Exemplo de `spin`:

```
@color: #ff0000;
.green {
   background-color: spin(@color, 120deg);
}
.blue {
   background-color: spin(@color, -120deg);
}
```

CSS:

```
.green {
   background-color: #00ff00;
}
.blue {
   background-color: #0000ff;
}
```

8.3.4 Transparência

Função	Recebe	Retorna

`fadein(c,p)`	`c` = cor,
	`p` = percentagem
		Diminui a transparência *relativa* somando a percentagem ao valor alfa (até o máximo).
		`fadein(rgba(255,0,0,1), -10%)`
		gera `rgba(255, 0, 0, 0.9)`
		`fadein(rgba(255,255,0,0.5), 10%)`
		gera `rgba(255, 255, 0, 0.6)`

`fadeout(c,p)`	`c` = cor,
	`p` = percentagem
		Aumenta a transparência *relativa* subtraindo a percentagem do valor alfa (até o mínimo).
		`fadeout(rgba(255,0,0,1), 10%)`
		gera `rgba(255, 0, 0, 0.9)`
		`fadeout(rgba(255,255,0,0.5), 10%)`
		gera `rgba(255, 255, 0, 0.4)`

`fade(c,p)`	`c` = cor,
	`p` = percentagem
		Aplica um valor de transparência *absoluto* na cor (substitui o valor alfa existente).
		`fade(rgba(255,0,0,1), 10%)`
		gera `rgba(255, 0, 0, 0.1)`
		`fade(rgba(255,255,0,0.5), 10%)`
		gera `rgba(255, 255, 0, 0.1)`

8.4 Operações de combinação de cores

As operações de combinação de cores podem ser classificadas em simétricas (onde a ordem das cores não é relevante) e assimétricas (onde a ordem importa).

8.4.1 Simétricas

Função	Recebe	Retorna

`mix(c1, c2)`	2 cores	Mistura duas cores de forma proporcional e inclui canais alfa).
		`mix(orange, purple)` gera `#c05340`
		`mix(rgba(255,255,0,0.5),red)`
		gera `rgba(255, 64, 0, 0.75)`

Função	Recebe	Retorna

`average(c1, c2)`	2 cores	Mistura duas cores utilizando a média de cada canal.
		`average(orange, purple)` gera `#c05340`
		`average(rgba(255,255,0,0.5),red)` gera `#ff4000`

`multiply(c1, c2)`	2 cores	Cor mais escura resultante da multiplicação entre os componentes das duas cores. Oposto de `screen`. `multiply(#800080, #800080)` gera `#400040` `multiply(yellow, red)` gera `#ff0000` (red)
`screen(c1, c2)`	2 cores	Cor mais clara resultante da multiplicação do valor inverso dos componentes das duas cores. Oposto de `multiply`. `screen(#800080, #800080)` gera `#c000c0` `screen(yellow, red)` gera `#ffff00` (yellow).

`difference(c1, c2)`	2 cores	Cor resultante da subtração da segunda cor da primeira, canal por canal. Subtração de branco inverte a cor.
		`difference(#ff0000,#ffffff)` gera `#00ffff`
		`difference(#ff00ff,#00ff7f)` gera `#ffff80`

`exclusion(c1, c2)`	2 cores	cor resultante de um difference de menos contraste. `exclusion(#ff44ff,#44ff7f)` gera `#bbbb80`

`negation(c1, c2)`	2 cores	cor resultante da efeito inverso de difference.
		`negation(#ff0000, #ffffff)` gera `#00ffff`
		`negation(#ff00ff,#00ff7f)` gera `#ffff80`

8.4.2 Assimétricas

Função	Recebe	Retorna
`overlay(c1, c2)`	2 cores	Cor com canais mais escuros ou mais claros determinados pela primeira cor. `overlay(red, yellow)` gera `#ff0000` (red) `overlay(yellow, red)` gera `#ffff00` (yellow)

Função	Recebe	Retorna
`hardlight(c1, c2)`	2 cores	Cor com canais mais escuros ou mais claros determinados pela segunda cor. Oposto de `overlay`. `hardlight(red,yellow)` gera `#ffff00` (yellow) `hardlight(yellow,red)` gera `#ff0000` (red)

`softlight(c1, c2)`	2 cores	Cor resultante de `overlay` de menos contraste.
		`softlight(#400000, #808000)` gera `#400000`
		`softlight(#808000, #400000)` gera `#604000`

{9: +less }

Este capítulo contém tópicos avançados ou menos usados que não foram abordados em módulos anteriores.

9.1 Importação de estilos

Diferentemente do CSS, diretivas `@import` podem ser colocadas em qualquer lugar do arquivo Less. A geração do CSS irá posicioná-las no início do arquivo corretamente (embora seja uma boa prática sempre mantê-los no início do arquivo).

A sintaxe é a mesma do CSS:

```
@import "documento.css";
```

Less também permite que o arquivo importado seja um documento Less. Neste caso, ele funcionará como se o arquivo importado estivesse anexado antes do arquivo que o importa.

Se a extensão do arquivo for `.css`, o less irá tratá-lo como um `@import` comum do CSS, e simplesmente copiar a declaração para o CSS resultante.

Se for qualquer outra extensão, o arquivo será incorporado e tratado como sendo Less.

Se o arquivo for importado sem extensão, o less irá anexar uma extensão `.less` a ele e buscar incorporar o arquivo.

Pode-se redefinir o comportamento default do import com seis opções que são passadas entre parênteses entre a diretiva `@import` e o nome do arquivo, da forma:

```
@import (parametro) "arquivo";
```

Os seguintes parametros podem ser usados:

- **css**: trata o arquivo como CSS (independente de sua extensão)
- **inline**: inclui o conteúdo do arquivo no CSS final, mas não o processa
- **less**: trata o arquivo como LESS (independente de sua extensão)
- **multiple**: aceita que um um arquivo seja importado mais de uma vez
- **once**: permite importa o arquivo uma vez e ignora outras tentativas
- **reference**: inclui e usa o arquivo LESS mas não utiliza o conteúdo na saída

Exemplo:

```
import (reference) 'biblioteca.less';
```

9.1.1 Import e variáveis

Se a importação resultar em uma variável definida mais de uma vez, a *última* definição, considerando o escopo do atual para o mais externo, é usada. Por exemplo, na pagina `more-3-imports-base.less` foram definidas as seguintes variáveis:

```
@base: #ff0000;
@background: darken(@base, 20%);
@foreground: lighten(@base, 20%);
```

Em uma outra folha de estilos Less, essa biblioteca é importada:

```
@import "more-3-imports-base.less";
@base: blue;
.sec {
   background-color: @background;
   color: @foreground;
}
```

O valor de `@base` foi redefinido, mas `@background` e `@foreground` não foram redefinidas. Elas são calculadas com base no valor *atual* de `@base`, que agora é `#0000ff`. O resultado em CSS é:

```
.sec {
   background-color: #000099;
   color: #6666ff;
}
```

9.1.2 Bibliotecas úteis

Existem várias bibliotecas de mixins para o Less que podem ser incorpodados em quaisquer projetos Less, evitando a necessidade de se criar mixins óbvios como os que lidam com o suporte multiplataforma dos recursos do CSS3. A maioria pode ser usada através da importação de um único arquivo `.less`. Outros oferecem bibliotecas de mixins em arquivos separados, mais adequados ao uso em browsers.

A biblioteca mais simples é **less elements**. Para usá-la, baixe o arquivo `elements.less` do site http://lesselements.com e importe no seu projeto:

```
@import "elements.less";
```

Com isso pode-se usar qualquer mixin disponível, por exemplo:

```
.secao {
     .opacity(0.9); // usando mixin do less elements
     background-color: blue;
}
```

Se você precisar de mais mixins, há pelo menos três outras opções mais completas: **LessHat**, **Clearless** e **Preboot**, além do próprio **Twitter Bootstrap** que inclui mixins reutilizáveis em vários arquivos `.less` que podem ser importados.

Para usar uma biblioteca de forma estática, o ideal é importar a biblioteca inteira. O CSS gerado conterá apenas os resultados dos mixins efetivamente usados. Para usar no browser a performance é crítica, portanto o ideal é carregar apenas o necessário. A maior parte das bibliotecas também oferece os mixins em arquivos separados, para que se importa apenas o que será usado.

9.2 Acessibilidade e suporte multi-plataforma

Esta seção aborda algumas funções que ajudam a desenvolver Web sites accessíveis, mais eficientes e independentes de browser.

9.2.1 Contraste

A função `contrast` recebe uma cor e devolve outra que contrasta com a outra (default é 43% de diferença). Apenas a cor precisa ser passada como parâmetro, mas também pode-se configurar o nível de contraste informando um par de cores como referências de claro/escuro e a percentagem de contraste desejada.

Função	Recebe	Retorna

`contrast(c)`
`contrast(c,c1,c2)`
`contrast(c,c1,c2,p)`
	`c` = a cor que servirá de parâmetro para calcular o contraste.
	`c1, c2` = cores de referência claro/escuro (opcionais) - *default* é preto/branco.
	`p` = percentagem de contraste (opcional) - *default* é 43%
		cor contrastante em relação à cor de referência.

O mixin abaixo é usado para aplicar um estilo com em um bloco em que o texto contrasta com a cor de fundo:

```
.labeled-box(@color, @label) {
   background-color: @color;
   &:before {
     content: @label;
     color: contrast(@color);
   }
}
```

Pode ser usado para garantir que o texto seja sempre visível:

```
.sec-1 {
   .labeled-box(red, '#ff0000')
}
.sec-2 {
   .labeled-box(yellow, '#ffff00')
}
.sec-3 {
   .labeled-box(blue, '#0000ff')
}
```

CSS gerado:

```
.sec-1 {
   background-color: #ff0000;
}
.sec-1:before {
   content: '#ff0000';
   color: #ffffff;
}

.sec-2 {
   background-color: #ffff00;
}
.sec-2:before {
   content: '#ffff00';
   color: #000000;
}

.sec-3 {
   background-color: #0000ff;
}
.sec-3:before {
   content: '#0000ff';
   color: #ffffff;
}
```

9.2.2 Conversão de cor para formato Microsoft

A função `argb` converte uma cor em uma representação de cor proprietária compatível com aplicações *Microsoft* e *Internet Explorer*. Pode ser usada em um mixin que gera declarações CSS independentes de browser. Nesta representação o fator alfa é um número hexadecimal entre 00 e FF e é representado antes dos pares hexadecimais dos componentes RGB. O formato resultante é`#aarrggbb`.

Função	Recebe	Retorna
`argb(a,r,g,b)`	`a` = número entre 0 e 1 ou percentagem, `r,g,b` = números (0-255) ou percentagem	cor em hexadecimal no formato `#aarrggbb`. *Exemplo*: `argb(rgba(255,0,0,0.5))` gera `#80ff0000`.

9.2.3 Conversao data-uri

Esta função converte uma URL de imagem em um recurso embutido no CSS (`url(data)`). A imagem deve estar accessível ao compilador Less em caminho absoluto ou relativo à folha de estilo. Ela será codificada pelo processador em formato *Base 64*, e terá seus bytes embutidos diretamente na folha CSS gerada.

```
@uri: data-uri('../../images/less-logo.png');
.sec10 {
   image: @uri;
}
```

Resultado em CSS:

```
.sec10 {
   image: url(" ... kJggg==");
}
```

O arquivo CSS ficará maior que a imagem original. Esta é uma solução para imagens que são utilizadas muitas vezes (fontes, ícones) quando fazer download da imagem em nova requisição é uma alternativa menos eficiente.

9.3 Bibliotecas de mixins com namespaces

Um namespace em less é declarado com um mixin usando seletor de ID. Namespaces são úteis para agrupar outros mixins e evitar conflitos de nome, principalmente em aplicações que importam folhas de estilo de terceiros.

Abaixo estão dois mixins de mesmo nome, porém em namespaces diferentes:

```
#circular {
   .mask() {
     -webkit-mask-image: url(circle.svg);
   }
}

#rectangular {
   .mask() {
     -webkit-mask-image: url(rectangle.svg);
   }
}
```

Para usar deve-se usar o nome do mixin externo (namespace) anexado ao nome do mixin interno (pode-se também usar a notação`#externo > .interno`).

```
.photo1 {
    border: none;
    #circular.mask();
}

.photo2 {
   border: none;
   #rectangular.mask();
}
```

Resultado em CSS:

```
.photo1 {
   border: none;
   -webkit-mask-image: url(circle.svg);
}
.photo2 {
   border: none;
   -webkit-mask-image: url(rectangle.svg);
}
```

9.4 Uso de less no browser

O processador Less pode ser executado no browser através do módulo `less.js` . Ele pode ser baixado de http://lesscss.org, ou vinculado a um recurso CDN:

```
<script
     src="//cdnjs.cloudflare.com/ajax/libs/less.js/1.7.0/less.min.js">
</script>
```

Quaisquer folhas de estilo Less em uso na página que precisem ser processadas pelo script, devem ser carregadas na página *antes* dessa chamada:

```
<link rel="stylesheet/less" type="text/css" href="estilo1.less" />
<link rel="stylesheet/less" type="text/css" href="estilo2.less" />
```

9.4.1 Desenvolvimento e depuração no browser

Para trabalhar com Less em tempo de desenvolvimento, e ter os resultados reprocessados browser sempre que uma alteração for feita na folha de estilos, ligue o modo `watch` chamando `less.watch()` *depois* de carregar o `less.js`:

```
 <script>less.watch()</script>
```

Uma extensão para o FireBug, o FireLess permite obter informações detalhadas de erros, números de linha onde os erros ocorreram, etc. quando se trabalha com Less.

9.5 Execução de JavaScript via Less

Existe um recurso *não documentado* que permite executar trechos de JavaScript dentro de Less. Não é possível executar trechos grandes. Funções externas são possíveis, mas apenas com Less no browser ou em uma aplicações Node.js. É um recurso que não está mais documentado no site (desde Less 1.6), mas ainda pode ser usado.

Para executar JavaScript dentro de uma folha de estilos Less, chame o JavaScript dentro de um bloco delimitado por crases:

```
@texto: "Digite aqui!";
p.editor {
   &:before {
     content: `@{texto}.toUpperCase()`;
   }
}
```

As variáveis podem ser passadas de forma interpolada, como em strings. O método `toUpperCase()` em JavaScript põe a string em caixa alta. O resultado em CSS é:

```
p.editor:before {
   content: "DIGITE AQUI!";
}
```

É possível fazer coisas bem mais sofisticadas, e contornar diversas limitações do Less, pagando o preço do aumento da complexidade.

Este outro exemplo é um mixin que gera propriedades e valores (a linha é uma só e não tem quebra):

```
@coords: "TOP LEFT BOTTOM RIGHT";
.center(@position) {
   margin: auto !important;
   position: ~`'@{position};\n ' + @{coords}.toLowerCase().split(' ')
                                                .join(': 0;\n  ') + ': 0'`;
}
```

Uso do mixin:

```
.section {   .center(absolute); }
```

Resultado em CSS:

```
.section {
   margin: auto !important;
   position: absolute;
   top: 0;
   left: 0;
   bottom: 0;
   right: 0;
}
```

9.5.1 Acesso ao DOM do browser via JavaScript

JavaScript em Less *não* garante acesso ao DOM, apenas ao *core* JavaScript. Pode-se transformar strings, arrays, gerar datas e fazer contas matemáticas, mas não é possível ler variáveis do modelo de objetos:

```
@variavel: `document.images.length`;  // NÃO FUNCIONA SEM BROWSER!
```

Quando se executa o pré-processador Less no servidor ou em linha de comando *não existe página*, logo não existe DOM nem `document`. Less standalone precisa gerar um CSS estático.

Mas não há garantia que isto irá funcionar no browser. Os objetos da árvore DOM podem não ter sido criados quando a página Less gerar o CSS. É possível contornar algumas limitações, mas o comportamento ainda será dependente de plataforma.

Uma maneira de garantir que o DOM exista, é processar mais uma vez toda a folha de estilos *após* a carga. Isto pode ser feito com `less.watch()`, que carrega a folha de estilos Less periodicamente. Pode-se configurar as opções do processador para que o próximo reprocessamento demore muito:

```
<script>
   less.poll = 86400000;
   less.watch(); // reprocessa uma vez a cada 86400 segundos.
</script>
```

Se o objetivo for usar informações dinâmicas do browser para reconfigurar a folha de estilos, pode-se fazer o oposto: diminuir ao máximo o tempo entre os reprocessamentos. Assim, se o usuário redimensionar uma janela, o Less poderá usar esse dado para reposicionar os objetos.

Por exemplo, o código abaixo (só funciona em browser) usa a altura e largura da janela obtida via JavaScript para posicionar um DIV no centro da janela:

```
// Variaveis DOM  - funciona apenas no browser!
@win-height:`$(window).height()`;
@win-width:`$(window).width()`;

@div-width: (@win-width / 2px);
@div-height: (@win-height / 2px);
@div-left-position: (@win-width - @div-width) / 2px;
@div-top-position: (@win-height - @div-height) / 2px;

.box {
   background: red;
   height: @div-height;
   width: @div-width;
   position: absolute;
   top: @div-top-position;
   left: @div-left-position;
}
```

Incluindo o bloco abaixo no browser:

```
<script>
    less.poll = 10;
    less.watch()
</script>
```

o Less será reprocessado a cada dez milissegundos, fazendo com que as posições mudem dinamicamente.

É importante observar que essas técnicas não são (ainda) muito eficientes. A performance é baixa e pode ser mais simples e eficiente usar JavaScript.

9.6 Interagindo com o parser

Pode-se ter acesso ao parser do Less e configurá-lo, ler componentes de uma folha de estilos e alterar variáveis usando os recursos de scripting do Less, acessíveis tanto no servidor via Node.js, quanto no browser. Poucos desses recursos são documentados, portanto, devem ser usados com reserva.

9.6.1 Substituição de variáveis no browser

Variáveis globais podem ser modificadas via JavaScript. A alteração não altera nem recarrega a folha de estilo mas força o reprocessamento. As variáveis alteradas irão valer para todo o escopo da folha de estilos na página.

Para modificar use um script carregado *após* o `less.js` definindo um objeto `less.modifyVars` contendo a lista das variáveis a serem redefinidas:

```
<script>
 less.modifyVars({
   '@cor-default': 'rgba(128,128,128,0.9)',
   '@tam-fonte-default': '12pt',
   '@margens': '2px'
 });
</script>
```

Variáveis globais novas (que não existem na folha de estilos) podem ser inseridas de forma similar usando `less.globalVars`.

9.6.2 Funções customizadas

Pode-se também criar funções em JavaScript que serão usadas dentro de folhas de estilo Less carregadas no browser. Essas funções devem ser definidas em `less.functions` e dependem do uso de tipos de dados não-documentados para funcionar. Podem ser declaradas de duas formas:

1) Como parte do objeto `less` *antes* de carregar o `less.js`:

```
<script>
  less = {
      functions: {
         media: function(a, b) {
             return new less.tree.Dimension((a.value + b.value) / 2);
         }
      }
  };
</script>
```

2) ou *depois* de carregar o `less.js` desta forma:

```
<script>
   less.tree.functions.media = function(a, b) {
       return new less.tree.Dimension((a.value + b.value) / 2);
   };
</script>
```

Uma vez definida, a função pode ser chamada dentro da folha de estilos Less como se fosse uma função nativa:

```
.centralizar(@left, @right) {
   @a: unit(@left);
   @b: unit(@right);
   margin-left: unit(media(@a, @b), px);
}

.section {
   .centralizar(25px; 35px;);
}
```

CSS gerado:

```
.section {
     margin-left: 30px;
}
```

9.6.3 Leitura de variáveis

O parser do Less pode ser usado para carregar uma folha de estilos e ter acesso ao seu conteúdo. A folha de estilos pode ser uma string gerada na hora por JavaScript, ou mesmo um documento externo que não seja carregado na folha como Less.

O script JQuery abaixo carrega uma folha de estilos Less localizada no servidor e imprime uma lista de suas variáveis e valores:

```
<script>
 $( document ).ready(function() {
    var parser = new(less.Parser);
    $.get( "../stylesheets/more/more-6-scripts.less", function(data) {
        parser.parse(data, function (e, tree) {
           var variables = tree.variables();
           for (i in variables) {
              var name = variables[i].name;
              // Falha se variavel contiver expressoes!
              var value = variables[i].value.toCSS();
```

```
            $("#varlist").append("<li>"+name + ":" + value+"</li>");
        }
    });
});
});
</script>
```

Vários desses recursos do Less ainda são pouco documentados e podem mudar, portanto evite usá-los em produção. O processamento Less no browser também é pouco performático, sendo ideal gerar as folhas de estilo CSS offline. Processamento Less online deve apenas ser usado em desenvolvimento ou em sites com movimento muito baixo.

9.6.4 Uso em aplicações Node.js

Em aplicações JavaScript lado-servidor como aplicações *Node.js*, o Less pode ser instalado localmente em um projeto usando o Node Package Manager `npm`:

```
npm install less
```

E depois usar `require` nos scripts que usarem o Less:

```
var less = require('less');
```

A partir daí pode-se ter acesso aos recursos do Less através do objeto `less` de forma semelhante ao uso no browser:

```
var fs = require('fs');
var less = require('less');
var parser = new(less.Parser);

fs.readFile('stylesheets/styles.less', 'utf8', function(err, data) {
    if(!err) {
        parser.parse(data, function (e, tree) {
           var variables = tree.variables();
           tree.toCSS();
           ...
        });
    }
});
```

9.7 Opções de linha de comando

Duas opções tornam o compilador Less mais rigoroso durante a compilação. As opções são passadas em linha de comando separadas por espaços da forma:

```
lessc opções arquivo.less
```

A opção `-su=on` (strict units) provoca erros de compilação em cálculos com unidades incompatíveis, multiplicação e divisão com unidades. Esses erros são ignorados por default.

A opção `-sm=on` (strict math) apenas executa expressões matemáticas que estiverem *dentro de parênteses*. Isto pode tornar-se default na versão 2.0 de Less.

Com `-sm=on` o bloco Less abaixo:

```
.sec {
    font-size: 12pt + 10pt;
    margin: (12cm + 10cm);
}
```

será gerado em CSS como:

```
.sec {
    font-size: 12pt + 10pt;
    margin: 22cm;
}
```

{10: referências }

10.1 Especificações e documentação oficial

1. Documentação oficial. A documentação oficial é bastante abrangente e oferece vários exemplos simples sobre o uso dos recursos do Less. A maior parte do material desta apostila baseia-se na documentação oficial.
   http://lesscss.org
2. Especificações do CSS. Less é CSS, portanto é importante conhecer bem o CSS e suas regras já que elas também se aplicam a Less, e para descobrir erros causados por CSS gerado incorretamente.
   http://www.w3.org/Style/CSS/specs
3. Documentação do Less4J. O compilador Less em Java é um projeto paralelo com menos recursos que o Less oficial, mas a sua documentação contém tópicos e exemplos que não fazem parte da documentação original.
   https://github.com/SomMeri/less4j/wiki/_pages
4. Discussão do projeto Less no GitHub. Para tópicos não documentados, bugs, requerimentos, versões, esta é a fonte oficial (embora respostas no StackOverflow talvez sejam mais rápidas).
   https://github.com/less/less.js/issues

10.2 Artigos e exemplos

1. Tutorial de CSS do Mozilla Developer Network - MDN. Less é CSS, portanto é importante conhecer bem o CSS. O tutorial do MDN é uma das melhores referências da Web.
   https://developer.mozilla.org/en-US/docs/Web/CSS

2. Less CSS - Tips and Trics (2013) por Mária Jurčovičová. Uma lista de truques e dicas com Less que não aparecem na documentação oficial.
   http://meri-stuff.blogspot.com.br/2013/03/less-css-tips-and-trics.html
3. Less CSS - Expressions and Traps (2013) por Mária Jurčovičová. Armadilhas do Less e outros tópicos que não são discutidos na documentação oficial.
   http://meri-stuff.blogspot.com.br/2013/05/less-css-expressions-and-traps.html
4. Q&A sobre Less em StackOverflow. Na ausência de documentação e exemplos sobre Less, o banco de perguntas e respostas StackOverflow (assim como o fórum do projeto Less no Github) são complementos valiosos.
   http://stackoverflow.com/questions/tagged/less?sort=votes
5. 10 Less CSS examples you should steal for your projects (2011) por Joshua Johnson. Lista de mixins úteis. É antigo e existem fontes melhores para mixins, mas é uma boa referência para entender Less através de exemplos.
   http://designshack.net/articles/css/10-less-css-examples-you-should-steal-for-your-projects/

10.3 Tutoriais não-oficiais

1. DON'T READ this Less CSS tutorial (2011) por Jonathan Verrecchia.
   http://verekia.com/less-css/dont-read-less-css-tutorial-highly-addictive
2. A Comprehensive Introduction to Less e Mixins (2012) por Ivaylo Gerchev. Tutoriais do site-point com uma introdução a Less e mixins.
   http://www.sitepoint.com/a-comprehensive-introduction-to-less/
   http://www.sitepoint.com/a-comprehensive-introduction-to-less-mixins/
3. Learning Less (2012) por Alex Ball. Um tutorial (incompleto) com vários exemplos.
   http://www.developerdrive.com/2012/04/learning-less-an-introduction

10.4 Bibliotecas

1. less.elements: coleção popular de mixins de uso geral para usar em projetos Less. É pequena, mas talvez seja tudo o que você precisa.
   http://lesselements.com
2. LESS Hat: outra biblioteca de mixins, com bem mais recursos.
   http://lesshat.madebysource.com/
3. Clearless: outra biblioteca bem completa.
   http://clearleft.github.io/clearless/

4. Preboot: biblioteca de mixins usada pelo Twitter Bootstrap.
   http://getpreboot.com/

10.5 Ferramentas

1. less.app: compilador Less gráfico para Mac.
   http://incident57.com/less/
2. Crunch: editor de código com gerador e compressor de CSS integrado para plataformas Adobe Air (Windows ou Mac)
   http://crunchapp.net/
3. WinLess: compilador Less gráfico para Windows.
   http://winless.org/
4. lesstester.com: aplicativo online para testar código Less e gerar CSS.
   http://lesstester.com/
5. Can I use: plataforma web com estatísticas de suporte para CSS, HTML5, SVG e JavaScript em browsers do mercado.
   http://caniuse.com/

Exercícios

Parte 1 (introdução, aninhamento, variáveis, extend e mixins)

1. [*Introdução*] Teste a configuração do seu ambiente:
    a. Transforme o documento `teste.css` em um documento `.less`, defina uma variável `@cor` para as cores que se repetem e aninhe os três blocos. Utilize um compilador Less para gerar um documento CSS e teste-o para verificar se ainda funciona.
    b. Altere o arquivo `teste.html` de forma que ele carregue o documento Less (em vez do CSS) para processamento durante a carga. Altere o documento Less (mude o valor da variável `@cor`) e veja se o HTML é alterado.

2. [*Aninhamento*] O arquivo `basics.css` contém diversos blocos de seletores contextuais e pseudo-classes. Utilize-o como base para criar um documento `.less` que reorganize os 17 blocos em um único bloco partindo de `body`, utilizando o aninhamento ao máximo (para contexto e pseudo-elementos).

3. [*Variáveis*] Remova a duplicação de dados da folha de estilos `variables.css`. Crie uma folha de estlos Less equivalente que utilize variáveis para:
    a. todas as cores
    b. famílias de fontes
    c. partes de nomes de propriedades que se repetem (border, color);
    d. parte da URL que se repete nas imagens
    e. nome da propriedade `.sec1` (inclua `sec1` em uma propriedade)

4. [*Extend*] Utilize o documento `extend.less` que importa `basics.less` e usando *apenas* pseudo-elementos `:extend` (sem declarar novas propriedades), faça com que a página `extend.html`, que usa `extend.css` "herde" alguns estilos de `basics.less`. Estenda:
    a. `.article` *com* `article`
    b. `.article .contents` *com* `.contents`
    c. `.article .contents .section` *com* `section`
    d. `.article .contents .section h1` *com* `section .h1`
    e. `nav li` *com* `.nav li`
    f. `nav li a:hover` *com* `.nav li a:hover`.

Use aninhamento e faça adaptações se necessário. Verifique se a declaração usada no extend é uma correspondência *exata*.

5. [*Extend*] O documento `mixin-extend.less` utiliza um mixin `.gradient()` para construir um gradiente de cinco cores. O mixin está definido em `gradient-mixin.less` que é importado em `mixin-extend.less`. A página `mixin-extend.html` exibe um `div` branco, cujas dimensões e borda estão definidas em `mixin-extend.less`. Analise o código gerado no CSS e altere o documento `mixin-extend.less` de tal maneira que o gradiente apareça desenhado no `div`.

6. [*Mixins*] O documento `mixin-6.html` contém quatro tabelas idênticas que diferem apenas no `id`. O documento `mixin-6.less` aplica funções que variam a largura e o brilho de uma cor para cada tabela. Observe que há bastante duplicação de código. Crie um mixin para gerar o CSS para as tabelas que receba como argumentos pelo menos uma cor e uma largura. Chame o mixin para cada `id` e verifique se o CSS gerado é o mesmo.

7. [*Mixins*] Analise o documento `mixin-7.less` e identifique trechos duplicados ou trechos que possam ser embutidos em mixins. Crie os mixins indicados utilizando ou não os nomes sugeridos, e substitua os trechos duplicados ou longos por chamadas a esses mixins. Defina argumentos como indicados e use valores default se desejar.