HTML5 e CSS3

Domine a web do futuro

Casa do Código

LUCAS MAZZA

© 2012, Casa do Código
Todos os direitos reservados e protegidos pela Lei nº9.610, de 10/02/1998.
Nenhuma parte deste livro poderá ser reproduzida, nem transmitida, sem autorização prévia por escrito da editora, sejam quais forem os meios: fotográficos, eletrônicos, mecânicos, gravação ou quaisquer outros.

Casa do Código
Livros para o programador
Rua Vergueiro, 3185 - 8º andar
04101-300 – Vila Mariana – São Paulo – SP – Brasil

Agradecimentos

"Nothing of me is original. I am the combined effort of everybody I've ever known."
– Chuck Palahniuk, Invisible Monsters (1999)

Tive a ajuda de muitas pessoas durante a criação deste livro, e sou extremamente grato a elas pela sua participação durante os meses em que trabalhei nele:

Ao Adriano Almeida e ao Paulo Silveira pela oportunidade única de escrever este livro, e a ajuda de todos da Casa do Código e da Caelum.

A minha família e a minha namorada, pela paciência e ajuda durante estes meses, por todas as noites e finais de semana dedicados a este projeto. A incrível equipe da Plataformatec, que é muito boa no que faz. Por todos os projetos, apresentações, cafés e chopps compartilhados até então - que muitos outros venham no futuro!

A todos os meus colegas e amigos de projetos e empresas passadas - devo muito a todos pelo conhecimento compartilhado ao longo dos anos, que sem dúvida foi muito importante para eu chegar até aqui.

E a você, leitor, que está prestes a ler o meu livro. Muito obrigado por dedicar o seu tempo e a sua atenção a ele. Espero contribuir para que você crie ótimos projetos e faça a sua parte para criar uma web melhor.

Sumário

1 O desenvolvimento web hoje **1**

1.1 Por quê você deve aprender HTML e CSS 2
1.2 O estado dos navegadores 2
1.3 A complicação dos prefixos proprietários 3
1.4 A longa e sinuosa estrada deste livro 4

2 Os primeiros passos com o nosso site **7**

2.1 Escrevendo HTML, de dentro para fora 8
2.2 Adicionando formatações básicas 10
2.3 Bordas e margens 12
2.4 Um pouco de cor sempre é bom 14
2.5 Primeiro contato com imagens 18
2.6 Adicionando elementos secundários 22
2.7 Faça para sua cidade também! 25

3 HTML5: o que mudou? **27**

3.1 Escrevendo menos e fazendo mais 27
3.2 Atributos personalizados 29
3.3 Tags novas para elementos antigos 29
3.4 Refatoração da página de São Paulo 31
3.5 Seja pragmático 34

4 O que todo desenvolvedor precisa saber sobre CSS **35**

4.1 A incompatibilidade dos browsers e a razão dos resets de CSS 35
4.2 Compreendendo o Box model 38
4.3 Utilizando pseudo elementos 41
4.4 Desenhando uma faixa com "*before*" *e* "after" 41
4.5 Decorando mensagens 45

4.6 Criando conteúdo através de CSS 47
4.7 Arquitete o seu CSS para o futuro 49
4.8 Gere relatórios inteligentes e simples com os estilos de impressão . . 52

5 O que você consegue fazer com CSS 3 61
5.1 A regra @font-face 62
5.2 Como utilizar serviços de distribuição de fontes 63
5.3 Substituição de ícones por fontes 64
5.4 Explore novas possibilidades com bordas 67
5.5 Manipulação de cores com rgba e gradientes 71
5.6 Trabalhe com sombras e crie menus elegantes 79
5.7 Combinando tudo 86

6 Tomando controle da estrutura visual 95
6.1 A propriedade 'display' 96
6.2 Flutue elementos 97
6.3 O clearfix, uma classe obrigatória em seus projetos 106
6.4 Compreenda o uso de position 107
6.5 Crie a sua própria janela modal 110
6.6 Como escolher os métodos para posicionar os seus elementos 116
6.7 Grids - um padrão de estrutura para as suas páginas 117
6.8 Posicionando elementos com CSS 3 118

7 Melhorando os seus formulários 123
7.1 O que temos no HTML 5 123
7.2 Formulários HTML 5 nos dispositivos móveis 124
7.3 Criação do primeiro formulário 125
7.4 Alinhamento e estilos visuais nos campos e formulários 127
7.5 Exibição de mensagens de ajuda 129
7.6 Mostre mensagens de erro 132
7.7 Levando o usuário direto ao que importa com o autofocus 134
7.8 A flexibilidade do atributo placeholder 134
7.9 Aplicando CSS3 em botões 140

8 Efeitos 101: Trabalhando com animações e transições 149
8.1 Transformando elementos 150
8.2 Os efeitos rotate, scale, skew e translate em uma galeria de fotos . . . 150
8.3 Transições de estilos 155
8.4 Transições na galeria de fotos 156
8.5 Um detalhe importante sobre transições e JavaScript 160
8.6 Transformações em 3D 161
8.7 Girar formulários com apenas um clique 161
8.8 Utilizando animações 170
8.9 Começando com keyframes 170

9 O universo fora dos desktops e notebooks 175
9.1 O que é "Responsive Web Design" e porquê você deve se preocupar . 176
9.2 O funcionamento dos media queries 177
9.3 Não é uma questão de aparelhos 178
9.4 Por um futuro melhor 179

10 Ferramentas - Frameworks, Plugins e Pré-processadores 181
10.1 Twitter Bootstrap 182
10.2 HTML5 Boilerplate 183
10.3 Plugins em JavaScript 184
10.4 Modernizr 184
10.5 Polyfills 185
10.6 Pré-processadores 186
10.7 É tudo CSS e HTML 189

11 Não pare por aqui 191

Índice Remissivo 193

Bibliografia 196

Versão: 15.2.11

Capítulo 1

O desenvolvimento web hoje

Talvez eu seja um otimista, mas acredito que nos encontramos em uma época revolucionária para o desenvolvimento web. A internet se proliferou pelo mundo graças aos avanços de banda larga e os diversos dispositivos capazes de navegar pela rede - celulares, tablets, televisões e videogames. Diversas empresas do mundo digital impulsionam a evolução das tecnologias que usamos para criar uma web melhor. Profissionais em empresas como Google, Microsoft, Apple e Facebook trabalham exclusivamente em melhorias para os navegadores mais utilizados, além de participar da definição de novos padrões e disseminam bastante conhecimento junto a comunidade de desenvolvedores.

E tudo irá continuar a se expandir. Novas empresas, novos padrões, novos dispositivos. A indústria da internet vai continuar a crescer e precisamos de bastante ajuda para continuar esse trabalho. E é muito fácil garantir o seu espaço neste universo que se move como um trem bala. Basta participar.

1.1 Por quê você deve aprender HTML e CSS

Caso você seja um designer sem muita experiência com desenvolvimento, ou um programador mais acostumado a trabalhar com linhas de comando e compiladores do que navegadores, me deixe explicar um pouco a importância de se aprender HTML e CSS. Se você estiver trabalhando ou pretende trabalhar com tecnologia, acredito que boa parte - ou tudo - dos seus projetos será utilizado através de um navegador. Seja um sistema interno de um banco, uma rede social, um grande portal de notícias ou sites para campanhas de publicidade, o meio comum hoje em dia é a web, e é bastante interessante ter uma ótima base de conhecimento sobre desenvolvimento *front-end* (um dos termos usados para se referenciar a interface de uma aplicação) para contribuir para o sucesso dos projetos que você estiver participando.

Por isso, recomendo muito que você aprenda sobre o que faz a web funcionar. Além de livros como este, existem diversos cursos e sites para você aprender mais a respeito ou aprender outras tecnologias relacionadas. Imagine cuidar de um parque de servidores sem ter bons conhecimentos de sistemas operacionais e de topologia de redes, ou trabalhar em propaganda impressa sem entender sobre as cores ou o papel usado.

1.2 O estado dos navegadores

Todo o HTML e o CSS que escrevemos ganha vida dentro dos navegadores utilizados por quem acessa nossas páginas e sites, por isso é bastante importante ter um bom entendimento de como eles funcionam e, principalmente, dos buracos no meio do caminho. Desde os tempos do Mosaic (que eventualmente se tornou o Netscape) e das primeiras versões do Internet Explorer em 1995 sempre enfrentamos uma guerra de incompatibilidade entre os nevagadores e a necessidade de padrões para garantir a interoperabilidade da web entre clientes diferentes. Enquanto essa briga pode custar alguns cabelos nossos, ela também impulsiona a evolução das tecnologias que fazem os navegadores funcionar. O Google Chrome mudou a perspectiva sobre o processo de atualização contínua, garantindo que a maioria dos seus usuários possuam a última versão do navegador, e o Firefox começou a adotar um processo similar. E o componente de debug e inspeção do Firefox, o Firebug, definiu o modelo básico para que os outros navegadores implementassem ferramentas similares.

Atualmente, o Google Chrome costuma ser o campeão dos testes de compatibilidade com as últimas especificações, sempre seguido de perto das últimas versões do Firefox, Safari e Opera. Boa parte das inclusões do HTML5 e CSS3 já estão dis-

poníveis nesses navegadores por completo ou através de prefixos proprietários. O Internet Explorer, atualmente na versão 9 (no momento da escrita desse livro a versão 10 está em desenvolvimento), ainda se encontra bem atrás dos demais, mas as promessas para a sua próxima versão são interessantes.

Os perigos moram nos casos onde precisamos ir além das últimas versões e trabalhar com mais antigas, como versões do Firefox anteriores a 8 e os famigerados Internet Explorer 6 e 7. A diferença entre os navegadores e a performance do JavaScript em comparação com as suas últimas versões é assombrosa, mas em alguns casos a audiência desejada se mantém presa a esses navegadores, por complicações para se atualizar o navegador ou até o sistema operacional - As últimas versões do Internet Explorer não são compatíveis com o Windows XP por exemplo, o que atrapalha a atualização em alguns ambientes corporativos.

Por isso, é importante definir as fronteiras dos navegadores que você pretende trabalhar, e se aproveitar das técnicas adequeadas ou das soluções existentes para problemas relacionados a eles.

1.3 A complicação dos prefixos proprietários

Para quem está acompanhando a crista da onda, diversas propriedades e funcionalidades novas não se encontram completamente definidas e implementadas nos navegadores, mas isso não nos impede de utilizar versões "experimentais" delas. Para isso, cada navegador costuma expor essas novidades com prefixos específicos para diferenciar da implementação final. O que a princípio pode soar como uma ótima idéia, se tornou um fardo para todos os desenvolvedores: escrever as mesmas linhas de CSS (até) 4 vezes, assim:

```
.button {
  /* Prefixo para o Firefox. */
  -moz-transition: all 0.2s linear;
  /* Prefixo para o Chrome, Safari, Safari Mobile e Android. */
  -webkit-transition: all 0.2s linear;
  /* Prefixo para o Opera. */
  -o-transition: all 0.2s linear;
  /* Versão final, ainda nao suportada em todos os browsers */
  transition: all 0.2s linear;
}
```

Enquanto *hacks* como esse podem parecer desnecessários e culpados por sujar o código, é preciso entender a importância deles. Já estamos desfrutando e testando

essas propriedades enquanto elas são definidas e refinadas. O objetivo é, eventualmente, não precisar mais das versões prefixadas e se utilizar apenas a versão canônica das propriedades.

Mas nem todos param para acompanhar o progresso desses itens. Propriedades como `border-radius` e `box-shadow` não precisam mais de prefixos para os principais navegadores de desktops, e até o final de 2012 outras propriedades como `animation`, `transition` e `transform` estarão finalizadas e devidamente implementadas.

Para isso, se precisa levar em consideração dois itens: Mantenha-se atualizado sobre o suporte dos navegadores para novas funcionalidades - minha referência favorita é o http://caniuse.com, que possui a relação de navegadores com suporte completo, parcial ou inexistente, e que eu costumo visitar de tempos em tempos - principalmente ao começar novos projetos - para me atualizar sobre esse assunto. Outro ponto importante é que você deve mitigar ou automatizar o ruído e retrabalho gerado pelo uso de prefixos, seja com soluções como pré-processadores ou ferramentas em JavaScript, e focar o seu trabalho no que é tido como padrão, pois em algum momento os prefixos não serão mais necessários.

Para isso vamos trabalhar com os padrões definidos e eventualmente citando os prefixos quando necessários.

1.4 A longa e sinuosa estrada deste livro

Neste livro, vamos passar por diversas receitas e técnicas para você utilizar em seus projetos, e vamos ler e escrever **muito** código - muito mesmo. Usaremos tags e propriedades que estão em uso a anos, além de passar por adições recentes do HTML5 e do CSS3. A minha intenção é mostrar exemplos práticos que provavelmente você chegou a ver ou irá usar no seu dia-a-dia, mas deixo essa avaliação a você. Eu sei que vários deles já me ajudaram uma ou duas vezes por aí.

Para tudo isso, precisamos de um ponto de saída, uma tela em branco para o nosso trabalho. Além da estrutura básica de HTML abaixo, vamos utilizar algumas ferramentas para ajudar o nosso trabalho e deixar algumas partes não tão interessantes do desenvolvimento web de lado, assim podemos focar no que realmente importa. Utilizaremos o **Normalize.css** (http://necolas.github.com/normalize.css/) como um estilo de base, para garantir uma consistência melhor entre navegadores diferentes e o **-prefix-free** (http://leaverou.github.com/prefixfree/), que será responsável por tratar os prefixos proprietários para as propriedades de CSS3 que ainda não se tornaram um padrão entre navegadores. Além deles, iremos precisar de

um pouco de JavaScript para finalizar alguns dos nossos exemplos, e ai o **jQuery** (http://jquery.com/) entrará em ação.

Mas não tema! Caso tudo isso pareça demais para você, fique seguro que iremos entrar em detalhes sobre resets de CSS e sobre o -prefix-free. Por enquanto, posso te garantir que eles irão ajudar bastante o nosso trabalho ao decorrer deste livro. Abaixo, segue o mínimo necessário de código HTML que iremos precisar.

Você pode baixar este template em http://lucasmazza.github.com/template.zip.

```
<!doctype html>
<html lang='pt-BR'>
  <head>
    <meta charset="UTF-8">
    <title></title>
    <link rel="stylesheet" href="normalize.css">
    <script src="prefixfree.min.js"></script>
  </head>
  <body>
    <!-- O seu HTML vem aqui! -->
    <script src="jquery.min.js"></script>
  </body>
</html>
```

Para o CSS específico de cada exemplo, você pode criar um arquivo com um nome relacionado ao que estiver fazendo e adicionar uma referência a ele, logo abaixo do `normalize.css`. Assim, no exemplo de animações, você pode criar um arquivo `animacoes.css` e colocar uma tag `link` dentro da tag `head`.

```
<!doctype html>
<html lang='pt-BR'>
  <head>
    <meta charset="UTF-8">
    <title>Trabalhando com animações!</title>
    <link rel="stylesheet" href="normalize.css">
    <link rel="stylesheet" href="animacoes.css">
    <script src="prefixfree.min.js"></script>
  </head>
  <body>
    <!-- O seu HTML vem aqui! -->
    <script src="jquery.min.js"></script>
  </body>
</html>
```

Com isso em mãos, hora de começar a trabalhar no que realmente interessa!

Capítulo 2

Os primeiros passos com o nosso site

Antes de mergulharmos de cabeça em técnicas específicas, propriedades de CSS e combinações de tags e estilos, vamos praticar um pouco de *front-end* arroz com feijão para garantir que estamos prontos para os demais assuntos e que as bases de desenvolvimento web estejam bem claras para você.

São Paulo, São Paulo

Quando você mora bastante tempo em uma mesma cidade você sempre define uma lista de lugares e regiões da sua preferência, e sempre os indica para os seus amigos de fora quando eles chegam para passar alguns dias. Para deixar a minha lista de paradas obrigatórias de São Paulo um pouco mais interessante de se ver vamos criar o **O que eu mais gosto em São Paulo...**, um site de uma página só para falar sobre alguns lugares da cidade. A página terá uma lista de 3 lugares de maior importância, com uma breve descrição, uma foto, e uma lista adicional de mais coisas a se ver na

cidade. Nós vamos criá-la do zero, e a versão final ficará assim:

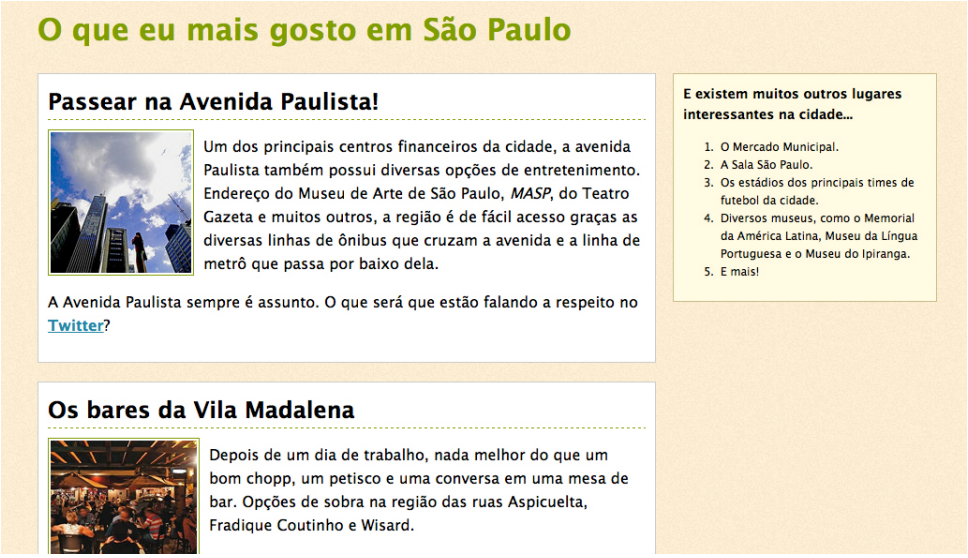


Figura 2.1: O nosso site sobre lugares de São Paulo que não se pode deixar de visitar.

A versão final da nossa página também está disponível online em http://saopaulo.herokuapp.com.

2.1 Escrevendo HTML, de dentro para fora

Vamos começar pela parte mais importante da página, o conteúdo. Que no caso, são os 3 lugares que você não pode deixar de visitar em São Paulo. O primeiro, a Avenida Paulista, ficará assim:

```
<h2>Passear na Avenida Paulista!</h2>
<p>
  Um dos principais centros financeiros da cidade, a avenida Paulista
  também possui diversas opções de entretenimento. Endereço do Museu
  de Arte de São Paulo, <em>MASP</em>, do Teatro Gazeta e muitos outros,
  a região é de fácil acesso graças as diversas linhas de ônibus que
  cruzam a avenida e a linha de metrô que passa por baixo dela.
</p>
<p>
  A Avenida Paulista sempre é assunto. O que será que estão falando a
```

```
  respeito no <a href='https://twitter.com/#!/search/Avenida Paulista'
   target='_blank'>Twitter</a>?
</p>
```

O que temos aqui? Primeiro, um título secundário utilizando a tag `h2`, acompanhado de dois parágrafos com texto, delimitados pela tag `p`. Dentro dos parágrafos, além do próprio texto, temos outras tags para definir mais elementos. Uma delas é a tag `em`, utilizada para indicar ênfase, que terá o seu texto exibido em itálico na página. O outro elemento, é um link para se navegar aos resultados de busca do Twitter para o termo "Av. Paulista", é criado pela tag `a` (derivada da palavra *âncora* em inglês, *anchor*). O atributo `href` indica qual o endereço que o link aponta, e o `target` indica que o link deve ser aberto em uma nova página. Estas são algumas das principais tags utilizadas para se definir elementos de conteúdo, então você irá vê-las bastante por aqui.

Seguindo em frente, podemos adicionar o resto do nosso conteúdo, outros 2 lugares. Um deles é a região da Vila Madalena, e o último é o Parque do Ibirapuera.

```
<h2>Os bares da Vila Madalena</h2>
<p>
  Depois de um dia de trabalho, nada melhor do que um bom chopp, um
  petisco e uma conversa em uma mesa de bar. Opções de sobra na região
  das ruas Aspicuelta, Fradique Coutinho e Wisard.
</p>
<p>
  Veja quais os melhores bares e restaurantes da região no
  <a href='http://vejasp.abril.com.br/busca?chn=bares&qu=
  &nbh=Pinheiros%20/%20Vila%20Madalena' target='_blank'>
    Guia da VEJA São Paulo.
  </a>
</p>

<h2>O Parque do Ibirapuera</h2>
<p>
  Um dos cartões postais da cidade, o parque dispõe de mais de 1,5 km{^2}
  de área verde, lagos artificiais e pistas de cooper e ciclismo. E se
  isso não fosse o suficiente, o parque costuma ser palco de diversos
  eventos culturais ao longo do ano.
</p>
<p>
  Veja no <a href='http://goo.gl/maps/HoH8' target='_blank'>mapa</a>
```

```
  como chegar ao parque.
</p>
```

Nada de novo até o momento. Novos parágrafos, títulos e links para outros lugares - a busca de bares do site da Veja São Paulo e a localização do Parque no Google Maps. Com a parte principal no seu lugar, vamos continuar na lista de melhorias a fazer para a nossa página. Todo site precisa de um título, certo? Para isso vamos utilizar a tag `h1`, que representa um título principal. Além disso, dentro do elemento `head`, podemos preencher a tag `title` com um título para a janela do navegador.

```
<!doctype html>
<html lang='pt-BR'>
<head>
  <!-- outras tags do head aqui -->
  <title>O que eu mais gosto em São Paulo</h1>
</head>
<body>
  <h1>O que eu mais gosto em São Paulo</h1>
  <!-- o nosso conteúdo aqui... -->
</body>
</html>
```

Figura 2.2: Todo o nosso conteúdo, sem estilos.

Ótimo! Já podemos dizer que estamos com uma página em mãos, certo? Bem, o HTML está no seu lugar, mas precisamos de um pouco de CSS para melhorar a apresentação do nosso conteúdo.

2.2 Adicionando formatações básicas

Para começar o CSS da nossa página vamos fixar a largura disponível para o conteúdo e centralizar o mesmo na página, deixando um espaço em branco nas laterais. Para

isso, adicionamos uma tag `div` em volta de todo o HTML que criamos dentro do `body`, com uma classe chamada `container`

```
<div class='container'>
  <!-- Todo o corpo da página vem aqui... -->
</div>
```

E o CSS para centralizar esta `div` na página:

```
.container {
  margin: 0 auto;
  width: 960px;
}
```

A propriedade `width` fixa a largura para `960px`, adequando o conteúdo a telas de pelo menos 1024px de largura, como tablets e netbooks. A margem superior e inferior está com `0`, e as margens laterais em `auto` cuidam de centralizar o elemento na página. O próximo passo é melhorar a exibição dos textos, ajustando um pouco o estilo das fontes com o seguinte código, que deve ser adicionado acima do CSS anterior:

```
body {
  font-family: "Lucida Grande", "Lucida Sans Unicode", Verdana,
    sans-serif;
  line-height: 1.6;
}
```

Aqui, estamos trocando a *família* de fontes utilizadas para uma lista de 4 fontes diferentes que são utilizadas da seguinte maneira: caso a primeira fonte listada não esteja disponível no sistema operacional do usuário, o navegador tentará utilizar a próxima e assim sucessivamente. A `Lucida Grande` será a fonte utilizada pelo Mac OS X ao exibir a nossa página, enquanto a `Lucida Sans Unicode` será vista por usuários do Windows, já que este sistema operacional não possui a primeira fonte da nossa lista. Em casos mais extremos, `Verdana` será utilizado ou o navegador irá exibir uma fonte sem serifa por padrão. Uma forma de se contornar isso é instalando diversas fontes diferentes por conta própria, mas não podemos contar com isso. Enquanto o `line-height` de `1.6`, sem nenhuma medida específica - como `px` ou `em` - indica para a altura das linhas ficarem com 160% da altura da fonte. Caso ela esteja em `16px`, o `line-height` será `25px`. Caso a altura da fonte mude, a altura da linha de

texto irá acompanhar essa mudança, respeitando a regra de sempre ser `1.6` da altura da fonte.

Não estamos alterando a altura da fonte por enquanto - a propriedade `font-size`, para utilizar o valor padrão do navegador, que costuma ser de `16px` em desktops e notebooks, enquanto outros aparelhos ou navegadores com configurações diferentes possuem um tamanho padrão diferente. Outro detalhe importante, é que estamos adicionado estes estilos a tag `body`. Desta forma, todos os elementos da página irão herdar essas regras "padrões" da nossa página, e podemos alterá-las conforme a necessidade de cada elemento.

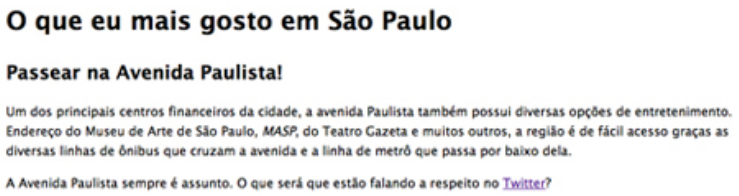


Figura 2.3: O container de 960px de largura e a alteração de fonte.

2.3 Bordas e margens

Se analisarmos com mais detalhe o nosso conteúdo, estamos falando sobre 3 lugares diferentes, onde cada um possui um título e uma descrição. Para essa definição transparecer no nosso código, podemos colocar os elementos de cada lugar dentro de um novo elemento, para agrupar cada um.

```
<div class='place'>
  <!-- O título e a descrição da Avenida Paulista... -->
</div>
<div class='place'>
  <!-- O título e a descrição da Vila Madalena... -->
</div>
<div class='place'>
  <!-- O título e a descrição do Parque do Ibirapuera... -->
</div>
```

Esta alteração não tem impacto nenhum no visual da nossa página, mas nos dá por onde adicionar um estilo para cada um dos 3 lugares, já que agora existe um

elemento específico que engloba o seu conteúdo. Assim, adicionamos um pouco de espaçamento e bordas a cada um deles.

```
.place {
  border-color: #CCC #999 #999 #CCC;
  border: 1px solid #CCC;
  margin-bottom: 20px;
  padding: 10px;
}
```

Agora, cada elemento que contenha o `class='place'`, ou lugar, terá uma borda sólida - uma linha - de `1px`. A princípio todas as bordas são do mesmo tom de cinza, mas vamos mudar as bordas da direita e de baixo para outro tom de cor, repetindo o mesmo cinza para as outras bordas, através do `border-color`. Além disso, o `margin-bottom`, coloca `20px` abaixo de cada um dos elementos, adicionando espaço entre eles e ao término da lista. Enquanto a propriedade `margin-bottom` adiciona o espaçamento do lado de fora dos elementos, a propriedade `padding` adiciona o espaçamento dentro dele, que no caso é de `10px` em todos os lados.

As três propriedades - `padding`, `margin` e `border` - funcionam de maneira similar, sendo possível definir um valor específico para cada um dos 4 lados do elemento. Caso você queria definir o valor para apenas um deles, é possível utilizar a variação `-top/right/bottom/left`, como o `margin-bottom`. Enquanto o uso da propriedade sem o sufixo vai aplicar o mesmo valor para todos os lados do elemento, ou você pode informar uma lista de valores diferentes e eles serão aplicados em sentido horário - o primeiro será para o topo (`top`), o segundo para o lado direito e assim por diante.

Apesar dos `10px` de `padding`, o espaçamento entre a borda e o título aparenta ser mais do que isso. Isso é resultado do `margin` padrão do elemento `h2` somado com o `padding` da `div` que encapsula. Para ajustar isso, podemos remover a margem do `h2` completamente. De brinde, adicionamos uma borda verde abaixo do título.

```
.place h2 {
  border-bottom: 1px dashed #7E9F19;
  margin: 0;
}
```

Assim, todos os lados do elemento `.place` terão o mesmo espaçamento de `10px`. Este estilo só se aplicará as ocorrências de `h2` **dentro** de outro elemento, que possua a classe `place`. E diferente da borda sólida de antes, utilizamos o `dashed` no lugar de `solid`, para criar um efeito diferente, com uma borda tracejada.

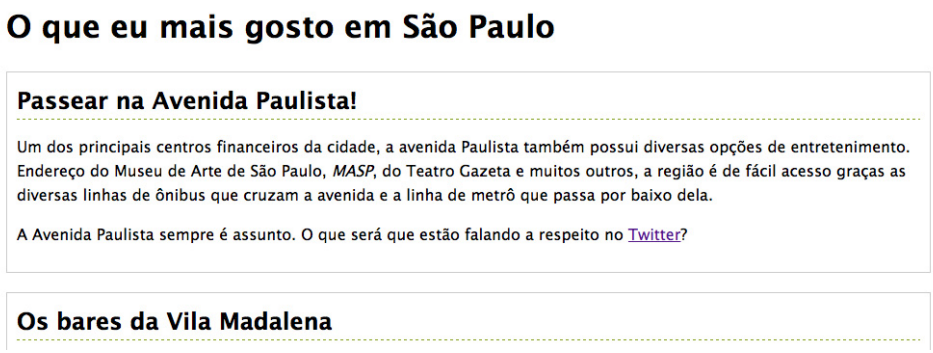


Figura 2.4: As bordas e o espaçamento da classe '.place'.

2.4 Um pouco de cor sempre é bom

Hora de trocar o preto e branco da nossa página e colocar algumas cores. Primeiro, cuidaremos do fundo da página. Além de uma nova cor nova, vamos combinar isso com uma imagem com um efeito de ruído, que será aplicado em cima da cor de fundo, assim:

```
body {
  background: #FFF1D6 url(images/noise.png);
  font-family: "Lucida Grande", "Lucida Sans Unicode", Verdana,
    sans-serif;
  line-height: 1.6;
}
```

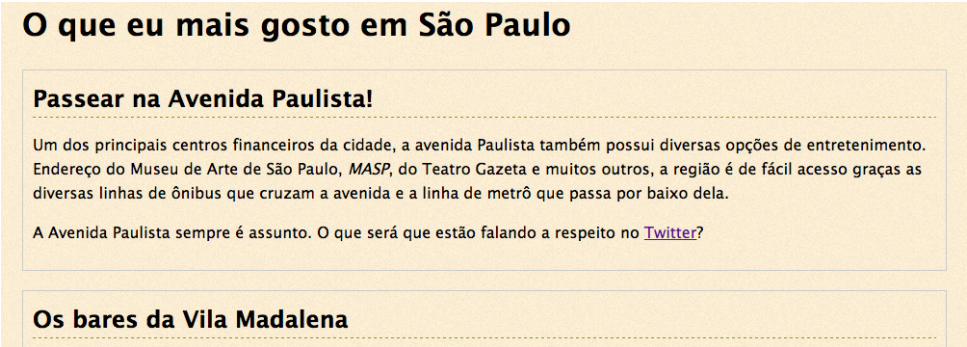


Figura 2.5: O fundo da nossa página.

Como a imagem irá preencher todo o fundo do `body`, não precisamos nos preocupar com as propriedades de `background-repeat` ou `background-position`, que são usadas para controlar mais detalhes sobre a aplicação das imagens de fundo. Lembre-se se ter a imagem `noise.png` no caminho correspondente ao que está no código - no nosso caso, em um diretório `images` que está junto ao arquivo `.html` da página. Você pode baixar este arquivo em http://saopaulo.herokuapp.com/images/noise.png.

Agora, os elementos `.place` precisam de um fundo diferente, para ajudar a leitura dos textos. Podemos então colocar um fundo branco neles.

```
.place {
  background-color: #FFF;
  border: 1px solid #CCC;
  border-color: #CCC #999 #999 #CCC;
  margin-bottom: 20px;
  padding: 10px;
}
```

Melhor, certo? Como estamos apenas mudando a cor de fundo, podemos utilizar tanto a propriedade `background` quanto o `background-color`, a escolha é livre.

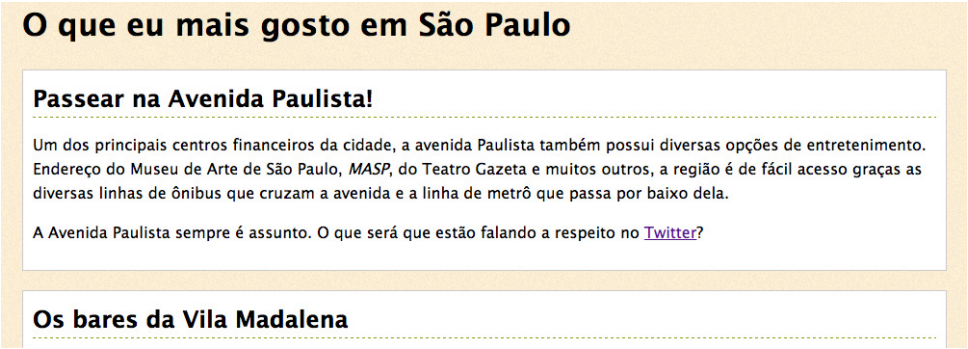


Figura 2.6: O fundo branco da div em cima do fundo da página.

Podemos também mudar a cor dos links existentes no nosso conteúdo. Desde que eles mantenham uma diferença em relação ao resto do texto, essa mudança não prejudica a usabilidade ou o visual da página. Podemos trocar o tom de azul (cor padrão dos navegadores para links), e mudar os links para negrito, o que é bastante simples.

```
.place a {
  color: #2C88A7;
  font-weight: bold;
}
```

Figura 2.7: O novo estilo dos links.

Assim, os 3 links da página estão com um estilo próprio, mas deixamos um link importante de fora, o da própria página. É comum adicionar um link ao título da página que leve para a página principal do site, que no nosso caso é a própria página em que estamos. Vamos mudar o código HTML do nosso `h1` para adicionar um link.

```
<h1><a href=''>O que eu mais gosto em São Paulo</a></h1>
```

E é claro, uma cor especial para este link:

```
h1 a {
  color: #7E9F19;
  text-decoration: none;
}
```

Figura 2.8: Agora o nosso título está verde!

Reutilizamos a cor verde das bordas tracejadas que vimos anteriormente, e tiramos o sublinhado do texto do link. Também podemos mudar a cor do link para quando se posicionar o cursor em cima dele, utilizando um pseudo-seletor do estado `hover` do elemento.

```
h1 a:hover {
  background-color: #7E9F19;
  color: #FFF;
}
```

Agora, ao se posicionar o cursor do mouse no título da página, o verde que antes estava no texto irá preencher o fundo do elemento, e o branco irá tomar o lugar do verde do texto. Quando o cursor sair de cima do título da página, o link voltará ao seu estilo tradicional.

Além do `hover`, existem pseudo seletores que podem ser utilizados para adicionar um estilo específico a outros estados de links ou outros elementos, e os mais tradicionais são:

- `:focus`, para elementos que estejam focados pelo teclado, como links ou campos de um formulário;
- `:visited`, para endereços já visitados pelo usuário;
- `:active`, para o instante onde o elemento está sendo pressionado pelo mouse.

Figura 2.9: O efeito de :hover do título.

Se for necessário trocar as cores do site, seria o caso de alterar apenas as ocorrências de `color`, `background`, `background-color` e talvez de `border`.

2.5 Primeiro contato com imagens

Um dos elementos clássicos que não utilizamos até agora é o `img`, utilizado para exibir imagens que fazem parte do seu conteúdo - diferentes de imagens utilizadas no estilo da sua página, que são aplicadas utilizando a propriedade `background` via CSS. Vamos dar uma olhada nisso enquanto adicionamos algumas fotos na descrição de cada um dos lugares de São Paulo que estamos falando. Vamos adicionar uma tag `img` entre cada `h2` e os parágrafos de cada um dos elementos `.place`.

```
<div class='place'>
  <h2>Passear na Avenida Paulista!</h2>
  <img src='images/paulista.jpg' alt="O céu da avenida Paulista,
    foto por http://www.flickr.com/photos/jairo_abud">
  <!-- os parágrafos vem aqui... -->
</div>

<div class='place'>
  <h2>Os bares da Vila Madalena</h2>
  <img src='images/piola.jpg' alt="A varanda do Armazém Piola,
    foto por Fernando Moraes">
  <!-- os parágrafos vem aqui... -->
</div>

<div class='place'>
  <h2>O Parque do Ibirapuera</h2>
  <img src='images/ibirapuera.jpg' alt="O Parque do Ibirapuera,
    por http://www.flickr.com/photos/soldon/">
  <!-- os parágrafos vem aqui... -->
</div>
```

O atributo `src` deve conter o caminho para o arquivo da imagem, enquanto o `alt` é o um texto alternativo para a imagem caso ela não seja carregada pelo navegador (no caso do caminho estar errado ou algum problema na rede do usuário). Este mesmo texto pode vir a ser usado no atributo `title`, para exibir um *tooltip* na imagem, mas deixamos ele de lado para manter o exemplo simples.

Enquanto o navegador se encarrega de descobrir a largura e a altura da imagem, também podemos fixar esses valores com os atributos `width` e `height` - caso o tamanho real da imagem seja diferente, o navegador irá redimensionar a imagem conforme o necessário. Fixar a altura e largura da imagem no HTML também é útil para que o navegador já "reserve" os pixels necessários para exibir a imagem, evitando que o conteúdo seja empurrado para os lados conforme é feito o *download* da imagem, evitando o chamando *repaint* da página. Caso queira praticar isso, nossas imagens estão com `156px` de largura e de altura.

Atenção aos tamanhos

Imagens de tamanho muito grande são um dos principais responsáveis por problemas de performance na web. Mesmo se você fixar um tamanho para as suas imagens que seja menor do que o seu tamanho real, o arquivo original será transferido pela rede e pode causar uma certa lentidão caso as suas imagens sejam pesadas, então fique atento ao tamanho das imagens de fundo, ícones e fotos das suas páginas.

Por isso, não deixe de comprimir as suas imagens para economizar diversos bytes de tráfego. Caso você utilize o Mac OS X, aconselho a usar o **ImageOptim** (http://imageoptim.com/) para esta tarefa

---

Figura 2.10: a foto da Avenida Paulista, mas parece que quebramos algo no nosso layout.

Criamos um problema no layout da nossa página com essas imagens. O `img` e os parágrafos não ficam um do lado do outro, e o texto só aparece logo abaixo da foto. Isto ocorre porquê a imagem é um elemento `inline` e o elemento `p` é `block`. Vamos entrar em mais detalhes a respeito disso na seção 6.3, mas o que precisamos agora é uma solução para a nossa página. Primeiro, vamos aplicar alguns estilos às imagens e utilizar a propriedade `float` para alinhar os elementos horizontalmente.

```
.place img {
  border: 1px solid #7E9F19;
  float: left;
  margin: 10px 10px 0 0;
  padding: 2px;
}
```

Isto vai adicionar uma borda verde na imagem, e um espaçamento entre ela e os demais elementos - como as bordas da `div` e o `h2` acima dela. Apesar de solucionar o problema de posições dentro da `div` sobre a Avenida Paulista, nos demais elementos a imagem vaza para fora da `div` - um dos problemas clássicos de se usar a propriedade `float`. Vamos resolver isto adicionando um elemento vazio com a classe `clear` logo após os parágrafos. Ele irá garantir que as imagens continuem dentro do seu respectivo `div` e que eles terão a altura adequada para conter todo o seu conteúdo.

```
<div class='place'>
  <h2>Passear na Avenida Paulista!</h2>
```

```
  <!-- a foto e os parágrafos aqui... -->
  <div class='clear'></div>
</div>

<div class='place'>
  <h2>Os bares da Vila Madalena</h2>
  <!-- a foto e os parágrafos aqui... -->
  <div class='clear'></div>
</div>

<div class='place'>
  <h2>O Parque do Ibirapuera</h2>
  <!-- a foto e os parágrafos aqui... -->
  <div class='clear'></div>
</div>
```

E o CSS da classe `.clear`:

```
.clear {
  clear: both;
}
```

Este é um truque bastante conhecido e utilizado, mas não sou muito fã dele devido ao elemento vazio extra que é adicionado ao HTML, sem nenhum valor semântico, que existe apenas para resolver um problema do estilo da página. Vamos voltar a este assunto sobre o `float` e aprender uma solução mais prática e elegante para isto.

Figura 2.11: A foto com seu estilo, posicionada corretamente.

2.6 Adicionando elementos secundários

Bem, já temos todo o conteúdo da nossa página no seu devido lugar com um pouco de CSS para deixar tudo mais apresentável. Precisamos agora de mais alguns elementos adicionais para complementar a nossa página, como um rodapé e uma barra lateral com mais conteúdo, no caso uma lista adicional de outros lugares de São Paulo, mas sem descrições ou fotos como os que temos atualmente. Vamos primeiro ao rodapé, o mais simples dos dois, com o seguinte código, a ser adicionado no final do elemento `.container`.

```
<div class='footer'>
  <p>Parte do livro "HTML5 e CSS3: Domine a web do futuro."</p>
</div>
```

E um pouco de CSS, é claro.

```
.footer {
  font-size: 12px;
  text-align: center;
}
```

O rodapé terá uma fonte menor e será centralizado. Diferente da margem automática que utilizamos no `.container` anteriormente, estamos centralizando o texto, e não o elemento do rodapé inteiro.

O próximo da nossa lista é a barra lateral. O HTML dela deve ser adicionando antes do rodapé que acabamos de criar.

```
<div class='sidenote'>
  <h3>E existem muitos outros lugares interessantes na cidade...</h3>
  <ol>
    <li>O Mercado Municipal.</li>
    <li>A Sala São Paulo.</li>
    <li>Os estádios dos principais times de futebol da cidade.</li>
    <li>Diversos museus, como o Memorial da América Latina, Museu da
        Língua Portuguesa e o Museu do Ipiranga.
    </li>
    <li>E mais!</li>
  </ol>
</div>
```

Mais uma `div`, um título em um `h3` e uma lista ordenada, a tag `ol`, com alguns itens a mais. E é claro que este elemento acompanha um pouco de CSS.

```
.sidenote {
  background-color: #FFFBE4;
  border: 1px solid #C9BC8F;
  padding: 10px;
}

.sidenote h3 {
  font-size: 14px;
  margin-top: 0;
}

.sidenote ol {
  font-size: 12px;
}
```

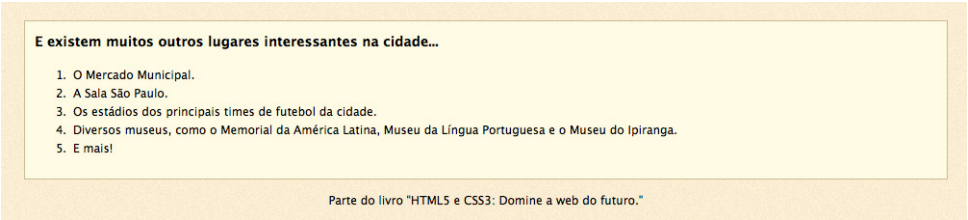


Figura 2.12: O estilo da nossa barra lateral.

Um pouco de cor e espaçamento, da mesma forma que vimos anteriormente, e alguns ajustes do tamanho das fontes já que este elemento não deve chamar mais atenção do que o nosso conteúdo principal. E por ser uma barra *lateral*, ela não deveria pegar toda a largura disponível, e se encaixar ao lado do nosso conteúdo principal. Outro caso de uso para o `float`, como utilizamos no caso das imagens. Primeiro, precisamos colocar todo os elementos com a classe `place` dentro de uma `div` nova para separá-los do resto do conteúdo, e então adicionar o estilo necessário para este elemento ficar sempre a esquerda da página.

```
<div class='places'>
  <!-- conteúdo principal vem aqui... -->
</div>

.places {
  float: left;
  width: 660px;
}
```

Agora, o conteúdo principal terá `660px` de largura, e o restante (dos `960px` que definimos para a largura total da nossa página) fica disponível para a barra lateral. Então vamos atualizar o estilo dela para posicioná-la no seu lugar de direito.

```
.sidenote {
  background-color: #FFFBE4;
  border: 1px solid #C9BC8F;
  float: right;
  padding: 10px;
  width: 260px;
}
```

Com a propriedade `float` com o valor `right`, a barra lateral vai ficar sempre no lado direito, enquanto o conteúdo está a esquerda. E a largura fixa, que é menos do que o espaço restante da página, deixa um espaço a mais entre os elementos e garante que as suas bordas não vão se juntar e deixar o visual um pouco confuso.

Só o que falta é colocar o rodapé de volta no seu lugar. Com o uso do `float` nos outros elementos, ele agora foi parar abaixo da barra lateral, enquanto deveria estar abaixo de tudo, como antes. Simplesmente adicionamos a classe `clear` ao elemento do rodapé, e voilà!

```
<div class='footer clear'>
  <p>Parte do livro "HTML5 e CSS3: Domine a web do futuro."</p>
</div>
```

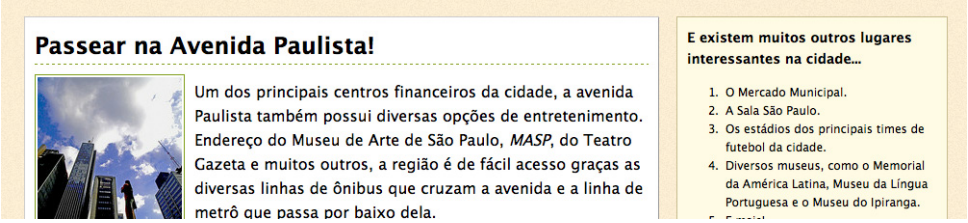


Figura 2.13: A barra lateral, no seu devido lugar.

2.7 Faça para sua cidade também!

Foi simples, não? Após apenas algumas páginas, já temos um exemplo simples, elegante e funcional de um layout com uma pequena pitada de CSS. Vamos aprender muito mais nos próximos capítulos, inclusive técnicas que poderiam deixar esse exemplo muito mais interessantes.

Caso você esteja começando nesse mundo de HTML e CSS, que tal implementar uma página similar para a sua cidade? Coloque-a no ar e eu ficarei muito contente de fazer um link para ela. Apenas me conte por e-mail o endereço da sua página. Você pode enviar para casadocodigohtmlcss@gmail.com.

Capítulo 3

HTML5: o que mudou?

O surgimento do HTML5 mudou muitas coisas no mundo do desenvolvimento web, com novos elementos, novas funcionalidades e diversas outras novidades que possibilitam experiências melhores e integrações que antes eram apenas desejos e sonhos dos desenvolvedores por aí. Mesmo com algumas funcionalidades ainda em processo de definição, já podemos aproveitar diversas das novidades que o HTML5 trouxe para o mundo.

Vamos passar por algumas delas, inclusive algumas que são referente ao código que escrevemos: novos elementos, mudanças de sintaxe e atributos personalizados. Mas não deixe de conferir no capítulo 11 sobre por onde aprender mais a respeito de outras novidades.

3.1 Escrevendo menos e fazendo mais

Uma das minhas mudanças prediletas é como a tarefa de escrever HTML ficou mais simples. Diversos pontos pequenos se tornaram opcionais ou desnecessários e o que

antes era repetitivo deixou de ser parte da nossa rotina e como consequência, o nosso código se torna mais simples e fácil de se ler.

Um ótimo exemplo disso é o `Doctype` do HTML5, a instrução que informa aos navegadores como ele deve processar o código HTML, que ficou bastante simples em relação ao seus antecessores. No HTML 4 ou no XHTML 1, a linha do `Doctype` é algo assim:

```
<!- DTD do HTML 4, em modo "strict".
<!DOCTYPE HTML
PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">

<!-- DTD transitional do XHTML -->
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

Dificilmente alguém vai conseguir lembrar de tudo isso, e geralmente deixamos a responsabilidade de escrever a linha do `Doctype` para ferramentas de geração de projetos ou de código. Agora, ele está tão pequeno que é bem fácil escrever ele de cor.

```
<!DOCTYPE html>
```

Não existe mais URL de `dtd` ou tipo de `Doctype` para se preocupar. Bastante simples!

Além disso, as tags de `link` e `script` também emagreceram um pouco. Agora, para referenciar arquivos de CSS e JavaScript, você não precisa mais informar o atributo `type`. Caso ele não esteja presente, o navegador irá assumir que o arquivo é do tipo `text/css` ou `text/javascript`.

```
<!-- O que antes era assim... -->
<link rel="stylesheet" href="normalize.css" type="text/css" />
<script src="prefixfree.min.js" type="text/javascript"></script>

<!-- ...pode ser escrito assim -->
<link rel="stylesheet" href="normalize.css">
<script src="prefixfree.min.js"></script>
```

Se você comparar os exemplos anteriores da tag `link`, pode notar que a barra no final da tag também não está mais presente - mais um detalhe que se tornou opcional no HTML5. Tags que não precisam ser fechadas, como `link`, `input`, `img`, `br` e `meta`, não precisam mais terminar em />, apenas >.

3.2 Atributos personalizados

Em alguns casos precisamos adicionar metadados ou informações extras no nosso HTML, mas não queremos depender de elementos adicionais que não possuem valor de verdade a nossa página. Um exemplo muito comum, é fazer com que alguma biblioteca leia uma tag a procura de alguns atributos específicos, como por exemplo, um atributo indicando que a requisição de um link deve ser feita assincronamente via AJAX.

```
<a href='posts/2' remote='true'>Apagar</a>
```

O atributo `remote` não existe na definição da tag `a`. Esta abordagem funcionaria em páginas de HTML 4, por exemplo, mas o seu código estaria inválido de acordo com a ferramenta de validação do W3C.

Uma forma de resolver isto são com atributos personalizados, que fazem parte do domínio do seu projeto e ao mesmo tempo mantém o seu código válido. Por exemplo, para definir um link que irá apagar um post do seu blog, podemos utilizar algumas configurações via atributos personalizados com o prefixo `data-`.

```
<a href='posts/2' data-remote='true' data-method='delete'
  data-confirm='Você tem certeza disto?'>Apagar</a>
```

Desta forma, o seu JavaScript pode ler os atributos de `data-*` e executar o código necessário para apagar o post, pedindo uma confirmação do usuário com uma mensagem personalizada.

3.3 Tags novas para elementos antigos

No campo da semântica existe um problema complicado de se lidar, que é o uso excessivo da tag `div`. Um verdadeiro coringa do HTML, que por ser um elemento genérico, acabamos utilizando para definir quase toda a estrutura das nossas páginas, desde o cabeçalho ao rodapé.

Embora isso nunca tenha impedido ninguém de colocar um site de sucesso no ar, acaba gerando um certo ruído e uma necessidade de se utilizar classes CSS para identificar o papel de cada `div` dentro do que criamos. Com isso, o HTML5 traz alguns elementos para suprir essa falta de tags mais semânticas e descritivas para alguns elementos clássicos em diversos cenários, como cabeçalhos, menus de navegação e áreas de conteúdo secundário.

Algumas das alternativas existentes são:

- `section` - utilizado para representar uma seção genérica, geralmente com um cabeçalho próprio e o seu conteúdo;
- `nav` - representação de um bloco principal de links de navegação - nem todo grupo de links deve ser tratado como um `nav`;
- `aside` - a tag `aside` pode ser utilizada para representar uma seção de conteúdo secundário ou auxiliar a outro pedaço de maior importância. Citações, links de referência ou notas adicionais, por exemplo;
- `header` - referente ao cabeçalho de uma seção específica (ou da própria página), contendo títulos, introduções e outros elementos similares;
- `footer` - o rodapé referente a um bloco de conteúdo;
- `article` - finalmente, o `article` identifica o conteúdo em si, como uma notícia de um portal, um post em blog ou um comentário em uma lista de comentários.

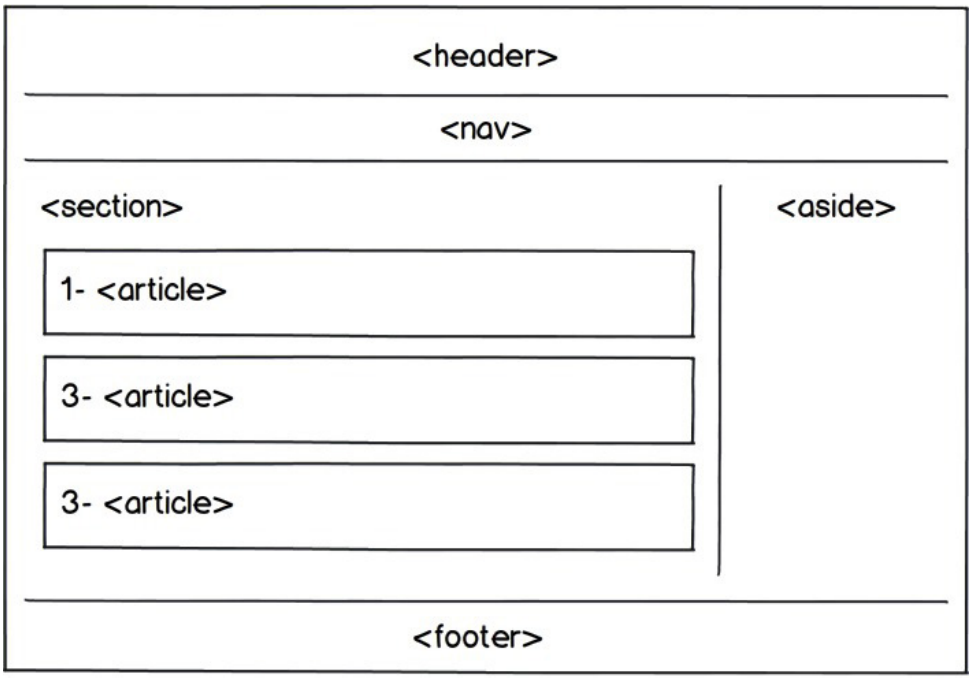


Figura 3.1: O papel de cada elemento novo em uma página completa.

Um ponto crucial destas mudanças é que podemos definir diversas seções independentes entre si, cada uma com a sua hierarquia própria. Por exemplo, podemos ter 2 elementos `h1`, cada um em sua seção, ou podemos criar um `header` para o cabeçalho de um blog, com o título e links de navegação, e cada post, devidamente criado em um `article`, ter um `header` com o título do post e a data em que ele foi publicado, por exemplo. Isso pode ser bastante útil para se arquitetar páginas mais modulares, para melhorar a qualidade e a facilidade de manutenção do nosso código.

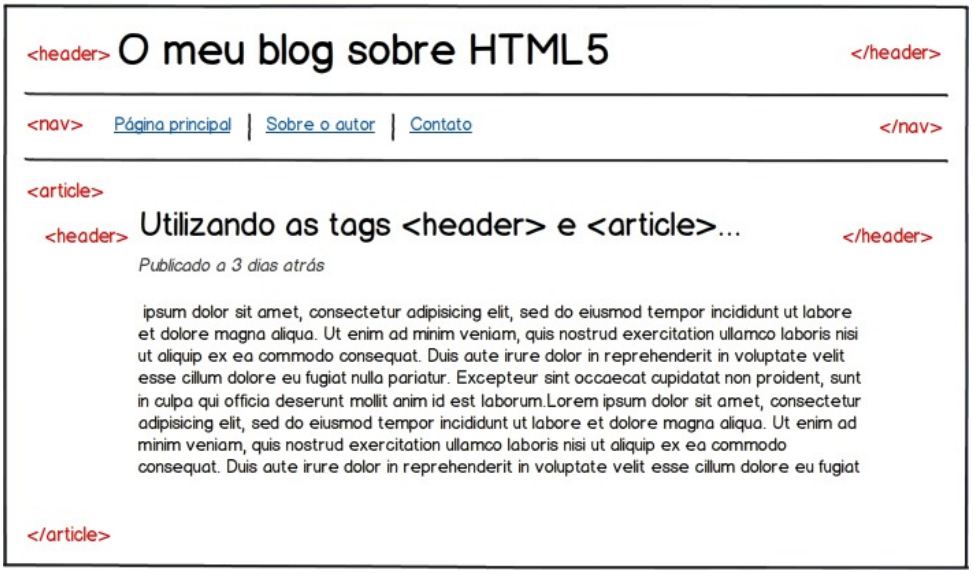


Figura 3.2: Um mockup do exemplo descrito - a tag '<header>' pode aparecer em 2 seções diferentes.

Com base nesta lista fica fácil de pensar em diversos lugares onde uma tag `div` foi utilizada com alguma classe para identificar o seu propósito. O que acha de praticar um pouco esses elementos novos na nossa página sobre São Paulo?

3.4 Refatoração da página de São Paulo

Nossa página *O que eu mais gosto em São Paulo* tem alguns pontos que podemos substituir o uso da tag `div` por alguns dos elementos novos do HTML5. Não teremos nenhuma mudança visual na página, apenas a semântica do código escrito irá mudar. A diferença no tamanho do arquivo ou na quantidade de código escrito é

insignificante, então não vamos nos preocupar com isso.

Vamos começar pelo nosso conteúdo principal, os 3 lugares de São Paulo que são descritos com a classe `.place`. Podemos alterar esses elementos para utilizar a tag `article`, removendo a necessidade da classe para identificar estes elementos.

```
<article>
  <h2>Passear na Avenida Paulista!</h2>

  <img src='images/paulista.jpg' alt="O céu da avenida Paulista, foto
    por http://www.flickr.com/photos/jairo_abud">
  <p>
    Um dos principais centros financeiros da cidade, a avenida Paulista
também possui diversas opções de entretenimento. Endereço do Museu de
Arte de São Paulo, <em>MASP</em>, do Teatro Gazeta e muitos outros,
a região é de fácil acesso graças as diversas linhas de ônibus que
cruzam a avenida e a linha de metrô que passa por baixo dela.
  </p>
  <p>
    A Avenida Paulista sempre é assunto. O que será que estão falando a
respeito no <a href='https://twitter.com/#!/search/Av.%20Paulista'
target='_blank'>Twitter</a>?
  </p>
</article>
```

E claro, alterar o nosso CSS para refletir esta mudança, trocando onde antes usávamos a classe `place` pela tag `article`.

O próximo lugar que podemos alterar é o elemento com a classe `.sidenote`, a nossa barra lateral, que possui um conteúdo adicional ao que mexemos anteriormente. Um ótimo candidato a se tornar um `aside`.

```
<aside>
  <h3>E existem muitos outros lugares interessantes na cidade...</h3>
  <ol>
    <li>O Mercado Municipal.</li>
    <li>A Sala São Paulo.</li>
    <li>Os estádios dos principais times de futebol da cidade.</li>
    <li>Diversos museus, como o Memorial da América Latina, Museu da
Língua Portuguesa e o Museu do Ipiranga.</li>
    <li>E mais!</li>
  </ol>
</aside>
```

Sem deixar de alterar o CSS, trocando as referências a classe `.sidenote` pela tag `aside`.

```
aside {
  background-color: #FFFBE4;
  border: 1px solid #C9BC8F;
  float: right;
  padding: 10px;
  width: 260px;
}

aside h3 {
  font-size: 14px;
  margin-top: 0;
}

aside ol {
  font-size: 12px;
}
```

E for final, o uso da classe `.footer` no nosso rodapé é outro caso óbvio de mudança.

```
<footer class='clear'>
  <p>Parte do livro "HTML5 e CSS3: Domine a web do futuro."</p>
</footer>

footer {
  font-size: 12px;
  text-align: center;
}
```

Existem outros pontos que poderíamos mudar no nosso HTML: substituir a `div` com a classe `.places` por um `section`, e colocar um `header` em volta do título da nossa página `h1`, mas não precisamos chegar a tanto.

Nossos usuários não notariam esta mudança, mas estamos agregando mais semântica ao nosso código e facilitando a leitura dele, para manutenções futuras.

Preenchendo alguns buracos

Quando um navegador não possui suporte a uma tag desconhecida para ele, o elemento é exibido na página, porem nenhum estilo é aplicado. É assim que versões antigas do Internet Explorer, por exemplo, tratam elementos *novos* como o `section` e o `header`. Se você for utilizar algumas dessas tags e precisa dar suporte a navegadores que ainda não entendem elementos novos do HTML5, a solução é utilizar um pouco de JavaScript para forçar a aplicação de estilo pelo navegador.

O script mais utilizado para isto é o **html5shiv** (https://github.com/aFarkas/html5shiv/), Que além de tratar a aplicação de estilos nesses elementos, também trata alguns problemas relacionados a impressão da página que utilize tags do HTML5.

3.5 Seja pragmático

Enquanto este mundo de novas possibilidades e tags a nossa disposição pode parecer a solução, é muito fácil se perder no mérito das discussões e definições do tipo de *"isto aqui deveria ser um article ou um section?"*. Continuar utilizar a tag `div` não causa mal nenhum, e você sempre pode voltar no seu código depois e trocar alguma tag por outra que você considere mais adequada. Divya Manian fala bastante sobre os problemas que essa busca incessante por semântica pode causar em um post na Smashing Magazine http://coding.smashingmagazine.com/2011/11/11/our-pointless-pursuit-of-semantic-value/. O ponto importante é não perder o foco do que realmente precisamos fazer.

Capítulo 4

O que todo desenvolvedor precisa saber sobre CSS

Entender a sintaxe e as propriedades do CSS é vital, porém mais importante do que isso é aprender alguns truques e técnicas para resolver problemas do dia a dia. Só saber a linguagem não é o suficiente, é preciso saber como usá-la e quais são as suas opções quando precisar colocar a mão na massa.

4.1 A incompatibilidade dos browsers e a razão dos resets de CSS

Todos os navegadores, das primeiras versões do Internet Explorer aos navegadores de celulares e tablets, possuem estilos padrões que são aplicados em todas as páginas, como o negrito da tag `strong`, a borda do `legend` e os tamanhos de fonte diferentes do `h1` ao `h6`. E é claro que existem diferenças entre os padrões de um navegador para outro. Algumas dessas diferenças, por menores que possam ser, costumam afetar o

nosso trabalho, impactando no resultado final das nossas páginas quando visualizadas em diferentes navegadores e soluções específicas para estes cenários precisam ser utilizadas.

Para entender melhor esta situação, podemos testar um pouco de HTML, sem estilos, e ver como que ele será exibido no Google Chrome e no Firefox, por exemplo.

```
<h1>Título</h1>
<fieldset>
  <legend>Campos do formulário</legend>
  <abbr title="Hypertext Markup Language">HTML</abbr>
  <p><label>Faça a sua pesquisa:</label><input type="search"></p>
  <textarea>Um pouco de texto</textarea>
</fieldset>
```

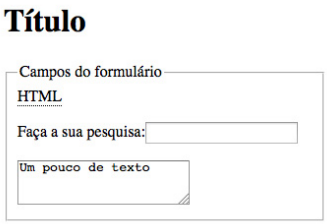


Figura 4.1: O resultado no Firefox 9.0.1

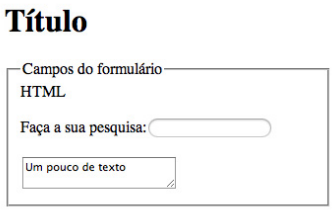


Figura 4.2: O resultado no Chrome 19.0

Conseguimos notar leves diferenças, que podem impactar o nosso trabalho final. As bordas do `fieldset`, a ausência do sublinhado no elemento `abbr` no Chrome e as diferenças dos campos de busca e de texto.

Como qualquer problema recorrente no mundo de desenvolvimento, soluções já foram criadas para não nos preocuparmos com coisas assim e focarmos no trabalho de verdade. Os arquivos de *reset*, como são chamados, possuem uma gama de regras para alinhar os navegadores em um mesmo patamar de estilo, seja corrigindo problemas ou resolvendo inconsistências.

Talvez o mais famoso dos *resets* seja o Escrito pelo Eric Meyer, que criou a versão *1.0* em 2008 - bastante tempo, não? - sua última atualização foi feita em 2011, e você pode encontrá-la no próprio blog do seu criador, http://meyerweb.com/eric/tools/css/reset/.

Os desenvolvedores do Yahoo! também criaram um *reset* sólido, parte do *Yahoo! User Interface Library*, disponível no site http://yuilibrary.com/. Um ponto negativo do *reset* do Yahoo! é que ele remove alguns padrões que eu considero importante em alguns elementos, como o negrito da tag `strong` e a marcação lateral de itens em uma lista.

Normalize.css

O **Normalize.css** (http://necolas.github.com/normalize.css/), é o *reset* que estamos utilizando neste livro e toma uma abordagem diferente a este problema. No lugar de sobrescrever diversas propriedades para definir um novo padrão de estilo para os navegadores, ele apenas adequa os pontos diferentes, preservando diversos estilos aplicados pelos navegadores, além de aproveitar para definir algumas melhorias sutis, removendo o `outline` de links e melhorando a formatação de elementos `pre`, por exemplo. [3]

O código fonte do Normalize é muito bem escrito e documentado, o que lhe permite remover partes dele que não te interesse ou entender melhor o que ele faz por debaixo dos panos. Confira o exemplo de código a seguir:

```
/*
 * 1. Improves usability and consistency of cursor style between
   image-type 'input' and others
 * 2. Corrects inability to style clickable 'input' types in iOS
 * 3. Removes inner spacing in IE7 without affecting normal text inputs
 *    Known issue: inner spacing remains in IE6
 */

button,
input[type="button"],
```

```
input[type="reset"],
input[type="submit"] {
    cursor: pointer; /* 1 */
    -webkit-appearance: button; /* 2 */
    *overflow: visible;  /* 3 */
}
```

Cada propriedade é devidamente documentada, e sabemos quais versões dos navegadores precisa das correções. Caso você não queria se preocupar com versões do Internet Explorer abaixo da 8, por exemplo, você pode remover do código fonte do Normalize as regras específicas destas versões.

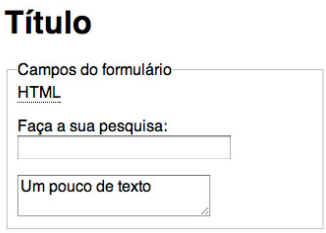


Figura 4.3: O nosso HTML de teste, agora com o Normalize.css

Não saia de casa sem o seu

Independente de qual *reset* você for usar, o mais importante é estar usando um, e faça disso parte do seu processo de desenvolvimento ao participar de novos projetos.

Isto não tira a responsabilidade de testar manualmente as páginas em diversos navegadores, mas impede que alguns problemas relacionados a posicionamento, tamanhos e fontes apareçam do nada e você tenha que dedicar tempo a esses problemas ao invés de se dedicar ao que precisa ser criado.

4.2 Compreendendo o Box model

Não importa quais cores, imagens de fundo ou posição você adicionar em um elemento, ele sempre será tratado como uma caixa. Isto pode não ser perceptível a primeira vista, mas basta começar a usar um inspetor de elementos como o Firebug ou o console do WebKit e você irá se acostumar com esta abordagem. O **Box model** é como as propriedades de CSS compõem as dimensões, onde além do `width` e do

height, as propriedades border e padding também influenciam no resultado final, mas de uma forma um tanto quanto confusa. Considere o código a seguir.

```
.button {
  border: 1px solid #999;
  height: 26px;
  padding: 5px 15px;
  width: 90px;
}
```

Um botão

Esta classe button, ao ser aplicada em um elemento qualquer, irá ocupar 122px de largura e 38px de altura na sua página. Isto acontece por que os valores de padding e border são somados a largura e altura definidas, fazendo com que as propriedades width e height sejam responsáveis por definir apenas a largura e a altura do seu conteúdo, e não do elemento como um todo. Além da confusão que pode ser gerada ao se calcular os tamanhos de antemão, isso torna bastante complicado o uso de valores com porcentagem nestas propriedades - um elemento com 100% de width e 2px de bordas ficará maior do que os 100% definidos. Complicado, não?

Outro exemplo para ilustrar este problema: ambos elementos possuem o mesmo valor de width, mas o valor do padding do segundo o deixa com 300px de largura.

```
.box {
  background-color: LimeGreen;
  width: 250px;
}

.box-with-padding {
  background-color: LightBlue;
  padding: 0 25px;
  width: 250px;
}
```

```
<div class='box'>elemento sem padding</div>
<br>
<div class='box-with-padding'>elemento com padding</div>
```

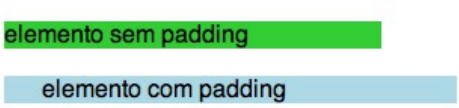


Figura 4.4: O padding do segundo elemento expande a sua largura.

Esta regra do Box model aplicar estes valores a apenas o conteúdo e não utilizar como limites fixos do elemento como um todo pode complicar diversos casos de diagramação de elementos em páginas complexas, mas é possível alterar este comportamento e não ficar recalculando larguras e alturas baseado em bordas e espaçamentos. A propriedade `box-sizing`, uma das novidades do CSS 3, permite alterar esta regra de `content-box` para o valor `border-box`, que força o navegador a respeitar estes limites e ocupar o espaço interno do elemento com os valores de `border` e `padding`, no lugar de expandir o elemento.

Desta forma, as duas definições abaixo irão gerar elementos com a mesma largura

```
* {
  box-sizing: border-box;
}

.box {
  background-color: LimeGreen;
  width: 250px;
}

.box-with-padding {
  background-color: LightBlue;
  padding: 0 25px;
  width: 250px;
}
```

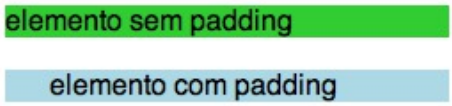


Figura 4.5: O box-sizing tornando ambos elementos do mesmo tamanho.

4.3 Utilizando pseudo elementos

Existem casos onde as tags que utilizamos para definir o nosso conteúdo não são o suficiente para estruturar os elementos de interface que desejamos. Seja para tratar problemas de float, definir bordas adicionais ou algo similar, muitas vezes acabamos dependendo de elementos vazios para resolver estes problemas. Por exemplo, ao trabalhar com bordas arredondadas sem o uso de CSS 3, já vi (e escrevi), coisas assim, onde o `span` é utilizado para criar as bordas do lado direito de um botão.

```
<div class='button'>
  <a href='#'>Clique aqui</a>
  <span></span>
</div>
```

Já foi muito comum utilizar soluções assim para trabalhar com sombras, bordas arredondadas, letras iniciais de títulos e outros detalhes de interface que não conseguimos criar apenas com o mínimo de código HTML. Mas existem alternativas bastante interessantes para isto, utilizamos apenas CSS. São *pseudo elementos*, que não existem no nosso código HTML, mas podem receber estilos específicos e auxiliar a estruturar elementos mais complexos nas nossas interfaces. Os mais tradicionais deles são os elementos `::before` e `::after`, que vivem antes e depois do **conteúdo** de uma tag (e não antes e depois da tag em si), e estão disponíveis para todas as tags que adicionarmos ao `body` da nossa página.

4.4 Desenhando uma faixa com "*before*" e "after"

Vamos criar uma faixa que dá a ilusão de que ela contorna um `section` que o contém, e os elementos `::before` e `::after` irão fazer o trabalho pesado para a gente. Começamos com o mínimo de HTML necessário, uma tag `section` e um título para a faixa.

```
<section>
  <h1>Um exemplo de pseudo elementos</h1>
</section>
```

Vamos adicionar um pouco de CSS para definir a largura do `section` e a posição do título.

```
section {
  border: 1px solid #000;
  height: 100px;
  margin: 40px auto;
  width: 400px;
}

h1 {
  font-size: 1.2em;
  text-align: center;
}
```

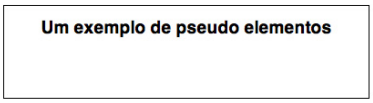


Figura 4.6: O elemento section com 400px de largura.

Vamos começar a transformar o nosso `h1` na faixa que queremos criar. Além da cor, vamos posicionar as laterais da faixa para fora da seção.

```
h1 {
  background-color: #990000;
  color: #FFF;
  font-size: 1.2em;
  left: -10px;
  padding: 5px 0;
  position: relative;
  text-align: center;
  width: 420px;
}
```

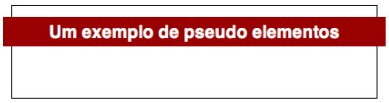


Figura 4.7: O lado frontal da faixa vermelha.

Agora, temos `10px` de cada lado da faixa *vazando* pela seção, só precisamos criar o efeito da faixa contornando o `section`, utilizando pseudo elementos. Primeiro, vamos criar este efeito para o lado esquerdo da faixa, utilizando o `::before`.

```
h1::before {
  border: 5px solid #7C0000;
  content: "";
  left: 0;
  position: absolute;
  top: -10px;
}
```

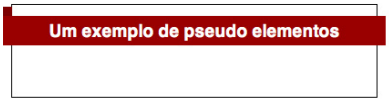


Figura 4.8: Desenhamos um quadrado no canto da faixa com o pseudo elemento '::before'.

Fazemos referência ao `before` utilizando a mesma sintaxe de pseudo classes que utilizamos para links, com o `:` separando os elementos.

Utilizando o `position` junto de coordenadas exatas com as propriedades %left e top, `conseguimos posicionar o elemento exatamente no canto superior esquerdo da faixa e com uma borda de` 5px, `desenhamos um quadrado de` 10px

A propriedade `content` é utilizada para definir o conteúdo do elemento - já que não estamos adicionando o elemento pelo HTML - e neste caso não precisamos de conteúdo algum. Para conseguir o efeito que queremos, precisamos transformar o

quadrado que criamos em um triângulo, só precisamos de um ajuste nas cores das bordas.

```
h1::before {
  border-color: transparent #7C0000 #7C0000 transparent;
  border-style: solid;
  border-width: 5px;
  content: "";
  left: 0;
  position: absolute;
  top: -10px;
}
```

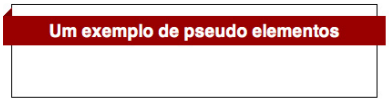


Figura 4.9: O quadrado se tornou uma aba da faixa.

Trocando algumas das cores para `transparent`, conseguimos desenhar um triângulo no lugar do quadrado vermelho. Para duplicar este efeito no outro lado da faixa, utilizando o `::after`, só precisamos inverter alguns detalhes - o que era feita com a propriedade `left` será feito com a `right`, e as posições da lista de cores também precisa ser modificada, desta forma.

```
h1::after {
  border-color: transparent transparent #7C0000 #7C0000;
  border-style: solid;
  border-width: 5px;
  content: "";
  position: absolute;
  right: 0;
  top: -10px;
}
```

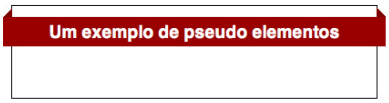


Figura 4.10: A faixa completa, utilizando '::before' e '::after'.

Feito! Criamos o efeito da faixa contornando a seção, sem precisar de novos elementos no HTML.

"`:after`" ou "`::after`"?

É comum ver por ai o uso dos pseudo elementos `before` e `after` em duas formas diferentes: prefixada com dois pontos (`:before`) ou quatro pontos (`::before`). Você sabe a diferença entre eles?

Efetivamente, nenhuma. A especificação do CSS 2 criou a versão com dois pontos, mas ela foi mudada no CSS 3 para ser diferente das definições de pseudo seletores. Navegadores modernos possuem suporte aos dois formatos, enquanto algumas versões antigas só entender o formato de dois pontos do CSS 2.

4.5 Decorando mensagens

Outra utilidade dos pseudo elementos é criar formas de botões que antes só era possível com o uso de imagens de fundo. Usando uma lógica similar a utilizada no exemplo da faixa, podemos criar um balão de ajuda com um indicação sobre o que ele é a respeito. Imagine o seguinte parágrafo, sendo usado em um formulário de cadastro de um site de um concurso qualquer:

```
<p class='help'>
  Preencha o campo com um e-mail válido, assim poderemos entrar em
  contato com você para informar o resultado do nosso concurso.
</p>
```

Podemos adicionar um pouco de estilo e transformar este parágrafo em uma caixa de ajuda.

```
.help {
  background-color: #F1EFE6;
  border: 1px solid #D3CDAE;
  font-size: 0.9em;
  padding: 10px;
  position: relative;
  width: 300px;
}
```

Preencha o campo com um e-mail válido, assim poderemos entrar em contato com você para informar o resultado do nosso concurso.

Figura 4.11: O estilo da caixa de ajuda.

Considerando que este elemento está ao lado de um campo de e-mail, podemos simular uma seta na lateral da caixa de ajuda apontando para o campo do formulário, demonstrando a relação entre a mensagem e o campo que o usuário irá preencher. Utilizar o `::before` para criar esta seta é uma ótima opção.

```
.help::before {
  border-color: transparent #D3CDAE transparent transparent;
  border-style: solid;
  border-width: 14px;
  content: "";
  left: -28px;
  margin-top: -14px;
  position: absolute;
  top: 50%;
}
```

Preencha o campo com um e-mail válido, assim poderemos entrar em contato com você para informar o resultado do nosso concurso.

Figura 4.12: A seta lateral da caixa da mensagem de ajuda.

Utilizamos a mesma lógica das bordas da faixa que criamos anteriormente para transformar este elemento em um triângulo, e definimos a posição exata dele com a combinação da propriedade `position` e das coordenadas com `top` e `left`. Mas parte do truque é a propriedade `top` com `50%` e a `margin-top` negativa, o que força o elemento a ficar sempre centralizado, independente da altura do parágrafo.

4.6 Criando conteúdo através de CSS

Além de serem bastante úteis para desenhar novos elementos na página, estes pseudo elementos também podem gerar conteúdo para complementar seus elementos de referência. A propriedade `content` aceita pedaços de texto para preencher o seu conteúdo - mas você não pode adicionar código HTML, infelizmente.

Talvez o exemplo mais clássico dessa funcionalidade é o de exibir o endereço de links da página logo após o seu texto - uma ótima pedida para estilos de impressão, que veremos logo mais neste capítulo. Mas também podemos adicionar ícones e outros tipos de elementos decorativos para complementar o nosso conteúdo. Exemplo, em um texto informativo não tão relacionado, conseguimos adicionar a indicação de uma mão apontando para o texto, assim:

```
<p class='tip'>Você sabia que...</p>
```

Para posicionar uma indicação próxima ao texto, basta o uso do `::before` junto de um símbolo Unicode que representa uma mão apontando o dedo.

```
.tip::before {
  content: "\261E";
  margin-right: 10px;
}
```

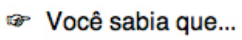


Figura 4.13: Adicionamos o símbolo unicode através da propriedade 'content'.

Pronto - não precisamos de uma imagem de um ícone específico para isso.

Outro caso de uso para gerar conteúdo com pseudo elementos é ao adicionar estilo a elementos de citação, como o `blockquote`. É bastante simples adicionar aspas em torno do texto para indicar que aquele conteúdo se trata de uma citação. Começando com o mínimo necessário de HTML:

```
<blockquote>
  O problema com citações na Internet é que você não pode
  confirmar a sua veracidade.
</blockquote>
```

E um pouco de CSS para formatar a citação:

```
blockquote {
  color: #444;
  font-style: italic;
}
```

O problema com citações na Internet é que você não pode confirmar a sua veracidade.

Figura 4.14: Um bloco de citação levemente formatado.

Podemos adicionar aspas ao redor do texto, com uma fonte maior e uma cor diferente, apenas adicionando o conteúdo necessário no `::before` e `::after` e ajustando o seu estilo como necessário.

```
blockquote::before, blockquote::after {
  color: #000;
  font-size: 3em;
}

blockquote::before {
  content: "\201C";
}

blockquote::after {
  content: "\201D";
}
```

“ ”
O problema com citações na Internet é que você não pode confirmar a sua veracidade.

Figura 4.15: As aspas adicionais via pseudo elementos.

Como o `::before` e o `::after` herdam as propriedades do seu elemento, precisamos sobrescrever a cor do texto e definir um tamanho de fonte (três vezes) maior.

Um ponto chave para posicionar pseudo elementos

Como você pode ter notado em alguns dos nossos exemplos, as propriedades `position`, `top` e `left` são bastante importantes para garantir que nossos pseudo elementos estejam em seus lugares exatos. Não deixe de conferir a seção 6.4, onde iremos conferir em detalhes o uso destas propriedades.

4.7 Arquitete o seu CSS para o futuro

Escrever CSS pode ser fácil, mas escrever CSS bem escrito é uma habilidade relativamente rara e preciosa. Conforme os seus projetos vão ganhando novas funcionalidades e sofrendo mudanças visuais, você pode se encontrar em situações onde alterar os estilos de um elemento ou adicionar uma variação de algo existente pode ser uma tarefa herculana, e a duplicação de código só traz mais complicações e problemas para a sua equipe e o seu projeto. Por isso, é importante refletir sobre o código CSS escrito e tratar a sua manutenção com a mesma atenção que se cuida do código da sua aplicação, independente da linguagem que você utilize.

Escolha bem seus seletores

Um dos vilões clássicos de um CSS mal cuidado é a complexidade de seletores usados. Além da degradação da performance das suas páginas, compreender o escopo de uma seletor para alterar alguma propriedade pode ser complicado.

Para entender o impacto de performance é necessário entender como os navegadores processam os seletores que escrevemos. Considere o seguinte seletor: `ul#nav li a`.

O navegador começará a ler o seletor de trás para frente - ao contrário do que nós estamos acostumados a ler, o que surpreende diversos desenvolvedores - e irá procurar na página *todos* os elementos da tag `a`. Após isso, ele irá filtrar esta lista por todos que estejam contidos em um elemento `li`, e então por todos que também estejam dentro de um elemento com id `nav`. E por fim, todos em que o elemento com o id `nav` seja da tag `ul`.

Considerando que o HTML para uma estrutura dessas seria uma lista não ordenada apenas com links, algo como `#nav a` seria uma solução mais simples para o navegador processar, e com o mesmo efeito. Ou até mesmo utilizando uma classe `nav-link`, por exemplo, diretamente nos elemento `a` dentro da lista com o id `nav`.

Outro ponto que gera muita discussão e confusão entre desenvolvedores: utilizar ids ou classes? Enquanto ids podem ser mais rápidos para o navegador procurar os elementos necessários na árvore do DOM, a diferença de performance é ínfima, então a discussão acaba mais no assunto de preferência pessoal. Classes são ótimas para compor diversos aspectos compartilhados entre vários elementos, e ids ajudam a indicar que o seletor é específico para um elemento único da página.

Uma regra pessoal que eu tento seguir é evitar seletores com mais de 3 partes, como `header .nav a`. Caso você se encontre escrevendo seletores maiores que algo assim, pare e reveja a sua estrutura para que não seja necessário escrever seletores tão grandes - seja utilizando mais classes ou reduzindo a especificidade dos seus seletores.

Compondo padrões visuais através de classes

Nicole Sullivan popularizou uma abordagem na escrita de CSS e composição de estilos, similar a Orientação a Objetos, que é utilizada em linguagens de programação. O OOCSS - Object Oriented CSS - propõe a separação da estrutura dos estilos e os *containers* de seu conteúdo.

O objetivo é escrever um código mais fragmentado e reutilizável, onde diversas classes diferentes são utilizadas para compor o comportamento final de um elemento, independente da tag que ele possua ou o contexto que ele se encontre - dentro de uma lista, um formulário ou o cabeçalho da página.

Tome como exemplo os 2 botões a seguir.

Figura 4.16: Como você estruturaria o CSS para estes elementos.

Podemos separar os aspectos destes elementos em 4 classes diferentes: uma para definir os estilos básicos de um botão, uma para as cores do botão principal, e outras duas para controlar os tamanhos de cada um, assim:

```
<a href="#" class='button primary-button big-button'>Enviar mensagem</a>
<a href="#" class='button small-button'>Cancelar</a>
```

```
.button {
  background-color: #999999;
  color: white;
  display: inline-block;
  font-weight: bold;
  padding: 0.5em 1em;
  text-decoration: none;
}

.primary-button {
  background-color: #389739;
}

.big-button {
  font-size: 1.1em;
}

.small-button {
  font-size: 0.9em;
}
```

Desta forma, botões em outras páginas e que possuam detalhes diferentes (como uma cor de fundo diferente, ou um ícone posicionado ao lado do texto) podem compartilhar os estilos básicos de um botão e aplicar os seus pontos específicos com outras classes, como `danger-button` ou `icon-button`. Esta abordagem pode ser estendida para menus de navegação ou conteúdos relacionados, ou uma relação de posts - o famoso **media object** dos exemplos tradicionais de OOCSS. [7]

4.8 Gere relatórios inteligentes e simples com os estilos de impressão

Os navegadores nos permitem adicionar estilos específicos para quando uma página for impressa pelo usuário. Isto pode ser feito de duas maneiras: Com o atributo `media` na tag `link` ou criando um bloco de CSS dentro da diretiva `@media print`, conforme os exemplos a seguir.

```
@media print {
  /* Este CSS só será aplicado quando a página for impressa */
}
```

```
<!-- O atributo "media" indica que estes estilos
    só devem ser aplicado para impressões -->
<link href="print.css" media="print">
```

A primeira forma é a mais adequada, já que evitamos uma nova requisição ao arquivo de impressão. E como os estilos de impressão costumam ser pequenos, não existem problemas em adicionar este CSS ao final do estilo padrão da sua página.

Pode-se considerar que a impressão de páginas é algo do passado, mas dependendo do tipo de conteúdo que você trabalha, e o seu público, isto pode ser bastante importante para o seu projeto.

Por exemplo, ao utilizar o *Airbnb* (http://airbnb.com) para alugar um apartamento para uma viagem aos Estados Unidos, é interessante imprimir o itinerário da viagem, que possui endereços, datas e telefones úteis caso você tenha algum problema ao pousar no seu destino, onde você provavelmente não terá uma conexão de internet disponível logo ao chegar. Ou ao criar um trajeto pela cidade no Google Maps, você pode imprimir o mapa com o seu roteiro para ter em mãos ao dirigir.

Além disso a impressão das páginas não fica resumida a ter o conteúdo em uma folha de papel, já que hoje em dia é bastante simples de se exportar uma página da web para um arquivo `.pdf` utilizando o processo de impressão dos navegadores. Você consegue compartilhar páginas e e-mails ou arquivar páginas de recibos de compras ou pagamentos de serviços pela internet.

Independente do tipo de conteúdo que você estiver trabalhando, existem práticas recomendadas para tratar os estilos de impressão, visando a legibilidade do seu conteúdo e tirando elementos desnecessários da frente de coisas mais importantes. Vamos repassar alguns deles.

Não conte com cores

Não podemos depender da qualidade e da gama de cores disponíveis ao imprimir a página, pois não temos controle ou informações sobre a qualidade do hardware ou da disponibilidade de tinta, então o recomendado é se manter no branco e preto de praxe. Então podemos remover todo tipo de cor ou imagem de fundo, e forçar a cor do nosso para texto, assim:

```
@media print {
  * {
    background: transparent !important;
    color: #000 !important;
  }
}
```

O uso do `!important` é para garantir que esta regra sobrescreva qualquer outro seletor mais específico do seu CSS padrão, para garantir que tudo se mantenha na cor preta. Além disso, é recomendado remover qualquer outra propriedade de estilo visual, como `box-shadow` e `text-shadow`, além de revisar a cor das bordas utilizadas.

```
@media print {
  * {
    background: transparent !important;
    border-color: #000 !important;
    box-shadow: none !important;
    color: #000 !important;
    text-shadow: none !important;
  }
}
```

Ao remover a propriedade `background` completamente, também estamos removendo imagens de fundo. Caso alguma destas imagens devesse estar presente na sua versão de impressão, considere movê-la para uma tag `img`. E o contrário também - navegações criadas com imagens deveriam ser trocadas por texto simples ao ser impresso.

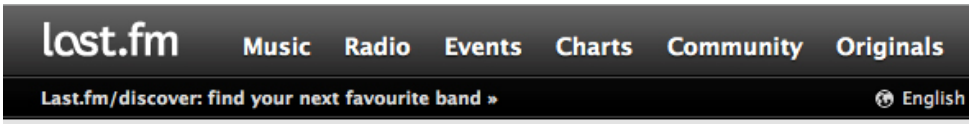


Figura 4.17: O cabeçalho do Last.fm, que utiliza diversas imagens para compor o logo e o menu principal.

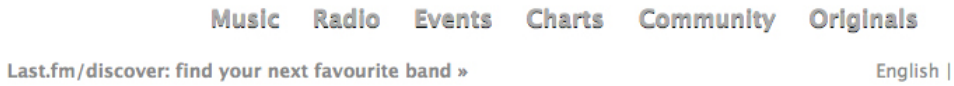


Figura 4.18: O cabeçalho impresso, sem o logo e com as imagens do menu no lugar de um texto comum.

Links devem ser links

Com a ausência de cores, todos os seus links azuis, verdes ou vermelhos terão a mesma cor preta do seu conteúdo. Uma forma de diferenciar os links do texto comum é sublinhando-os, caso você tenha retirado isto do estilo dos seus links.

```
a {
  text-decoration: underline !important;
}
```

Outra técnica recomendada para tratar links em casos de impressão é exibir o caminho que eles possuem logo após o seu texto. Isto é possível utilizando o pseudo elemento `::after` dos links, junto da função `attr()`, que permite ler os atributos de um elemento através de CSS.

```
a[href]::after {
  content: " (" attr(href) ")";
}
```

Desta forma, todos os links com um `href` presente terão o seu caminho exibido entre parênteses ao seu lado. Links que não possuem uma url a seguir, como links que utilizam funções de JavaScript não devem receber este efeito, então precisamos

remover o pseudo elemento nestes casos, utilizando um seletor mais específico que o anterior:

```
a[href]::after {
  content: " (" attr(href) ")";
}

a[href^="javascript:"]::after, a[href^="#"]::after {
  content: "";
}
```

Controlando quebras de páginas

Enquanto no navegador o nosso conteúdo existe em apenas uma página, na impressão existe a necessidade de se quebrar o conteúdo em páginas de acordo com a configuração do usuário. Por padrão isto será feito de acordo com o tamanho da sua página, mas é possível tomar controle sobre isto e definir regras específicas para informar em qual parte do seu conteúdo a quebra de página será feita. Atualmente, existem 3 propriedades disponíveis na maioria dos navegadores para isso: `page-break-before`, `page-break-inside` e `page-break-after`.

O `page-break-before` e `page-break-after` definem se a quebra de página deve ocorrer antes ou depois do elemento. A propriedade aceita os valores `always`, utilizado para forçar a quebra, ou `avoid`, indicando para o navegador que a quebra deve ser evitada. Com eles é possível, por exemplo, ao imprimir uma relação de posts de um blog, deve existir uma quebra de página entre um post e outro.

```
article {
  page-break-after: always;
}
```

Ou caso você precise de um controle mais refinado das quebras, você pode definir um elemento vazio para posicionar as quebras em pontos específicos do seu conteúdo.

```
.page-break {
  display: none;
}

@media print {
  .page-break {
```

```
    display: block;
    page-break-before: always;
  }
}
```

E então adicionar uma tag `div`, por exemplo, com a classe `.page-break` onde for necessário forçar a quebra de página.

Já com o `page-break-inside`, que aceita apenas as opções `auto` e `avoid`, podemos definir que o conteúdo de um parágrafo **não** deve ser quebrado entre duas páginas.

```
p {
  page-break-inside: avoid;
}
```

Este tipo de controle é interessante para blocos de conteúdo extensos, como manuais, documentações ou mesmo e-books criados em HTML. Além destas 3 propriedades existem outras duas, utilizadas para definir os limites de linhas que deve ficar em uma página ou em outra: `orphans` e `widows`. Mas infelizmente estas propriedades não possuem um suporte extenso pelos navegadores.

Impressão de tabelas

Vamos praticar um pouco e melhorar uma tabela de um relatório financeiro com débitos e lucros obtidos. Além dos valores, a tabela possui alguns links para navegar entre os dados e tomar outras ações no sistema. O HTML é bem simples, tendo apenas uma tabela e algumas linhas.

```
<h1>Transações de Maio, 2011</h1>
<table>
  <tr>
    <td>
      <a href='http://exemplo/transacao/1'>Transação 1</a>
    </td>
    <td class='expense'>R$ 100,00</td>
    <td class='actions'>
      <a href="http://exemplo/transacao/1/editar">Editar</a>
    </td>
  </tr>
  <tr>
```

```
    <td>
      <a href='http://exemplo/transacao/2'>Transação 2</a>
    </td>
    <td class='profit'>R$ 200,00</td>
    <td class='actions'>
      <a href="http://exemplo/transacao/2/editar">Editar</a>
    </td>
  </tr>
</table>
```

E um pouco de CSS para melhorar o estilo da tabela, definindo sua largura, cores para as bordas das células, alinhamentos e também a definição de links:

```
table {
  width: 500px;
}

td {
  border: 1px dotted #666;
  padding: 5px;
}

td a {
  color: #3B5998;
}

.expense, .profit, .actions {
  text-align: right;
}

.expense, .profit {
  font-weight: bold;
}

.expense { color: red;   }
.profit  { color: green; }

.actions a {
  font-size: 0.9em;
  color: white;
  padding: 2px 5px;
  background-color: #3B5998;
```

```
  text-decoration: none;
}
```

Transações de Maio, 2011

Transação 1	R$ 100,00	Editar
Transação 2	R$ 200,00	Editar

Figura 4.19: A tabela do relatório financeiro, em sua versão padrão.

Além de definir uma largura para a tabela e definir algumas regras de alinhamento, registros de lucro ficarão com a cor verde e gastos serão exibidos em vermelho. Ao imprimir esta página em preto e branco não possuímos nenhuma distinção visual disso e os links não estarão disponíveis no papel. Vamos escrever algumas regras de CSS para impressão e ajustar isto.

Primeiro, vamos garantir que todo o conteúdo esteja na cor preta, e que a tabela ocupe o máximo de espaço possível.

```
@media print {
  * {
    color: #000 !important;
  }

  table {
    width: 100%;
  }

  td {
    border-color: #000;
  }
}
```

Também podemos trabalhar um pouco nos links existentes na página. Os da primeira coluna podem ser expandidos utilizando a técnica do pseudo elemento. Já a coluna com os links de edição é dispensável para a nossa visão de impressão, por ser um elemento de navegação sem nenhum conteúdo relevante para este cenário, podemos então esconder esta coluna inteira.

```
a::after {
  content: "(" attr(href) ")";
  margin-left: 2px;
}

.actions {
  display: none;
}
```

Por fim, temos a coluna de receitas e débitos a tratar. Indo na onda de pseudo elementos, podemos adicionar duas características a casa célula desta coluna. Um sinal de - ou +, dependendo do caso, antes do valor, e os termos **Receita** e **Despesa**.

Com o uso do vermelho e do verde removido nos estilos de impressão, estas indicações serão úteis para identificar o tipo de transação de cada linha. O CSS para adicionar este conteúdo é o seguinte:

```
.expense::before {
  content: "- ";
}

.expense::after {
  content: " - Despesa";
}

.profit::before {
  content: "+ ";
}

.profit::after {
  content: " - Receita";
}
```

Transações de Maio, 2011

Transação 1 (http://exemplo/transacao/1)	**- R$ 100,00 - Despesa**
Transação 2 (http://exemplo/transacao/2)	**+ R$ 200,00 - Receita**

Figura 4.20: A versão de impressão finalizada.

Assim é possível adequar diversos aspectos do visual das nossas páginas para o modo de impressão, de acordo com as necessidades dos seus usuários e os objetivos do tipo de projeto que você estiver fazendo. Claro que este tipo de tratamento não é necessário em todos os projetos, mas é uma carta valiosa na sua manga.

Capítulo 5

O que você consegue fazer com CSS 3

Diversas adições do CSS 3 são extremamente úteis para substituir vários tipos de imagens que utilizávamos para adicionar cores e formas aos elementos HTML, o que não conseguíamos atingir com apenas CSS. Além da redução de arquivos para se trabalhar e a ausência de dependências externas que impactam na performance dos sites, a flexibilidade dessas propriedades permite diversas combinações diferentes, que geram muitos estilos que você não imaginaria ser tão simples de se criar com apenas CSS.

O poder de propriedades como `box-shadow` e funções como o `linear-gradient` é tão grande que é comum encontrar designers e desenvolvedores - eu faço parte deste grupo - que aposentaram os editores de imagens e criam as formas, cores e efeitos das páginas direto no código, conferindo os resultados ao vivo em seus navegadores.

5.1 A regra @font-face

Uma das funcionalidades que adiciona muita flexibilidade ao dar vida aos designs das nossas páginas é a regra `@font-face`. Ela permite definir novas famílias de fontes que as nossas páginas utilizam, e informa ao navegador e ao sistema operacional do usuário onde o arquivo necessário para desenhar os traços da sua fonte estão localizados. A definição de uma nova fonte é a seguinte:

```
@font-face {
  font-family: "Lobster";
  font-style: normal;
  font-weight: 400;
  src: local('Lobster'), url(/fonts/lobster.woff) format('woff');
}
```

Utilizando as mesmas propriedades usadas para alterar os estilos de fonte de um elemento, podemos definir os detalhes da nova fonte. No caso de definir fontes com itálico ou negrito, precisamos informar os valores corretos para as propriedades `font-style` e `font-weight` de acordo com o uso. Já a propriedade `src` é responsável por definir a localização da fonte: a função `local` informa um possível nome para a fonte que será pesquisado na máquina do usuário, ou uma URL externa para que a fonte seja baixada como um recurso adicional, igual a uma imagem de fundo, por exemplo.

Com a fonte definida, podemos utilizá-la normalmente:

```
h1 {
  font-family: "Lobster", cursive;
}
```

Caso a nossa fonte `Lobster` não esteja disponível, por um problema de rede ou a falta de um arquivo compatível, uma fonte alternativa como `cursive` deve ser definida para ser aplicada ao elemento. A compatibilidade de formatos diferentes é um ponto irritante: existem diversos formatos de fontes diferentes que você pode precisar dependendo dos navegadores que você pretende atender: `.eot` para algumas versões do Internet Explorer e `.svg`, `.ttf` ou novamente o `.eot` para o navegador nativo do Android, ou versões desatualizadas do Safari, tanto em desktops como no iOS. O formato tomado como padrão, suportado por todos os navegadores modernos, é o `woff`.

Trabalhando com @fonte-face

Trabalhando com @fonte-face

Figura 5.1: As diferenças da 'Lobster' para o 'cursive', uma alternativa aceitável caso a fonte não esteja disponível.

5.2 Como utilizar serviços de distribuição de fontes

Existem alguns serviços disponíveis na Internet que facilitam o uso de fontes adicionais, disponibilizando o código necessário, servindo os arquivos de fonte e tratando de assuntos relacionados a licenças de uso, por exemplo. Os mais famosos neste campo são o **Typekit** (https://typekit.com) e o **Google Web Fonts** (http://www.google.com/webfonts).

Alguns sites alternativos disponibilizam o download dos arquivos necessários e um exemplo de uso das fontes em CSS, mas sem servir os arquivos diretamente dos seus servidores, deixando esta responsabilidade a nós desenvolvedores, como o **FontSpring** (http://www.fontspring.com/) e **FontSquirrel** (http://www.fontsquirrel.com/).

Podemos utilizar o serviço de fontes do Google, o Google Web Fonts, para praticar um pouco. Comece com o seguinte HTML.

```
<h1>HTML & CSS</h1>
```

Agora, vamos carregar a fonte `Press Start 2P` (http://www.google.com/webfonts/specimen/Press+Start+2P), para adicionar um visual nostálgico de 8 bits ao título. Adicione a seguinte linha de HTML ao `head` da sua página.

```
<link href='http://fonts.googleapis.com/css?family=Press+Start+2P'
      rel='stylesheet'>
```

Se você acessar o endereço desta folha de estilo, http://fonts.googleapis.com/css?family=Press+Start+2P, verá que ela não faz nada além de definir a fonte `Press Start 2P`, referenciando um arquivo `.woff` presente nos servidores do Google. Podemos então utilizar esta fonte no `h1` que criamos.

```
h1 {
  font-family: 'Press Start 2P', cursive;
}
```

HTML & CSS

Figura 5.2: A fonte 'Press Start 2P' em uso.

5.3 Substituição de ícones por fontes

Outro uso bastante interessante para fontes é substituição de pacotes de ícones. No lugar de uma pilha de imagens, você pode utilizar uma fonte composta de ícones, e não caracteres normais, e ganhar o poder de controlar o tamanho e cor do seus ícones sem precisar editar imagens e exportar novos arquivos para dentro de seus projetos.

Existem diversas fontes e ferramentas existentes para ajudar a compor grupos de ícones e adicionar eles as suas páginas sem muito trabalho. Algumas das coleções de ícones mais interessantes por ai: Pictos (http://pictos.cc/), Font Awesome (http://fortawesome.github.com/Font-Awesome/) e Iconic (http://somerandomdude.com/work/iconic/). E para compor ou pesquisar outras opções, existem opções como Fontello (http://fontello.com/), IcoMoon (http://keyamoon.com/icomoon/#toHome) e o Shifticons (https://www.shifticons.com/).

Pegue a fonte Iconic, por exemplo, para testar um pouco. Você pode baixar o pacote direto do site do seu criador, P.J. Onori, em http://somerandomdude.com/work/iconic/, ou pegar a última versão da fonte no seu repositório no GitHub - https://github.com/somerandomdude/Iconic.

Em vez de utilizar o CSS que o projeto já disponibiliza para você, vamos escrever um exemplo do zero para ver o uso da fonte. Começando com a definição da fonte no CSS.

```
@font-face {
  font-family: 'IconicFill';
  src: url('iconic_fill.woff') format('woff');
  font-weight: normal;
  font-style: normal;
}
```

Então, adicionamos um elemento e uma classe para definir qual ícone será exibido junto do nosso título. Uma ótima forma de exibir tais ícones de fontes é utilizando a propriedade `content` dos pseudo elementos:

```
<h1 class='icon-next'>Avançar</h1>
```

```
.icon-next::after {
  content: '\2192';
  font-family: 'IconicFill';
  margin-left: 10px;
}
```

Avançar →

Figura 5.3: O ícone criado utilizando a fonte 'Iconic'.

Utilizando um pseudo elemento e um símbolo Unicode, o ícone de uma flecha para a direita é exibido ao lado do título. Qualquer mudança dos estilos do texto do elemento ou do seu pseudo elemento irá afetar o ícone, adequando cores e tamanhos. Podemos trocar a cor de todos os ícones associados a um `h1` e até utilizar o ícone em outro elemento, como um `p`, adicionando o seguinte código:

```
<p class='icon-next'>Ir para a próxima página</p>
```

```
h1.icon-next {
  color: blue;
}
```

Avançar →

Ir para a próxima página →

Figura 5.4: Os ícones refletem o estilo do elemento que os contém.

Desta forma conseguimos posicionar e estilizar todos os ícones disponíveis no Iconic facilmente com CSS. Outro exemplo são os clássicos ícones para ações feitas com sucesso ou erros de validações de formulários.

```
.icon-ok {
  color: #489D00;
}

.icon-ok::before {
  content: "\2714";
  font-family: 'IconicFill';
  margin-right: 10px;
}

.icon-invalid {
  color: #990000;
}

.icon-invalid::before {
  content: "\2718";
  font-family: 'IconicFill';
  margin-right: 10px;
}
```

```
<p class='icon-ok'>E-mail enviado com sucesso!</p>
<p class='icon-invalid'>Atenção, preencha todos os campos do
  formulário
</p>
```

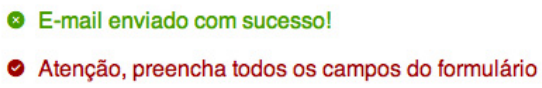


Figura 5.5: Combinando ícones e cores para criar mensagens.

Enquanto o Iconic utiliza símbolos Unicode como caracteres para os seus ícones, outras fontes podem utilizar outros caracteres para isto. É necessário conferir a documentação - caso exista uma - a respeito de cada fonte antes de utilizá-la.

5.4 Explore novas possibilidades com bordas

Todo elemento HTML é comparado a uma caixa, mas nem todos os elementos das nossas interfaces precisam ser quadrados. Em vez de se utilizar imagens a torto e direito, podemos utilizar o `border-radius` para arredondar os cantos dos seus elementos. A propriedade aceita até 4 valores, seguindo o modelo de outros como `margin` e `padding`, sendo a única diferença que o primeiro valor é referente ao canto superior esquerdo do elemento, seguindo em diante no sentido horário. Enquanto alguns cantos arredondados podem parecer apenas um pequeno detalhe estético, o `border-radius` consegue criar mais do que isso. Veja a seguir.

```
<h3 class='tnt'>TNT</h3>
```

```
.tnt {
  border-radius: 50%;
  border: 5px solid #000;
  height: 50px;
  line-height: 50px;
  text-align: center;
  width: 50px;
}
```

Figura 5.6: Utilizando border-radius para criar círculos

Podemos utilizar porcentagens e criar círculos ou elipses, independente do tamanho real do elemento. Fora isto, cada canto do elemento pode receber dois valores para definir exatamente qual será o raio utilizado para criar o efeito.

```
<span class='counter'>37</span>
```

```
.counter {
  background-color: #000;
  border-top-left-radius: 25px 10px;
  border-top-right-radius: 25px 10px;
  color: white;
  display: block;
```

```
  font-size: 1.7em;
  height: 50px;
  line-height: 50px;
  text-align: center;
  width: 50px;
}
```

Figura 5.7: Um exemplo de uso mais refinado do border-radius.

Este é um caso de uso mais raro de se encontrar, visto que a maioria das aplicações de `border-radius` são perfeitamente circulares, e geralmente feitos para realçar detalhes de certos elementos, como botões, caixas de texto ou caixas de mensagens.

Criando nuvens

Não, não vamos falar sobre *Cloud Computing* neste livro. Mas vamos desenhar um ícone de uma nuvem utilizando pseudo elementos e o `border-radius`, para demonstrar como é possível criar ícones e outras formas que geralmente se utiliza como imagens no seu código. No nosso HTML será bastante simples, sem nenhum conteúdo.

```
<div>
  <span class='cloud'></span>
</div>
```

Primeiro, com o `span` iremos desenhar a base da nuvem, e a `div` servirá como um fundo do nosso ícone.

```
div {
  background-color: #000;
  width: 50px;
  height: 10px;
  padding: 20px 50px;
}
```

```
.cloud {
  background-color: #FFF;
  border-radius: 48px;
  display: inline-block;
  height: 16px;
  position: relative;
  width: 48px;
}
```

Figura 5.8: A base da nuvem, criada com border-radius.

O ponto chave destes valores é o `border-radius` de `48px`, que é 3 vezes o valor da altura dele - `16px`. Caso fosse necessário uma nuvem maior, só iríamos manter esta proporção de valores ao definir novos tamanhos. Vamos ao próximo pedaço:

```
.cloud::before {
  background-color: red;
  border-radius: 50%;
  content: '';
  height: 14px;
  position: absolute;
  right: 7px;
  top: -6px;
  width: 14px;
}
```

Figura 5.9: O pseudo elemento posicionado acima da base.

Utilizando o `::before`, desenhamos uma esfera de `14px` posicionada de certa forma a deixar metade do elemento para fora da base. O fundo vermelho servirá apenas para ajudar a distinguir cada um dos 3 elementos entre si. Precisamos agora

de outro círculo, um pouco maior, posicionado ao lado direito do `span`. Podemos reaproveitar as definições de posição e tamanho do `::before` e sobrescrever apenas alguns valores para criar outro elemento maior.

```
.cloud::before, .cloud::after {
  background-color: red;
  border-radius: 50%;
  content: '';
  height: 14px;
  position: absolute;
  right: 7px;
  top: -6px;
  width: 14px;
}

.cloud::after {
  background-color: blue;
  height: 23px;
  left: 7px;
  right: auto;
  top: -10px;
  width: 23px;
}
```

Figura 5.10: Mais uma parte da nuvem, posicionada no outro lado da base.

Agora o `::after`, utilizando da mesma técnica para se desenhar círculos, ocupa o lado direito da nuvem. Agora basta alterar os pseudo elementos para que ambos fiquem com a mesma cor de fundo do `span` - `#FFF%` - e ver o resultado final do experimento.

Figura 5.11: Trocando os fundos para branco conseguimos ver a forma completa da nuvem.

Um caso clássico de uso para `border-radius` é na criação de estilos para componentes de formulários: de caixas de texto a botões. Esta aplicação merece uma aprofundamento maior, que você encontra na seção 7.9 deste livro.

5.5 Manipulação de cores com RGBA e gradientes

Além de fontes e bordas, outro artifício visual ganhou muito com a chegada do CSS 3 - as cores. Antigamente limitadas a apenas textos e fundos, o que sempre resultava na mesma situação das bordas arredondadas - uso de imagens para criar gradientes e opacidades, o que impacta na performance e na flexibilidade de manutenção do seu código. Funções muito úteis sugiram para suprir diversos casos de uso de imagens, como o `rgba` e o `linear-gradient`.

Primeiro, ao `rgba`. A função permite definir a opacidade de uma cor, permitindo criar efeitos de transparências utilizando apenas a cor de fundo ou as bordas de um elemento. Um ponto crucial é que a função não recebe um valor hexadecimal, como `#000000`, e sim utilizando os 3 decimais que representam a cor, no caso do preto são `0,0,0`. O quarto argumento da função é a opacidade deseja, de `0.0` a `1.0`.

Vamos a ação. Com o HTML a seguir, nós temos uma foto e uma legenda, utilizando novas tags como o `figure` e o `figcaption` para agrupar o `img` com a sua legenda.

```
<figure>
  <img src="/images/sf.jpg" alt="A Golden Gate.">
  <figcaption>
    San Francisco, Califórnia
    <small>
      Por Salim Virji (http://www.flickr.com/photos/salim/402618628/)
    </small>
  </figcaption>
</figure>
```

Nosso primeiro passo com o CSS será posicionar a mensagem por cima da foto, alguns pixels de distância do final dela.

```
figure {
  position: relative;
}

img {
  display: block;
}

figcaption {
  bottom: 5px;
  margin: 0 5px;
  padding: 5px;
  position: absolute;
  width: 300px;
}
```

Figura 5.12: Posicionamos a legenda sobre a foto, mas não é fácil ler o que está escrito.

Desta forma não temos contraste nenhum para ler o texto da legenda. Hora de aplicar um fundo preto e trocar a fonte da legenda para branco, melhorando sua leitura.

```
figcaption {
  background-color: #000;
```

```
  background-color: rgba(0,0,0,0.5);
  color: #FFF;
}
```

Figura 5.13: Não ficou melhor para ler a legenda da foto?

Com o `0.5` de opacidade do fundo preto conseguimos ler a legenda normalmente e ao mesmo tempo identificar as partes da foto que ficaram por trás dela. Caso o navegador utilizado não tenha suporte a `rgba`, será usado a definição de um fundo preto sólido, utilizando a clássica definição da cor em hexadecimal.

Combine transparência e bordas

Uma outra aplicação interessante para o `rgba` é poder combinar tons de preto e branco para escurecer ou clarear outras cores, como criando divisórias entre seções ou elementos de um menu. A praticidade desta abordagem é não se preocupar com a cor de fundo por trás da borda ou fundo com transparência, permitindo utilizar o mesmo CSS em contextos e lugares diferentes. Talvez uma forma melhor de entender isto é colocando a mão na massa.

Vamos montar uma lista, e alinhar todos os itens horizontalmente.

```
<ul>
  <li>Um</li>
  <li>Dois</li>
  <li>Três</li>
```

```
  <li>Quatro</li>
</ul>
```

Cada item da lista ocupará 50px, com o seu texto centralizado:

```
ul {
  height: 35px;
  list-style:none;
  padding: 0;
  width: 210px;
}

li {
  float: left;
  line-height: 35px;
  font-size: 0.9em;
  width: 50px;
  text-align: center;
}
```

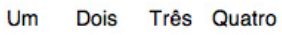


Figura 5.14: Os itens do menu devidamente posicionados.

Simples, certo? Hora de adicionar cores. Iremos mudar a cor do texto para branco e aplicar um fundo cinza na lista.

```
ul {
  background-color: #CCC;
  color: white;
}
```


Figura 5.15: O menu com um fundo cinza.

Agora que a lista possui uma cor sólida de fundo, podemos aplicar as bordas com transparência. Iremos aplicar uma borda preta com apenas 30% de opacidade na lista, e uma sequência de bordas brancas e pretas entre os elementos.

```
ul {
  border: 1px solid rgba(0,0,0,0.3);
}
li {
  border-left: 1px solid rgba(255,255,255, 0.3);
  border-right: 1px solid rgba(0,0,0, 0.3);
}

li:first-child {
  border-left: none;
}
li:last-child  {
  border-right: none;
}
```

Figura 5.16: As bordas com opacidade nos itens do menu.

Os seletores de `:first-child` e `:last-child` são utilizados para remover a borda esquerda do primeiro item da lista e a borda direita do último elemento, evitando que elas entrem em contato com a borda do `ul`. É possível notar que a borda preta pode ser considerada como uma versão mais escura do fundo cinza, enquanto a borda branca se torna um cinza mais claro, de forma bastante sutil. Se as bordas fossem sólidas, utilizando `#000000` e `#FFFFFF`, o resultado seria bastante diferente e nada interessante.

Figura 5.17: As bordas com cores sólidas - não é a mesma coisa.

Você pode agora alterar a cor de fundo da lista, utilizando desde cores primárias como azul ou vermelho ou tons específicos de verde ou laranja. O efeito que vimos no fundo cinza é replicável em diversos outros cenários e fundos. Faça alguns testes você mesmo e confira!

Figura 5.18: Variações do menu trocando apenas a cor de fundo.

Utilizando o linear-gradient

Talvez uma das partes do CSS 3 que foi mais esperado por diversos desenvolvedores, o `linear-gradient` é uma função capaz de criar imagens com gradientes a partir de uma lista de cores e posições. Extremamente útil para substituir imagens de fundo, os gradientes criados via CSS conseguem se adequar ao tamanho dos seus elementos e podem ser compostos de diversas cores em diversas posições, o que facilita traduzir um gradiente criado em um editor, como o Adobe Photoshop, para o seu código CSS.

A sintaxe da função é bastante direta. Primeiro se informa a direção em que o gradiente será projetado, aceitando palavras chave como `top` ou `left` (para gradientes na vertical ou horizontal) ou combinações de sentidos vertical e horizontal, como `top right`, para criar gradientes na diagonal. Caso você precise de mais controle sobre a direção dos seus gradientes, você pode informar um valor em graus (`deg`) também.

Além da direção, é necessário informar pelo menos duas cores, ou os chamados `color-stops`, um par de cor e posição (em `px` ou porcentagem), indicando em qual posição a cor deverá terminar ou começar. Toda a transição de uma cor para outra fica a cargo do navegador.

Apesar da flexibilidade da função, você não precisa utilizar muitas opções ou cores para chegar a gradientes tradicionais, como os utilizados em botões ou barras

de ações ou títulos.

```
div {
  font-weight: bold;
  height: 50px;
  line-height: 25px;
  margin-bottom: 10px;
  padding: 5px;
  text-align: center;
  width: 220px;
}

.blue {
  background-image: linear-gradient(top, #4377FA, #0537B7);
}

.reverse-blue {
  background-image: linear-gradient(top, #0537B7, #4377FA);
}

.omg-pink {
  background-image: linear-gradient(top right, #FC0050, #FF79A3);
}

.stops {
  background-image: linear-gradient(top, #F5B951 48%, #F2A31C 56%);
}
```

```
<div class='blue'>
  Um gradiente azul clássico, utilizando 2 tons similares.
</div>

<div class='reverse-blue'>
  Uma versão inversa do gradiente azul.
</div>

<div class='omg-pink'>
  Combinando "top" e "right" para a direção.
</div>

<div class='stops'>
  Utilizando posições específicas para as cores.
```

```
</div>
```

Figura 5.19: Os exemplos de 'linear-gradient'.

O caso clássico da função são os gradientes verticais (geralmente utilizando `top` como direção), partindo de um tom mais claro de uma cor para um tom mais escuro.

Uma regra simples para não se dar mal com gradientes é manter os tons de cores bastante similares, criando uma leve ilusão de uma superfície arredondada e sob o efeito de alguma fonte de luz. Para isto, tenha em mãos alguma ferramenta de manipulação de cores para montar as combinações necessárias. Como no exemplo anterior, o primeiro gradiente é composto de um tom específico de azul e uma versão mais escura dele mesmo.

Uma alternativa para garantir que os elementos com gradiente não fiquem sem fundo algum em navegadores que não possuam suporte a função é fixar a cor de fundo do elemento para a cor predominante, desta forma o elemento terá um fundo sólido caso o gradiente não seja exibido.

```
.blue {
  background-color: #4377FA; /* A cor superior do gradiente como fundo */
  background-image: linear-gradient(top, #4377FA, #0537B7);
}
```

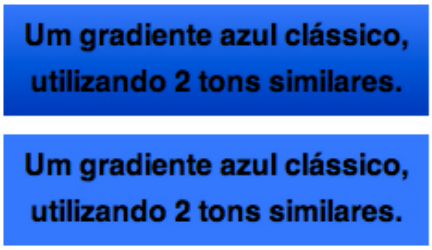


Figura 5.20: A diferença do gradiente e da cor de fundo. (exibida em navegadores sem suporte)

Outra solução seria definir um `background-image` antes da declaração do gradiente, contendo o caminho para uma imagem a ser utilizada caso a função não seja executada.

Apesar da vantagem de que a imagem só será acessada pelo navegador neste cenário de falha, o esforço em manter as imagens junto dos gradientes não me parece uma boa ideia, ainda mais com o aumento de uso de navegadores que suportam estes gradientes, estas imagens se tornarão obsoletas logo.

5.6 Trabalhe com sombras e crie menus elegantes

A propriedade `box-shadow` permite adicionar múltiplas sombras a elementos das suas páginas. Existem dois tipos de sombras que podemos trabalhar com CSS: sombras externas e internas. Sombras externas, que são o comportamento padrão do `box-shadow`, são exibidas atrás do elemento, similar ao efeito que é chamado de *drop shadow* em editores de imagens. As sombras internas, que são exibidas dentro das bordas dos elementos e acima do seu fundo, se compara aos efeitos de *inner shadow*.

Para se criar uma sombra você precisa, primeiro, as coordenadas em que a sombra ficará posicionado. Em seguida você pode definir dois valores opcionais: a densidade da sombra (geralmente chamado de *blur*), e um tamanho adicional para contrair ou expandir o efeito da sombra. Apesar de opcional, sombras sem o seu *blur* não passam de mais uma borda ao redor do elemento, então ele sempre está presente nas suas sombras. Após todos os valores numéricos você deve informar uma cor - que pode ser um valor hexadecimal ou uma chamada da função `rgba`.

Os efeitos de sombra são úteis para realçar a sobreposição de um elemento sobre outro, o que cai como uma luva na caixa de ajuda que criamos no capítulo 4.5. Basta

adicionar o uso do `box-shadow` na classe `.help`.

```
.help {
  box-shadow: 3px 3px 3px #AAA;
}
```

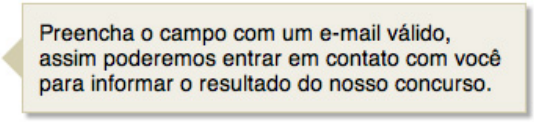


Figura 5.21: A aplicação do 'box-shadow' na mensagem de ajuda.

Valores positivos movem a sombra para baixo ou para o lado esquerdo do elemento, enquanto valores negativos tomam o caminho contrário. O ângulo da sombra é crucial para criar o apelo visual que você quer, e um uso incorreto dos valores pode trazer visuais menos interessantes.

```
.help {
  box-shadow: -3px -3px 3px #AAA;
}
```

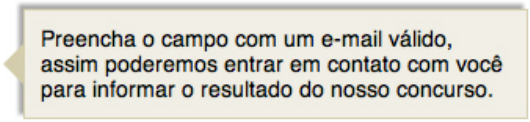


Figura 5.22: A sombra invertida não combina com o elemento como a anterior.

Sombras internas podem ser utilizadas para criar um efeito de profundidade dentro de um elemento. Com isto você pode criar a ilusão de algo "pressionado", como um botão de um formulário ou o item de menu que está selecionado, ou realçar a indicação que um elemento é um campo de texto, por exemplo.

```
Faça a sua pesquisa aqui: <input type="text">
```

```
input {
  border: 1px solid #999;
  box-shadow: 2px 2px 2px rgba(0,0,0, 0.25) inset;
  padding: 2px 5px;
}
```

Figura 5.23: A sombra interna da caixa de texto.

O efeito é bastante sutil, mas é uma opção elegante para os estilos dos seus formulários.

Outra combinação interessante de campos de formulários e `box-shadow` é criar uma sombra posicionada com `0 0`, criando um brilho ao seu redor, quando o usuário estiver com o cursor dentro do campo de texto. Alterar a cor da borda do campo para a mesma da sombra ajuda a consolidar este efeito.

```
input:focus {
  border-color: #35861B;
  box-shadow: 0 0 5px #35861B;
  outline: none;
}
```

Figura 5.24: O brilho da caixa de texto, mas a sombra interna não está presente.

Mas assim, a sombra que criamos anteriormente para o campo deixa de ser exibida, já que sobrescrevemos a propriedade. A solução é repetir a sombra existente, separando elas por vírgula. Assim é possível manter a consistência dos estados do elemento.

```
input:focus {
  border-color: #35861B;
```

```
  box-shadow: 0 0 5px #35861B, 2px 2px 2px rgba(0,0,0,0.25) inset;
  outline: none;
}
```

Figura 5.25: As duas sombras da caixa de texto ao receber foco.

Se utilizando a posição `0 0` criamos um efeito de brilho em sombras externas, fazendo o mesmo com sombras internas é possível alcançar um efeito de profundidade mais pesado, como se a superfície da sua página não fosse plana. Vamos criar um menu de navegação para fazer este teste. Começando com o HTML de uma lista simples.

```
<ul class='nav'>
  <li><a href="#">Home</a></li>
  <li><a href="#">Contato</a></li>
  <li><a href="#">Sobre</a></li>
</ul>
```

Vamos alinhar os elementos horizontalmente utilizando `float`, além de definir um tamanho padrão para cada link do menu e uma cor de fundo.

```
.nav {
  background-color: #B9522D;
  height: 30px;
  list-style:none;
  padding: 0;
  width: 210px;
}

.nav li {
  float: left;
  font-weight: bold;
  height: 30px;
  line-height: 30px;
  text-align: center;
  width: 70px;
```

```
}

.nav a {
  color: white;
}
```

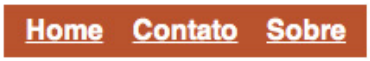


Figura 5.26: O menu horizontal que criamos.

Digamos que você está na página principal, e o menu deve ter uma indicação visual que o link *Home* é o da página atual. Além de mudanças no estilo do texto e no fundo do elemento, uma sombra interna pode ser utilizada para criar esta impressão. Primeiro, precisamos de uma classe para indicar qual dos links é o atual:

```
<ul class='nav'>
  <li class='current'><a href="#">Home</a></li>
  <li><a href="#">Contato</a></li>
  <li><a href="#">Sobre</a></li>
</ul>
```

Agora, utilizamos a mesma técnica do brilho ao redor da caixa de texto, só que utilizando o `inset`.

```
.nav .current {
  background-color: #A64A28;
  box-shadow: inset 0 0 10px rgba(0,0,0,0.5);
}
```

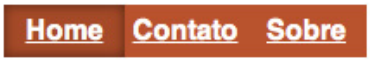


Figura 5.27: O link 'Home' não aparenta estar pressionado?

Além da mudança para um fundo mais escuro, a sombra contribui para diferenciar o link *Home* dos demais.

Cuidado com sombras pesadas

A combinação de `box-shadow` e `rgba` pode trazer resultados interessantes para as suas interfaces, mas cuidado ao abusar de opacidade e *blur* juntos. Ross Allen, da equipe do Airbnb, percebeu que ao navegar pelo site da empresa utilizando um Chromebook, um notebook criado pelo Google que era disponível para clientes da Virgin America durante os seus voos, a rolagem da página era drasticamente mais devagar do que em seu computador pessoal. Utilizando ferramentas de desenvolvimento para identificar o que estaria causando esta lentidão, ele chegou a conclusão que o uso do `box-shadow` com `10px` de *blur* em algumas partes do site era relativamente pesado para o navegador exibir e reconstruir conforme a rolagem era feita, e isto era perceptível em um computador de poder de processamento inferior, como o Chromebook.

Reduzindo a densidade e contando com a posição das sombras, e não o seu tamanho, para criar os efeitos desejados, o problema de performance foi resolvido e a experiência de uso do Airbnb em outros dispositivos foi melhorada com apenas mudanças no uso de `box-shadow`.[1]

Trabalhando com sombras em textos

Não tão poderosa quanto o `box-shadow`, a propriedade `text-shadow` permite aplicar sombras ao texto da sua página, e não aos elementos. O uso das propriedades é bastante similar: você pode posicionar a sombra a vontade e combinar diversos efeitos com diversas sombras, mas o `text-shadow` não possui a quarta opção de tamanho do `box-shadow` e o termo `inset` não é suportado. Então, se você já se deu bem com uma destas propriedades, utilizar a outra será como um passeio de bicicleta.

```
<h1>HTML & CSS</h1>

h1 {
  background-color:#1D9AC0;
  color: #FFF;
  padding: 10px;
  text-shadow: 2px 4px 2px rgba(0,0,0,0.3);
}
```

HTML & CSS

Figura 5.28: Um exemplo simples de 'text-shadow'.

O uso do `text-shadow` pode seguir a mesma linha do `box-shadow`: realçar sobreposições e adicionar efeitos sutis para melhorar a leitura de certos textos em fundos diferentes.

Imagine uma loja virtual, onde cada compra feita pode estar em um de três estados: *Pagamento recusado*, *Produto enviado* ou *Em aprovação*. Na exibição dos dados de uma compra, podemos exibir o seu estado com uma cor de fundo relacionada a ele, utilizando o seguinte código

```
<span class='status failed'>Pagamento recusado</span>
<span class='status shipped'>Produto enviado</span>
<span class='status processing'>Em aprovação</span>
```

```
.status {
  display: inline-block;
  font-weight: bold;
  font-weight: bold;
  padding: 7px 10px;
}

.failed {
  background-color: #BD2C00;
}

.shipped {
  background-color: #6CC644;
}

.processing {
  background-color: #FC9800;
}
```

Figura 5.29: As indicações de estado - não acha que este texto preto está ruim?

Para dar mais atenção ao texto (e não ao fundo), podemos trocar a cor do texto para branco e combinar com uma sombra, trazendo atenção e preservando a leitura dos textos ao mesmo tempo. Para este caso a sombra com `rgba` é uma ótima pedida.

```
.status {
  color: white;
  display: inline-block;
  font-weight: bold;
  font-weight: bold;
  padding: 7px 10px;
  text-shadow: 0 1px 2px rgba(0, 0, 0, 0.7);
}
```

Figura 5.30: Os textos de estado utilizando 'text-shadow'.

5.7 Combinando tudo

Passamos por diversas propriedades muito interessantes, que utilizadas sozinhas já produzem resultados visuais sensacionais. Mas ao utilizar algumas (ou todas) delas você consegue compor seções e componentes visualmente elegantes, e combinando algumas das técnicas apresentadas neste capítulo junto de novos usos das propriedades do CSS 3 conseguimos chegar ao seguinte resultado: uma seção para exibir o livro *Ruby on Rails: coloque sua aplicação web nos trilhos*, disponibilizando opções de compra e a situação do livro em estoque. Este será um ótimo exercício para combinar algumas das coisas que aprendemos.

Figura 5.31: O resultado final da seção que iremos criar.

Vamos começar com parte do HTML, junto do nome do livro e do autor. O CSS para estes texto será bem simples, similar o que já vimos anteriormente.

```
<article class="book">
  <header>
    <h1>Ruby on Rails: coloque sua aplicação web nos trilhos</h1>
    <h2>Vinicius Baggio</h2>
  </header>
</article>
```

```
h1 {
  color: #BD2C00;
  font-size: 1.2em;
  margin: 0;
}

h2 {
  font-size: 1em;
  margin: 5px 0;
}
```

Figura 5.32: Os estilos para o título e o autor do livro.

Em seguida, o estado de *Disponível* pode ser baseado nos exercícios de `text-shadow` que fizemos, com a adição de um `border-radius` para arredondar os cantos do elemento. Aplicamos um fundo verde, a cor branca e uma leve sombra no texto, junto de alguns ajustes de espaçamento e tamanhos.

```
<article class="book">
  <header>
    <h1>Ruby on Rails: coloque sua aplicação web nos trilhos</h1>
    <h2>Vinicius Baggio</h2>
    <span class='available'>Disponível</span>
  </header>
</article>
```

```
.available {
  background-color: #6CC644;
  border-radius: 3px;
  color: white;
  font-size: 0.8em;
  font-weight: bold;
  padding: 3px 7px;
  text-shadow: 0 1px 2px rgba(0, 0, 0, 0.7);
}
```

Figura 5.33: A indicação da disponibilidade do livro.

Em seguida, criaremos o menu de compras, que combina diversos truques. O HTML utilizado será uma simples lista não ordenada:

```
<ul>
  <li><a href="#" class='icon paper'>Comprar a versão física</a></li>
  <li><a href="#" class='icon digital'>Comprar a versão digital</a></li>
  <li>
    <a href="#" class='icon package'>Comprar o pacote completo!</a>
```

```
  </li>
</ul>
```

Aplicamos um `linear-gradient` azul para o fundo do menu, junto de ajustes a espaçamento e posição:

```
ul {
  background-color: #56B4EF;
  background-image: linear-gradient(top, #56B4EF,#148EDA);
  float: left;
  list-style: none;
  margin: 10px 10px 0 0;
  padding: 0;
  width: 230px;
}
```

E para os ícones, iremos reaproveitar a fonte **Iconic** e utilizar alguns de seus ícones no menu.

```
@font-face {
  font-family: 'IconicFill';
  src: url('iconic_fill.woff') format('woff');
  font-weight: normal;
  font-style: normal;
}

.icon::before {
  font-family: 'IconicFill';
  margin: 0px 5px;
}

.paper::before {
  content: "\e00b";
}

.digital::before {
  content: "\e044";
}

.package::before {
  content: "\e022";
}
```

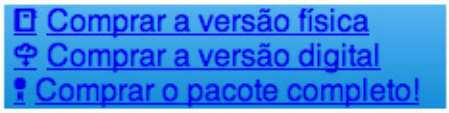


Figura 5.34: O 'linear-gradient' e os ícones do menu.

Para apresentar os links de uma forma mais interessante, precisamos mudar a sua cor e espaçar melhor os textos dos 3 itens - outro lugar onde o `text-shadow` cai como uma luva.

```
.icon {
  color: #FFF;
  display: block;
  font-size: 0.9em;
  font-weight: bold;
  padding: 5px;
  text-decoration: none;
  text-shadow: 0 1px 0 rgba(0, 0, 0, 0.9);
}
```

O próximo passo é criar as bordas entre os links, aplicando bordas com `rgba` utilizando preto e branco. Importante remover a borda superior do primeiro link e a inferior do último, aproveitando as classes utilizadas para exibir os ícones.

```
.icon {
  border-bottom: 1px solid rgba(0,0,0, 0.3);
  border-top: 1px solid rgba(255, 255, 255, 0.3);
}

.icon.paper {
  border-top: none;
}

.icon.package {
  border-bottom: none;
}
```

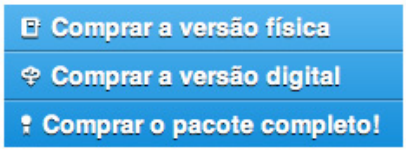


Figura 5.35: Uma versão melhorada do menu utilizando 'text-shadow' e bordas com 'rgba'.

Podemos utilizar a mesma cor em `rgba` usada no `border-top` dos links para contornar o menu inteiro, além de adicionar um `border-radius` ao seu estilo.

```
ul {
  border-radius: 5px;
  border: 1px solid rgba(0,0,0, 0.3);
}
```

O último detalhe do menu - uma indicação de `:hover` para os links. Podemos aplicar um fundo preto com transparência no link, que irá aplicar um efeito em que o fundo azul se tornará mais escuro.

```
.icon:hover {
  background-color: rgba(0, 0, 0, 0.1);
}
```

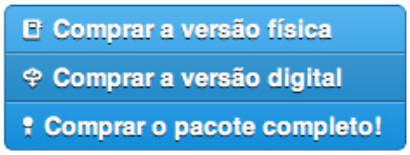


Figura 5.36: A versão final do menu com bordas arredondadas.

E o nosso menu está pronto, combinando bons usos de `rgba`, %linear-gradient, text-shadow `e` border-radius

A descrição do livro será apenas um parágrafo, sem estilo próprio algum, que deverá ir logo após o HTML do menu, que devido a sua definição de `float` irá alinhar os 2 elementos horizontalmente.

```
<p>
  Uma ótima leitura para iniciantes afim de se aventurar no ecossistema
  de Ruby.
</p>
```

O toque final será o acabamento da seção inteira, identificada pela classe `book`. Vamos seguir um estilo similar ao do menu - bordas arredondadas, fundo com gradiente e uma borda um pouco mais escura que o fundo, além de fixar a largura da seção e espaçar um pouco o seu conteúdo.

```
.book {
  background-color: #FAFAFA;
  background-image: linear-gradient(#FAFAFA,#EEE);
  border-radius: 5px;
  border: 1px solid #CCC;
  padding: 10px;
  width: 600px;
}
```

Como o nosso menu está utilizando `float`, precisamos corrigir a altura da seção para que todo os elementos fiquem devidamente contidos no `article`. Podemos repetir o uso de um elemento vazio com a classe `clear` ao final do conteúdo.

```
<div class='clear'></div>
```

```
.clear {
  clear: both;
}
```

Figura 5.37: A versão final da seção sobre o livro.

Estes são alguns do exemplos do que é possível criar juntando gradientes, sombras e similares. Combinações similares podem ser úteis para criar botões, cabeçalhos e rodapés, barras laterais e diversos outros componentes de interface que você precise nos seus projetos.

Capítulo 6

Tomando controle da estrutura visual

Uma das maiores complicações em projetos grandes e complexos é estruturar e posicionar corretamente todos os elementos existentes em uma página. Encaixar menus, barras laterais, imagens e textos nos seus devidos lugares pode resultar em um HTML complexo e um CSS muito extenso se não for feito corretamente. É importante compreender a necessidade de cada grupo de elementos, e o contexto do elemento *pai* que os envolvem. Existem soluções similares para certos casos, mas o impacto de cada opção deve ser levado em consideração.

Temos 3 propriedades bastante importantes para se controlar o fluxo e a posição dos seus elementos: `display`, `position` e `float`. Utilizamos elas em outros capítulos, mas não chegamos no detalhe seus motivos e o que elas estão fazendo com os nossos elementos. Hora de olhar cada uma delas e entender melhor o seus papéis na estruturação dos seus elementos.

6.1 A propriedade 'display'

A propriedade `display` determina como um elemento será exibido pelo navegador. Elementos de texto, como `a`, `span` e `small`, possuem um valor `inline`, que permite que elementos seguintes se mantenham na mesma linha que o atual, e não são afetados por propriedades como `height` e `width`. *Containers* como `div`, `p`, `section` e similares são tratados como blocos - com o valor padrão de `block` - quebrando o fluxo dos elementos em uma nova linha e ocupando a largura máxima que lhe é permitida. Elementos de uma tabela, como `td%`, `thead` e `tr` possuem seus próprios valores de display, como table-cell e table-rowmas se comportam como elementos de bloco. Uma forma de se testar o comportamento dos elementos é simplesmente adicionar uma cor de fundo para cada um deles e observar o espaço preenchido.

```
p {
  background-color: LimeGreen;
}

span {
  background-color: LightBlue;
}

<span>inline</span>
<span>inline</span>
<p>block</p>
```

Figura 6.1: A diferença de elementos 'inline' e 'block'.

Além dos valores padrões, existem duas outras possibilidades bastante úteis para a propriedade `display`: o `none` e o `inline-block`.

O `none` faz com que o elemento e todo o seu conteúdo sumam da página, como se não estivessem presentes no HTML, enquanto o `inline-block` mistura o comportamento dos valores `block` e `inline` - os elementos não geram quebras, mas possuem as dimensões equivalentes ao seu comportamento da propriedade `block`. Ele seria

uma solução perfeita para muitos problemas se não fosse um detalhe: o espaço entre os elementos no HTML impacta na exibição deles. Considerando o CSS a seguir:

```
span {
  display: inline-block;
}
```

E o HTML com o estilo aplicado:

```
<span>inline-block</span>
<span>inline-block</span>
<span>inline-block</span>
<br>
<span>inline-block</span><span>inline-block</span>
```

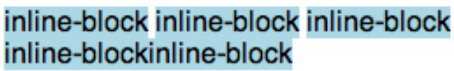


Figura 6.2: O comportamento do 'inline-block' com e sem quebra de linhas entre os elementos. Complicado, não?.

Caso exista uma quebra de linha ou um espaço entre os elementos no HTML, a renderização deles com `inline-block` irá possuir um espaço em branco - o que muitas vezes não é o desejado. Chris Coyier, do CSS - Tricks propõe diversas soluções [2] para este problema, mas nenhuma me parece válida o suficiente para ser utilizada com frequência. Assim como sugere o final do post, sou adepto de optar pelo uso de `float` nestes casos.

6.2 Flutue elementos

A propriedade `float` é uma das soluções mais versáteis quando se trata de alinhar e posicionar elementos, que teve origem na diagramação de textos no mundo da mídia impressa onde é bastante comum ver imagens *flutuando* ao lado de pedaços de texto, como em uma matéria de uma revista. Na web essa técnica evoluiu e soluciona diversos casos de posicionamento de elementos.

O caso mais simples é para alinhar elementos horizontalmente, flutuando parte deles e deixando que o resto acompanhe pela lateral, dessa forma, com o seguinte HTML:

```
<h2>#1</h2>
<h4>500 pontos</h4>
<p>Prêmio para o campeão: 1 MacBook Pro 15".</p>
```

E o CSS:

```
h2 {
  float: left;
  margin: 0 20px 20px 0;
}

h4 {
  margin-top: 5px;
}
```

Teremos:

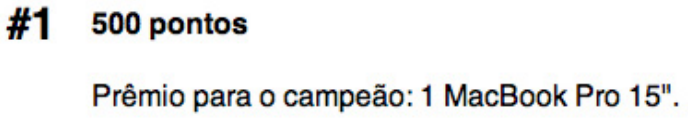


Figura 6.3: Alinhando elementos com 'float: left'.

Elementos com a propriedade `float` são tratados como elementos de bloco (lembra do `display: block`?), mas sem ocupar os 100% de largura tradicionais desses elementos. E da mesma forma que ele pode flutuar para esquerda, ele pode ir para a direita.

```
h2 {
  float: right;
}
```

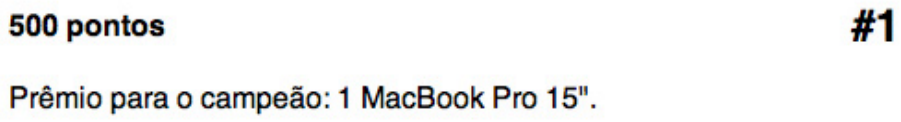


Figura 6.4: Você também pode jogar os seus elementos para a direita com 'float: right'.

Para garantir que um elemento não vá seguir a orientação de um antecessor que esteja com `float`, você precisa utilizar a propriedade `clear`. Você pode informar o mesmo sentido que os seus elementos estão flutuando - `left` ou `right` - ou utilizar `both`, que *quebra* o fluxo em ambos os sentidos.

```
p {
  clear: both;
}
```

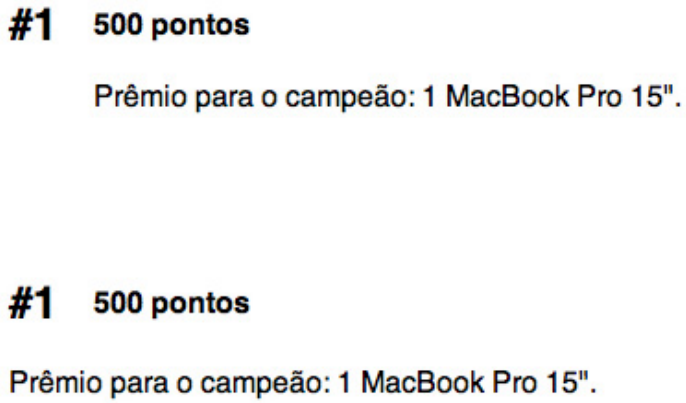


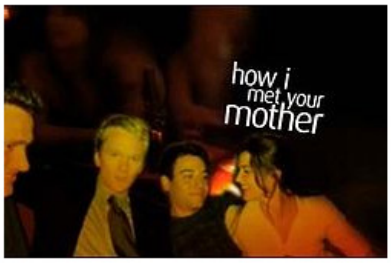
Figura 6.5: O efeito da propriedade 'clear' - o parágrafo não está mais alinhado com o título.

Outra aplicação tradicional da propriedade `float` é para alinhar imagens e textos, similar a uma matéria de jornal, onde a imagem fica ao lado do texto que a acompanha.

```
<h3>How I Met Your Mother</h3>
<img src="himym.jpg">
<p>
  A série gira em torno da vida de Ted Mosby e dos seus amigos,
  que é narrada pelo próprio aos seus filhos, 25 anos mais tarde.
  Bob Saget, como Ted Mosby do futuro, conta então aos filhos as
  histórias e peripécias que o levaram a conhecer a mãe deles.
  As outras personagens principais são Marshall Eriksen,
  Robin Scherbatsky, Lily Aldrin e Barney Stinson.
</p>

<p>
  Em 2005, aos 27 anos, o jovem Ted Mosby (Josh Radnor), após o
  seu melhor amigo, Marshall Eriksen (Jason Segel), ficar noivo,
  decide finalmente ir em busca da sua cara-metade. Com gestos
  românticos questionáveis, Ted conhece Robin Scherbatsky
  (Cobie Smulders), no bar que costumavam frequentar,
  Maclaren's Pub.
</p>
```

How I Met Your Mother



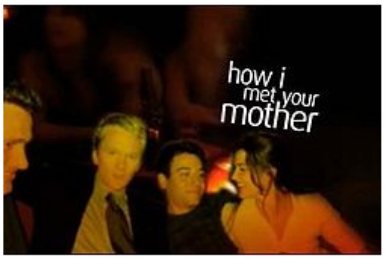
A série gira em torno da vida de Ted Mosby e dos seus amigos, que é narrada pelo próprio aos seus filhos, 25 anos mais tarde. Bob Saget, como Ted Mosby do futuro, conta então aos filhos as histórias e peripécias que o levaram a conhecer a mãe deles. As outras personagens principais são Marshall Eriksen, Robin Scherbatsky, Lily Aldrin e Barney Stinson.

Em 2005, aos 27 anos, o jovem Ted Mosby (Josh Radnor), após o seu melhor amigo, Marshall Eriksen (Jason Segel), ficar noivo, decide finalmente ir em busca da sua cara-metade. Com gestos românticos questionáveis, Ted conhece Robin Scherbatsky (Cobie Smulders), no bar que costumavam frequentar, Maclaren's Pub.

Com um simples uso de `float` e uma margem (para o texto não ficar *colado* com a imagem) o texto se posiciona ao lado da imagem, fluindo perfeitamente caso o texto seja muito grande, exatamente como o nosso exemplo.

```
img {
  float: left;
  margin-right: 10px;
}
```

How I Met Your Mother



A série gira em torno da vida de Ted Mosby e dos seus amigos, que é narrada pelo próprio aos seus filhos, 25 anos mais tarde. Bob Saget, como Ted Mosby do futuro, conta então aos filhos as histórias e peripécias que o levaram a conhecer a mãe deles. As outras personagens principais são Marshall Eriksen, Robin Scherbatsky, Lily Aldrin e Barney Stinson.

Em 2005, aos 27 anos, o jovem Ted Mosby (Josh Radnor), após o seu melhor amigo, Marshall Eriksen (Jason Segel), ficar noivo, decide finalmente ir em busca da sua cara-metade. Com gestos românticos questionáveis, Ted conhece Robin Scherbatsky (Cobie Smulders), no bar que costumavam frequentar, Maclaren's Pub.

Figura 6.6: Por padrão, o texto ficará abaixo da imagem, ocupando bastante espaço vertical.

E da mesma forma que organizamos elementos em exemplo menores, podemos organizar os principais elementos da página utilizando `float`. Hora de por a mão na massa e ver isso em ação:

```
<section class='posts'>
  <article>
    <h1><a href='#'>Título do post #1</a></h1>
    <span class='timestamp'>3 semanas atrás</span>

    <p>
      Uma breve descrição sobre o assunto abordado no post número 1...
    </p>
  </article>
```

```
  <article>
    <h1><a href='#'>Título do post #2</a></h1>
    <span class='timestamp'>3 semanas atrás</span>

    <p>
      Uma breve descrição sobre o assunto abordado no post número 2...
    </p>
  </article>
</section>
```

Vamos dedicar uma área de 600px para esta listagem, posicionando a informação da data do post a direita do título, devidamente alinhado.

```
.posts {
  width: 600px;
}

h1 {
  float: left;
  margin: 0 0 20px;
}

.timestamp {
  background-color: #000;
  color: #FFF;
  float: right;
  font-size: 0.9em;
  margin-top: 10px;
  padding: 5px 10px;
}
```

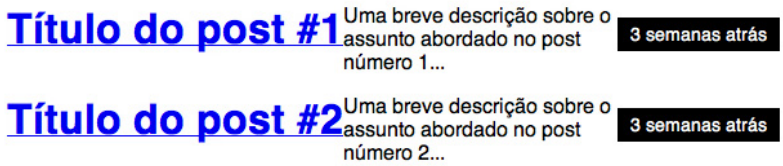


Figura 6.7: Os primeiros estilos da listagem de posts, organizando os elementos com 'float'.

O título e a informação da data estão nos seus devidos lugares, mas a descrição não deve ficar entre eles, e sim abaixo do título. Hora de aplicar a propriedade `clear` então.

```
p {
  clear: both;
}
```

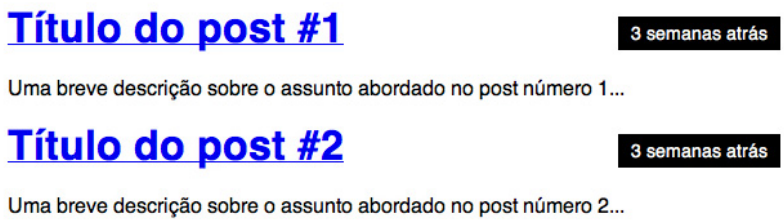


Figura 6.8: Corrigindo a posição dos parágrafos.

Vamos incrementar o nosso cenário. Hora de posicionar uma relação de links ao lado direito da seção de posts. Como fizemos na seção 2.6, a forma mais simples de tratar este o posicionamento de elementos é flutuando cada um - o conteúdo principal e a barra lateral - para lados diferentes. Então, temos o HTML com a lista de posts:

```
<aside>
  <h3>Veja outros posts</h3>
  <ol>
    <li><a href="#">Post #3</a></li>
    <li><a href="#">Post #4</a></li>
    <li><a href="#">Post #5</a></li>
    <li><a href="#">Post #6</a></li>
  </ol>
</aside>
```

E o CSS que coloca a lista com outros posts, que esterá dentro do `aside`, no lado direito, enquanto que a listagem dos posts que havíamos feito antes será exibida do lado esquerdo.

```
.posts {
  float: left;
```

```
  width: 600px;
}

aside {
  background-color: #FFFBE4;
  border: 1px solid #C9BC8F;
  float: right;
  padding: 10px;
  width: 200px;
}

aside h3 {
  margin: 0;
}
```

Figura 6.9: O visual da seção lateral de links.

Com cada seção flutuando para um lado, vamos colocar ambos elementos em uma nova `div`, para definir uma largura limite para o conteúdo e centralizá-lo.

```
<div class='page'>
  <section class='posts'>
    <!-- ... os 2 posts principais ... -->
  </section>
  <aside>
    <!-- ... e a lista de links para outros posts ... -->
  </aside>
</div>

.page {
  padding: 10px;
  margin: 0 auto;
  width: 940px;
}
```

Simples, certo? agora vamos aplicar uma cor de fundo a nova div, que criamos, que deverá preencher todo o fundo do conteúdo da página.

```
.page {
  background-color: #E8E8E8;
}
```

Figura 6.10: A cor de fundo não foi aplicada corretamente na página...

Mas esse não é o resultado que queremos! Um elemento com `float` não contribui com a altura do seu *container*, então elementos como o nosso `.page` aparentam ter apenas alguns `px` de altura, devido ao seu `padding`, e o seu conteúdo com `float` aparenta ter vazado do elemento. A solução para este comportamento estranho é ter um elemento com `clear` logo após os elementos com `float` que causaram este efeito.

Em exemplos anteriores isto foi feito utilizando um elemento vazio, mas esta solução acaba sujando o nosso HTML com elementos vazios para ajustar casos como esse. A seguir iremos conhecer uma solução mais adequada para este problema.

6.3 O clearfix, uma classe obrigatória em seus projetos

Este problema clássico de uso de `float` pode ser solucionado com um elemento vazio após o conteúdo que flutua, mas não precisamos adicionar tags vazias ao nosso código, pois estes elementos adicionais já estão presentes na página - os pseudo elementos, que foram apresentados na seção 4.3.

O chamado `clearfix hack` consiste em adicionar estilos específicos para os pseudo elementos de um *container* que tenha elementos com `float` e precise ter a sua altura adequada ao conteúdo.

O código CSS necessário é bem pequeno, mas bastante efetivo:

```
.clearfix::before,
.clearfix::after {
    content: "";
    display: table;
}

.clearfix::after {
    clear: both;
}

.clearfix {
    *zoom: 1;
}
```

Este uso específico do valor `table` para a propriedade `display` serve para ajustar bugs de margem entre o *container* e os elementos com `float` e o uso de `clear: both` no `::after` produz o mesmo resultado de se colocar um elemento vazio após o conteúdo.

A última parte - a propriedade `zoom`, é uma técnica específica para versões antigas do Internet Explorer (6 e 7, para ser exato) e pode ser ignorada caso você não precise trabalhar com estas versões deste navegador.

Sem precisar de novos elementos, podemos adicionar a classe `clearfix` na `div` do exemplo anterior, assim solucionamos o problema da sua cor de fundo.

```
<div class='page clearfix'>
  <section class='posts'>
    <!-- ... os 2 posts principais ... -->
```

```
  </section>
  <aside>
    <!-- ... e a lista de links para outros posts ... -->
  </aside>
</div>
```

Figura 6.11: Sucesso! problema de cor de fundo resolvido.

Eventualmente, alguns desenvolvedores acabam utilizando algum outro nome para a classe `clearfix`, como `cf`, reduzindo o nome para se escrever menos, ou `group`, para expressar melhor o contexto de uso. Independente do nome, sempre tenha ela nos seus projetos para tratar este problema clássico de uso da propriedade `float`.

6.4 Compreenda o uso de position

Outra forma de controlar o posicionamento dos seus elementos é combinar o uso da propriedade `position` e das propriedades de coordenadas `top`, `right`, `bottom` e `left`. De acordo com o valor do `position`, as coordenadas são aplicadas de uma forma diferente, o que muda toda a perspectiva de uso desse grupo de propriedades.

Dos quatro valores possíveis, o mais simples é o `static`. É o valor padrão para todos os elementos, e a sua posição não é afetada por nenhuma das propriedades de coordenadas, deixando que o navegador posicione o elemento no seu lugar de origem.

O valor `fixed` remove o elemento da sua posição original (reorganizando os elementos adjacentes, caso seja necessário) e fixa a sua posição na janela do navegador.

A posição do elemento é preservada conforme o usuário navega para cima e para baixo com a rolagem da página, útil para caixas de mensagem que devem sobrepor

o conteúdo ou elementos que devem acompanhar o fluxo do usuário pela página, como uma caixa de links de navegação.

Os dois outros valores possíveis são os que mais confundem os desenvolvedores: `relative` e `absolute`.

Com o valor `relative`, as coordenadas são aplicadas a partir do ponto original do elemento, assim:

```
<div class='box'>
  Teste
</div>

.box {
  background-color: PapayaWhip;
  height: 100px;
  width: 100px;
  position: relative;
  top: 30px;
  left: 5px;
}
```

Figura 6.12: A combinação do 'position: relative' e 'top' afetando a posição de um elemento.

Diferente do uso de `margin`, o elemento com `position: relative` não irá afetar a posição dos elementos presentes ao seu redor. Por exemplo, o uso de `top` não empurra os elementos seguintes para baixo, mas posiciona a `div` por cima deles, assim:

```
<div class='box'>
  Teste
</div>
<p>Um parágrafo de texto logo abaixo</p>
```

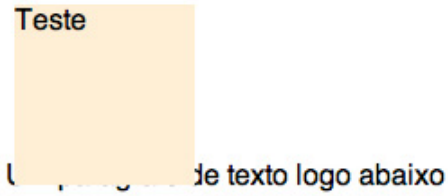


Figura 6.13: O elemento com position pode acabar por cima de outros elementos.

Já o valor `absolute` possui um comportamento mais complicado de se acostumar.

Elementos absolutamente posicionados são afetados pelas coordenadas das propriedades `top`, `bottom` e afins. No entanto, elas são aplicadas a partir do seu primeiro elemento pai que que não tenha `position: static` - então, elementos *absolutos* são *relativos* a um elemento pai. Caso nenhum elemento acima dele na hierarquia da sua página atenda a este critério, o `body` será utilizado como referência.

```
<h1>Posicionado com "absolute"</h1>

h1 {
  background-color: LightBlue;
  position: absolute;
  top: 20px;
  left: 30px;
}
```

Neste exemplo, o `h1` está sendo posicionado de acordo com o `body`. Mas se colocarmos ele dentro de outro elemento que esteja posicionado como `relative`, por exemplo, a posição do `h1` será afetada.

```
<header>
  <h1>Posicionado com "absolute"</h1>
</header>
```

```
header {
  margin-top: 30px;
  position: relative;
}
```

Figura 6.14: Ao utilizar o 'position: relative' no header, a posição do h1 muda.

Esta peculiaridade do uso de `absolute` pode apresentar muitos problemas, mas também adiciona muita flexibilidade ao uso deste valor.

Como os elementos com `absolute` e `fixed` são removidos do fluxo padrão da página, eles não afetam a altura dos elementos que supostamente os contêm, o que gera um problema similar ao uso de `float`, mas sem uma solução similar ao truque do `clearfix`.

6.5 Crie a sua própria janela modal

Janelas modais são um elementos recorrentes em diversos sites e são bons exemplos de aplicação de `position`.

Vamos começar com uma `div` e um parágrafo, que serão o conteúdo do nosso modal.

```
<div class='notice'>
  <h1>Atenção</h1>
  <p>Aqui você pode exibir alguma mensagem importante, que ela irá
    sobrepor todo o conteúdo do seu site.</p>
</div>
```

Vamos definir alguns estilos básicos para definir o formato da caixa que irá conter este conteúdo, para garantir que a `div` tenha uma altura e largura fixa, que irá facilitar seu posicionamento.

```
.notice {
  background-color: #FFF;
  border: 1px solid #000;
```

```
  height: 130px;
  padding: 10px;
  width: 380px;
}
```

Hora de posicionar o modal no centro da página, combinando a propriedade `position` com as coordenadas `top` e `left`.

```
.notice {
  position: fixed;
  top: 50%;
  left:50%;
}
```

Neste caso, é indicado utilizar o valor `fixed` e não o `absolute` - apesar da posição do elemento ser a mesma, a vantagem de se utilizar o `fixed` é que caso a sua página exiba a barra de rolagem vertical, o modal irá acompanhar a rolagem para cima e para baixo do usuário.

Com as porcentagens no `top` e `left`, o canto superior direito do modal está posicionado no centro da página, mas precisamos alinhar o elemento todo. Como fixamos a largura e altura do modal, fica simples corrigir isto utilizando margens negativas, finalizando o CSS para esta parte do modal.

```
.notice {
  background-color: #FFF;
  border: 1px solid #000;
  height: 130px;
  padding: 10px;
  width: 380px;
  position: fixed;
  top: 50%;
  left: 50%;
  margin: -75px 0 0 -200px;
}
```

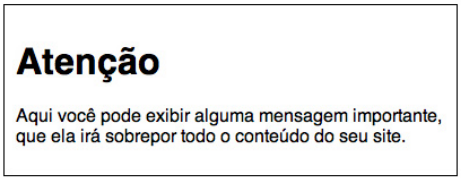


Figura 6.15: O conteúdo do nosso modal, devidamente centralizado na página.

A propriedade `margin` é aplicada de metade da altura do elemento (`150px`) no topo e metade da largura (`400px`) na margem esquerda (`margin-left`) do elemento, onde ambos os valores são negativos, para o elemento retroceder alguns `px` de volta para o centro da página.

Outro comportamento que sempre acompanha as janelas modais é uma sobreposição de cor - geralmente preta ou branca e com uma certa opacidade, entre o conteúdo da sua página e a sua janela modal.

Com a sobreposição de cores, é possível enfatizar a diferença do modal em relação ao resto da página, e até mesmo impedir que o usuário interaja com o conteúdo. Isto costuma ser feito com outro elemento fixo, ocupando todo o espaço útil do navegador, que costumamos chamar de *overlay*.

```
<div class='notice'>
  <h1>Atenção</h1>
  <p>Aqui você pode exibir alguma mensagem importante, que ela irá
    sobrepor todo o conteúdo do seu site.</p>
</div>
<div class='overlay'></div>
```

Podemos fazer um CSS para nosso `overlay`:

```
.overlay {
  position: fixed;
  top: 0;
  left: 0;
  background-color: rgba(0, 0, 0, 0.5);
  width: 100%;
  height: 100%;
}
```

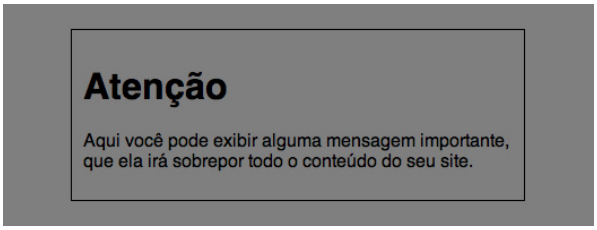


Figura 6.16: Adicionamos o overlay, mas ele está sobrepondo o modal.

Mas a nossa implementação levantou um problema - o `.overlay` ficou por cima do `.notice`.

Como os elementos com `position: fixed` ou `position: absolute` são empilhados por cima da página, a ordem do posicionamento é dada pela ordem dos elementos no HTML - então como `.overlay` aparece *após* o `.notice`, ele será posicionado por cima de tudo.

Podemos corrigir isto de duas maneiras - ou colocamos o código HTML da `div` antes do HTML do nosso modal, ou podemos controlar a ordem das posições com a propriedade `z-index`, da seguinte forma:

```
.notice {
  z-index: 1;
}
```

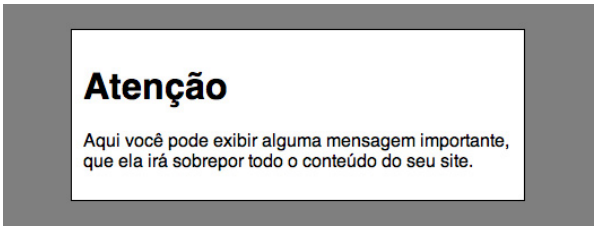


Figura 6.17: Problema de sobreposição resolvido!

A propriedade `z-index` redefine a ordem dos elementos que estão posicionados no mesmo contexto (no caso dos nossos elementos, estão posicionados de acordo com o `body`).

Elementos sem a propriedade `z-index` sempre ficarão atrás daqueles que possuem a propriedade, e entre os que possuem o `z-index` definido, os que tiverem um valor maior serão posicionados a frente dos de valor menor.

Elementos em contextos diferentes - como dois elementos com `position: absolute`, porém contidos em elementos diferentes com `position: relative` - não compartilham da mesma *sequência* de aplicação do `z-index`, então é bem fácil se complicar com a ordem dos elementos quando se possui uma página muito complexa.

Para casos assim, é indicado que os elementos do seu modal sejam os últimos elementos de *todo* o seu HTML. Além de evitar problemas com a mudança de `position` de outros elementos, você garante que ele estará por cima do seu conteúdo de acordo com a ordem padrão dos elementos.

O comportamento do modal via JavaScript

Com este código pronto, podemos começar a adicionar o comportamento em JavaScript para exibir ou esconder o modal. Vamos esconder os 2 elementos e adicionar um *link* para exibir o modal quando necessário.

```
<a id='show-modal' href='javascript:;'>Exibir modal</a>
```

```
.notice, .overlay {
  display: none;
}
```

Exibir modal

Figura 6.18: O link para exibir o modal, sem estilo nenhum.

O código JavaScript para exibir o modal é bastante simples caso você esteja utilizando jQuery.

```
$('#show-modal').on('click', function() {
  $('.overlay, .notice').show();
});
```

Estas linhas de JavaScript são responsáveis pelo seguinte comportamento: iremos associar uma função ao evento de `click` de qualquer elemento que atenda o seletor `#show-modal` (o botão, neste caso). Esta função irá selecionar os elementos do nosso modal e irá modificar a propriedade `display` para exibir eles para o usuário.

Agora que conseguimos exibir o modal, precisamos escondê-lo também, certo? Para isso, vamos precisar de um botão de fechar, dentro do modal, que pode ser posicionado por cima do canto superior direito do elemento, quase que saindo do espaço do modal. Seu HTML é um simples link:

```
<div class='notice'>
  <h1>Atenção</h1>
  <p>Aqui você pode exibir alguma mensagem importante, que ela irá
    sobrepor todo o conteúdo do seu site.
  </p>
  <a href='javascript:;' class='close'>Fechar</a>
</div>
```

E o seu CSS:

```
.notice .close {
  background-color: #FF1A2D;
  color: #FFF;
  padding: 5px;
  position: absolute;
  right: -25px;
  top: -12px;
}
```

Figura 6.19: O botão para fechar o nosso modal.

Como o `.notice` já possui uma definição de `position`, fica fácil posicionar o botão de *"Fechar"* em relação a posição do modal com outra combinação de `position` + coordenadas. E o que resta é adicionar o evento de `click` deste botão para esconder os elementos do modal.

```
$('.overlay, .close').on('click', function() {
  $('.overlay, .notice').hide();
});
```

Vamos associar o evento de `click` de dois elementos - o `.overlay` e o `.close`, separando os dois seletores com uma vírgula - e o evento irá esconder tanto o `.overlay` quanto o `.notice`. Assim, caso o usuário clique *fora* da janela ela será fechada, e esta ação não ficará reduzida para apenas o pouco espaço ocupado pelo botão. E é possível ir além: associar o evento de `keydown` do teclado do seu usuário para esconder o modal caso ele tecle `ESC`. Tente adicionar este comportamento e veja como fica o resultado final!

6.6 Como escolher os métodos para posicionar os seus elementos

Passamos por algumas opções diferentes de como estruturar e posicionar os nossos elementos nos layouts, mas a escolha de se devemos usar `display: inline-block` ou `position: absolute` em um determinado caso pode lhe custar alguns minutos, por isso é bom ter em mente em quais cenários cada combinação de propriedades é mais indicada para ser usada.

Enquanto resultados similares podem ser alcançados com técnicas parecidas, soluções mais simples e que requerem menos código são sempre recomendadas para facilitar a manutenção do seu projeto.

Se o espaçamento entre elementos com `display: inline-block` não for um problema para você, como na estruturação de um campos e rótulos em um formulários, utilize esta propriedade, por ser a implementação mais simples. Para casos de alinhamento horizontal mais complicados, em que o espaçamento causado pelo `inline-block` não seja uma opção, combine `float` com algum elemento com `clear` ou use o truque do `clearfix` no *container* dos elementos em questão.

Para casos de sobreposição de elementos ou cenários onde você precisa posicionar um elemento em um ponto específico, como no uso de pseudo elementos como ícones ou decorações visuais, ou botões e títulos que vazam o espaço de seus *containers*, combine o `position: absolute` com o `position: relative` para ter total controle da posição dos seus elementos.

6.7 Grids - um padrão de estrutura para as suas páginas

Grids são um sistema de regras para definir um número de colunas disponíveis nos seus layouts para definir a largura e espaçamento dos seus elementos de acordo com o tamanho padrão e espaçamento entre uma coluna e outra. Você pode utilizar qualquer tipo de grid, de 12 ou 16 colunas, de larguras fixas ou relativas, com espaçamento entre colunas ou não.

Diversas implementações e frameworks estão disponíveis por ai, como o clássico 960.gs (http://960.gs/), Twitter Bootstrap ou o Foundation (você pode ler mais a respeito destes últimos na seção 10.1), mas você pode criar o seu próprio grid para os seus projetos.

Pegaremos o seguinte grid como exemplo: 9 colunas de 50px cada, com 10px de margens laterais. Podemos definir que a coluna principal de conteúdo irá ocupar o espaço de 6 colunas do grid, enquanto a barra lateral ocupará 3 colunas.

```
<article class='column six-columns'>
  <!-- Conteúdo principal... -->
</article>
<aside class='column three-columns'>
  <!-- links de navegação -->
</aside>
```

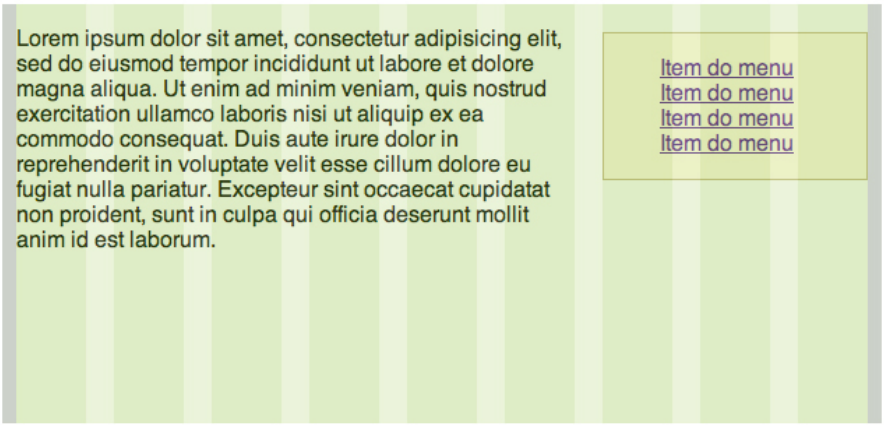


Figura 6.20: Um exemplo de conteúdo distribuído entre colunas de um grid.

As definições de classes para a quantidade de colunas que um elemento pode ocupar são as regras de largura que os seus elementos devem seguir. Neste exemplo, não podemos ter uma barra lateral que ocupe `200px` ou espaçar seções adjacentes em mais do que `20px`. Tais padrões de tamanho e espaçamento podem economizar muito trabalho na definição dos layouts e na escrita do seu CSS.

Antes de abraçar o uso de grids como uma solução perfeita para os seus problemas, não deixe se atentar a pontos importantes ao colocar a mão na massa: o seu grid que deve se adequar ao seu conteúdo, e não o contrário. E é muito fácil de se complicar definindo um número maior de colunas do que o necessário - um grid de 3 ou 5 colunas pode ser o suficiente para a maioria dos seus projetos.

6.8 Posicionando elementos com CSS 3

Existem diversas técnicas diferentes que podemos utilizar para diagramar e posicionar os elementos nos nossos projetos, mas nem todos os cenários e problemas possuem soluções simples e elegantes - é comum encontrar ajustes e correções feitas via JavaScript para problemas que deveriam ser solucionados com CSS.

Algumas adições do CSS3 permitem que problemas antigos sejam solucionados com códigos novos, o que é razão de festa para qualquer desenvolvedor.

Mas **atenção**, pois os módulos do CSS3 relacionados a posição e controle de conteúdo não estão tão maduros quanto as de propriedades como `box-shadow` ou `border-image`, então é importante conferir a situação do suporte dos navegadores a essas novidades.

Distribuição de conteúdo em colunas

Distribuir um bloco de conteúdo - textos, links ou imagens - em diversas colunas ficou muito fácil. Em vez de dividir o conteúdo manualmente em elementos separados e se preocupar com a largura e espaçamento entre eles você pode simplesmente deixar esta tarefa para o navegador, com a propriedade `column-count`, onde você define a quantidade de colunas que o seu elemento terá, e fica a cargo do navegador distribuir o conteúdo e dimensionar cada uma, independente do tamanho do seu elemento.

Além disso, existem propriedades adicionais para definir novos detalhes do comportamento das suas colunas, como `column-gap` (espaçamento entre colunas), `column-rule` (a borda que dividirá as colunas) ou `column-fill` (a regra utilizada para distribuir o conteúdo entre as colunas). Um exemplo a seguir demonstra quão

simples é dividir um pedaço de conteúdo entre colunas.

```
<p>
  Firefly é uma série de ficção científica criada pelo escritor/diretor
  Joss Whedon, criador de Buffy: A Caça Vampiros, Angel e Dollhouse,
  com sua companhia de produção Mutant Enemy Productions. Seu ambiente
  futurista naturalista, modelado a partir de temas dos filmes de Western
  tradicionais, apresenta um cenário de ficção científica atípico para a
  narrativa. Whedon também trabalhou como produtor executivo,
  junto com Tim Minear.
</p>
```

E o CSS que utiliza as novas regras para colunas:

```
p {
  column-count: 3;
  column-gap: 10px;
  column-rule: 1px solid #666;
  font-size: 0.9em;
  width: 500px;
}
```

Firefly é uma série de ficção científica criada pelo escritor/diretor Joss Whedon, criador de Buffy: A Caça Vampiros, Angel e Dollhouse, com sua companhia de produção Mutant Enemy Productions. Seu ambiente futurista naturalista, modelado a partir de temas dos filmes de Western tradicionais, apresenta um cenário de ficção científica atípico para a narrativa. Whedon também trabalhou como produtor executivo, junto com Tim Minear.

Figura 6.21: A divisão automática do conteúdo em colunas com CSS 3.

Elementos flexíveis

Um módulo bastante interessante do CSS3 é o **CSS Flexible Box Layout Module**, que introduz diversas propriedades novas para definir regras de posicionamento e alinhamento relativas, onde os elementos presentes em um *container* `flex` serão organizados automaticamente pelo navegador, de acordo com as definições feitas, sem a necessidade de se fixar tamanhos ou forçar alinhamentos com `float` ou `position`.

Tudo se baseia no novo valor da propriedade `display: flex`. Com ele, a gama de propriedades do `flexbox` estarão disponíveis para os elementos contidos dentro deste *container*. Como, por exemplo, forçar uma orientação vertical ou horizontal, sem a necessidade de aplicar a propriedade `float` nos elementos. Ou, então, distribuir a largura do elemento entre seus filhos, alterar a ordem de exibição ou forçar um espaçamento igual para todos.

Mas como nem tudo são flores, a especificação deste módulo sofreu uma reescrita pesada e ainda está em constante atualização. A versão antiga da especificação foi implementada em diversos navegadores, mas o novo formato ainda não está maduro o suficiente para se utilizar em seus projetos. Mas eu sugiro que você fique de olho nas novidades relacionadas ao `flexbox`, já que logo mais ele poderá ser adotado em seus projetos.

Vamos dar uma olhada de perto no que é possível fazer com o `flexbox`: primeiro, vamos precisar de um *container* com 3 elementos, onde eles irão preencher toda a largura disponível igualmente:

```
<div class='flexbox'>
  <span class='one box'>1</span>
  <span class='two box'>2</span>
  <span class='three box'>3</span>
</div>
```

Esse *container* terá `300px` de largura, então cada um dos elementos irá ocupar `100px`.

```
.flexbox {
  /*
    De acordo com a nova especificação, o valor "flexbox"
    deverá ser substituído por apenas "flex".
  */
  display: flexbox;
  height: 60px;
  width: 300px;
}
.box {
  color: #FFF;
  display: block;
  flex:  auto;
  font-weight: bold;
  height: 30px;
```

```
  line-height: 30px;
  text-align: center;
}
```

Para identificar melhor cada um dos elementos, vamos definir cores de fundo diferentes para cada um deles.

```
.one { background-color: #8820DD; }
.two { background-color: #2BB15A; }
.three { background-color: #755322; }
```

Figura 6.22: Três elementos flexíveis, distribuídos por 300px de largura.

Neste exemplo temos duas novidades: o valor `flexbox` (que será substituído por `flex` no futuro), que marca o *container* como um elemento flexível, e a propriedade `flex`, onde você pode definir diversos comportamentos diferentes para o elemento.

Neste caso, o valor `auto` define que ele irá ocupar o máximo de espaço possível, distribuindo os `300px` entre cada um deles. Mas isto pode ser alcançado utilizando porcentagens na propriedade `width` e flutuando os elementos, certo?

É aqui que mora a diferença entre estes métodos. Como não estamos definindo larguras fixas para os elementos flexíveis, podemos adicionar mais conteúdo com a classe `.box` que a largura dos elementos será ajustada pelo navegador - você não precisa pular no CSS e recalcular as larguras dos elementos.

```
<div class='flexbox'>
  <span class='one box'>1</span>
  <span class='two box'>2</span>
  <span class='three box'>3</span>
  <span class='four box'>4</span>
  <span class='five box'>5</span>
</div>
```

Não vamos deixar de lado as cores do quarto e quinto elemento:

```
.four { background-color: #02A3A3; }
.five { background-color: #C6363D; }
```

Figura 6.23: Os cinco elementos ocupando os 300px do container.

Não ter que ajustar as larguras ou definir novas classes é grande vantagem - os elementos flexíveis podem ser criados dinamicamente, variando de 3 a 10, e a responsabilidade de organizá-los e definir os seus tamanhos fica a cargo do navegador.

Capítulo 7

Melhorando os seus formulários

Formulários são um dos pontos mais críticos da maioria dos sites: além de serem a porta de entrada para seus usuários e clientes se comunicarem ou proverem informações para você, sempre estão relacionados a processos importantes para o seu negócio, seja finalizando a compra em um *e-commerce*, criando artigos para um portal ou enviando novas fotos para uma rede social. Por isso, devemos criar formulários que não fiquem no caminho para atrapalhar nossos usuários em suas tarefas nos projetos que criamos.

7.1 O que temos no HTML 5

Uma das adições mais interessantes do HTML5 são os diversos campos novos para formulários e novos comportamentos para enriquecer a experiência disponível para os formulários. Novos tipos de campos, como `email`, `search` e `range` e atributos como `placeholder`, `pattern` e `required` dizem mais sobre que tipo de informações nossos formulários precisam e reduz a necessidade de usarmos plugins em JavaScript para auxiliar a formatação dos elementos.

No assunto de novos campos, temos diversos novos valores para o atributo `type` da tag `input`. Tipos como `email`, `tel`, `number` e `url` agregam a semântica do seu HTML para definir o tipo de informação que o formulário precisa, apesar de não causar mudanças drásticas ao visual dos seus elementos. Os tipos `color`, `range` e `date` permitem que o navegador renderize elementos específicos para escolher o valor apropriado para esses tipos de informação.

Além disso, diversos novos atributos estão sendo adicionados para melhorar a experiência dos usuários ao interagir com os formulários, como o atributo `placeholder`, que exibe um texto específico quando o campo não está preenchido - que deve ser usado para exemplificar formatos e expor mais detalhes sobre os campos. Para efetuar validações no navegador, os atributos `required`, `maxlength` e `pattern` permitem definir quais campos devem ser preenchidos antes do formulário ser enviado de volta para o servidor; E para o campos do tipo `file`, se pode definir que o campo aceita vários arquivos com o atributo `multiple` e até restringir os tipos de arquivos aceitos através do atributo `accept`.

Esses diversos novos tipos e atributos estão sendo consolidados e implementados aos poucos, então confira como está o suporte ao que você precisa de acordo com os navegadores que você deseja suportar. Por padrão, caso o navegador não tenha suporte a um tipo novo de campo, o elemento será tratado como um campo de texto normal `type='text'`, para garantir o funcionamento básico dos formulários.

7.2 Formulários HTML 5 nos dispositivos móveis

Enquanto campos como o `email` ou `url` podem parecer desnecessários ou problemáticos devido a inconsistências entre navegadores, eles possuem uma funcionalidade muito interessante quando usados em navegadores de dispositivos móveis, como tablets ou celulares com iOS ou Android. Os aparelhos apresentam um teclado diferente ao usuário no momento de interagir com campos específicos. Para um campo com `type='number'`, o teclado já exibe as teclas de números e não as letras. Para o campo de e-mail, o teclado padrão recebe a adição de um botão de arroba (`@`). E no caso de campos com `type="url"`, o iOS gentilmente exibe um botão para se digitar '.com' com apenas um toque. Apesar de não ser um padrão definido entre os fabricantes, essas mudanças são muito úteis para os usuários - lembre-se de tirar proveito desses campos para facilitar a vida dos seus usuários que usam tais dispositivos.

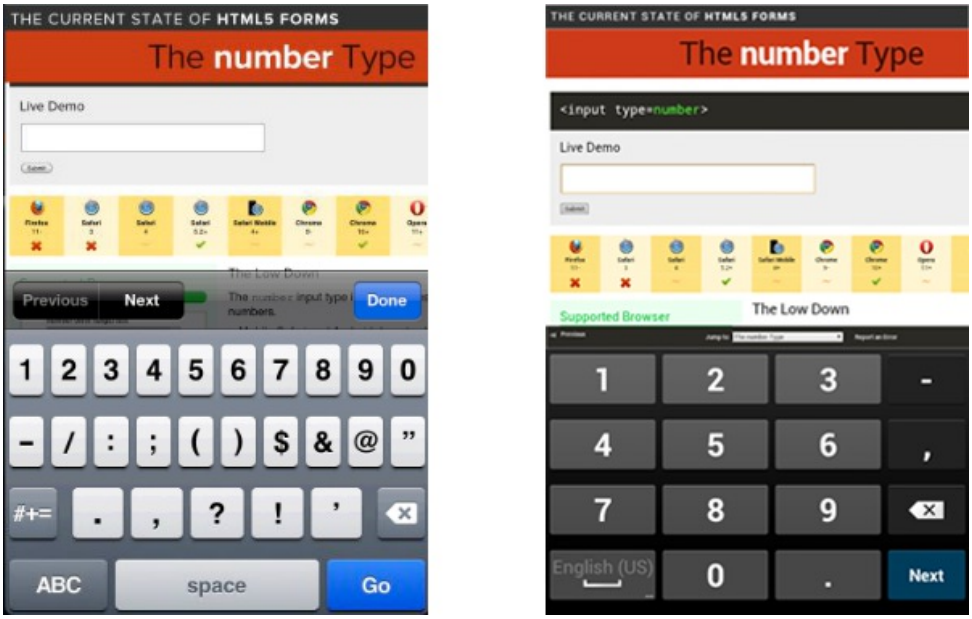


Figura 7.1: Exemplos do teclado numérico do iOS 5 e Android 4

Referências de quem vive do assunto

O Wufoo é um site que permite criar formulários sem precisar entender ou escrever HTML ou CSS e você consegue usá-los em outros sites para receber mensagens de contato, criar pesquisas e até receber pagamentos. Com o foco constante em melhorar seus formulários e prover diversas funcionalidades em cima deles.

Os desenvolvedores do Wufoo criaram uma ótima referência para os novos tipos de campos e as novas funcionalidades de formulários presentes no HTML5. Acesse http://wufoo.com/html5/ para saber mais sobre compatibilidade de navegadores, exemplos de código e mais.

7.3 Criação do primeiro formulário

Vamos começar com um exemplo básico porém bem refinado de um formulário: uma página de contato contendo campos para e-mail, assunto e mensagem. Esse

formulário terá dicas explicando como a informação deve ser preenchida em cada campo. Além disso, iremos preparar elementos para notificar os usuários caso haja problemas de validação no formulário - um passo que sempre deve ser levado em consideração ao criar páginas como essa.

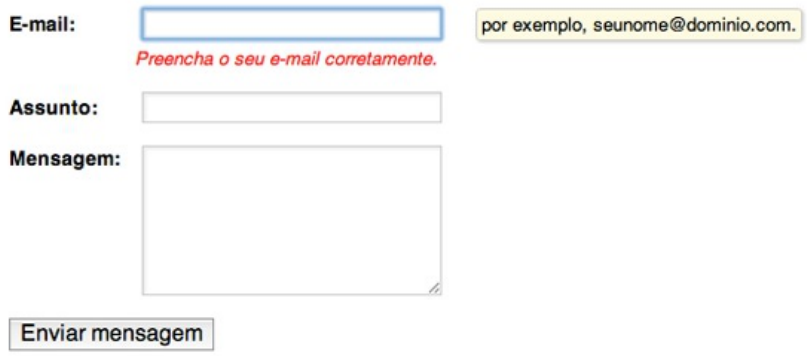


Figura 7.2: O formulário que vamos criar a seguir

Marcação básica

Vamos começar com a tag `form`, o campo de e-mail e um botão de envio:

```
<form method='post' action=''>
  <label for='email'>E-mail:</label>
  <input type='email' id='email' name='email'>

  <button>Enviar mensagem</button>
</form>
```

Nosso formulário está com o `method` definido como `post`, já que pretendemos criar uma mensagem de contato no site. Com o valor do atributo `action` com o valor vazio, o navegador irá enviar os dados do formulário para o mesmo endereço da página atual: se estamos em http://meusite/contato, ao clicar no botão de envio os dados irão para o mesmo http://meusite/contato através do método `post`.

Para separar o campo e botão de envio, vamos colocar os elementos do formulário dentro de parágrafos com a tag `p`.

```
<p>
  <label for='email'>E-mail:</label>
```

```
  <input type='email' id='email' name='email'>
</p>
```

No mesmo modelo, podemos adicionar os outros campos do formulário - um campo de texto para o assunto da mensagem e uma caixa de texto para o corpo da mensagem:

```
<p>
  <label for='subject'>Assunto:</label>
  <input type='text' id='subject'>
</p>
<p>
  <label for='content'>Mensagem:</label>
  <textarea id='content'></textarea>
</p>
```

Figura 7.3: Os campos do formulário

Com a estrutura básica e os campos necessários do nosso formulário, podemos passar para o próximo passo e adicionar regras de estilo para o nosso formulário.

7.4 Alinhamento e estilos visuais nos campos e formulários

Vamos começar alinhando todos as caixas de texto na mesma posição. Para isso, precisamos fixar a mesma largura de cada `label`, e vamos aproveitar para deixá-los com o texto em negrito:

```
label {
  display: inline-block;
```

```
  font-size: 0.9em;
  font-weight: bold;
  width: 90px;
}
```

Definindo a propriedade `display` como `inline-block`, podemos fixar a largura e manter os campos ao lado dos labels, utilizando apenas `block` os campos ficariam abaixo do seu respectivo `label`, independente da largura definida. Podemos fazer o mesmo para as tags `input` e `textarea` para manter todos os elementos devidamente alinhados.

```
input, textarea {
  display: inline-block;
  width: 200px;
}
```

Além da largura, vamos fixar também a altura da `textarea`, garantido um bom espaço para se digitar um texto comprido.

```
textarea {
  height: 100px;
}
```

Figura 7.4: Os campos e labels devidamente alinhados com CSS.

Hora de melhorar a aparência dos campos do formulário: vamos trocar as bordas padrões dos campos, adicionar uma sombra e espaçamento interno e reduzir um pouco a fonte:

```
input, textarea {
  border: 1px solid #CCC;
  box-shadow: inset 2px 2px 2px #EEE;
  font-size: 0.9em;
  padding: 2px 5px;
}
```

Figura 7.5: Os campos com as mudanças de estilo que criamos.

Pronto, alterando pequenos detalhes do estilo padrão dos elementos conseguimos chegar a um visual sutil porém bastante agradável. Para compor a sombra, usamos a opção `inset`, que define que a sombra será exibida dentro do elemento e não fora dele. A sombra não será exibida em navegadores que não possuem suporte a `box-shadow`, mas isso não impedirá ou complicará o uso do formulário, garantido sua funcionalidade em navegadores sem suporte a essas propriedades do CSS 3, como versões antigas do Mozilla Firefox e do Internet Explorer.

7.5 Exibição de mensagens de ajuda

Agora que temos os campos devidamente estilizados, podemos adicionar um novo elemento para mostrar uma descrição dos campos, para que auxilie as pessoas que forem preencher o formulário - mensagens de ajuda, que podem explicar a respeito de limite de caracteres, formatos específicos e detalhes extras a respeito do seu formulário. Para representar essas mensagens, usaremos a tag `span` com a classe `hint`.

```
<p>
  <label for='email'>E-mail:</label>
```

```
  <input type='email' id='email' name='email'>
  <span class='hint'>por exemplo, seunome@dominio.com.</span>
</p>
<p>
  <label for='subject'>Assunto:</label>
  <input type='text' id='subject'>
  <span class='hint'>Nos diga qual é o seu problema ou ideia.</span>
</p>
<p>
  <label for='content'>Mensagem:</label>
  <textarea id='content'></textarea>
  <span class='hint'>
    Escreva a sua mensagem, em até 500 caracteres.
  </span>
</p>
```

Agora, precisamos adicionar um estilo a essas mensagens para elas se destacarem na interface, chamando a atenção dos usuários.

```
.hint {
  background-color: #FFFBE4;
  border-radius: 3px;
  border: 1px solid #CCC;
  box-shadow: 1px 1px 3px rgba(0,0,0,0.2);
  display: inline-block;
  font-size: 0.8em;
  margin-left: 20px;
  padding: 3px;
}
```

Um tom de amarelo como fundo, bordas levemente arredondadas (usando a propriedade `border-radius`) e uma sombra (com a utilização de `rgba` em vez de uma cor sólida, reduzindo bastante a opacidade da cor preta). Além disso, uma margem de `20px` para distanciar um pouco as mensagens dos seus campos.

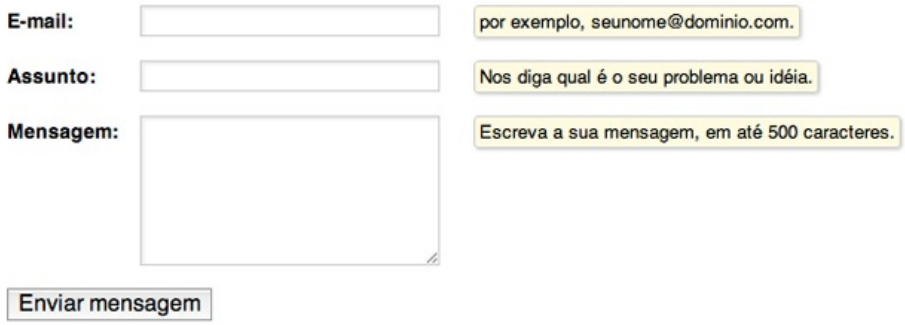


Figura 7.6: Todas as mensagens de ajuda junto dos campos.

De certa forma, exibir todas essas mensagens ao mesmo tempo pode atrair atenção demais e atrapalhar o fluxo de uso do formulário. **Por quê não exibir as mensagens conforme o formulário for preenchido? Isto é possível usando apenas CSS!**

Primeiro, vamos esconder todas as mensagens alterando a propriedade `display` de `inline-block` para `none`. Com todas elas escondidas, vamos exibir apenas a mensagem relacionada ao campo que esteja em foco - ao selecionar o campo de "E-mail", por exemplo, a mensagem de ajuda para este campo será exibida, quando se passar para o próximo campo, "Assunto", a mensagem do campo de "E-mail" desaparece novamente e a mensagem para o campo de "Assunto" é exibida.

Para isso utilizaremos o seletor de *sibling* (irmão, no caso), usando o símbolo de adição +:

```
.hint {
  display: none;
}

input:focus + .hint, textarea:focus + .hint {
  display: inline-block;
}
```

O seletor pode ser lido como "um elemento com a classe `hint`, posicionado exatamente depois de um elemento `input` (ou `textarea`) que esteja em foco no navegador". Assim, as mensagens se tornarão visíveis (tendo a propriedade `display` sobrescrita de `none` para `inline-block`) conforme os campos recebem e perdem o foco.

Figura 7.7: Agora, somente uma mensagem é exibida por vez.

Mas e o atributo "placeholder"?

O atributo `placeholder` foi originalmente criado para exibir frases de ajuda e exemplos de formatos dentro dos próprios campos criados com a tag `input`, mas com ele não possuímos tanta flexibilidade para mudanças de estilo e uma melhor interação com os usuários - no nosso exemplo podemos até adicionar links para páginas de ajuda mais completas e utilizar outros formas de estilo - como palavras em negrito ou em outras cores - para realçar a mensagem a ser exibida.

7.6 Mostre mensagens de erro

Com nossas mensagens de ajuda em posição, temos um outro tipo de mensagem a exibir no formulário: mensagens de erros e de validações. Não pretendo entrar em detalhes em como tais validações devem ser feitas (seja utilizando JavaScript no navegador ou processando isso com a sua linguagem de programação preferida como Ruby, Python ou Java em um servidor externo), mas sempre devemos guardar um espaço para tais mensagens e definir um estilo específico para elas.

Vamos simular que o formulário foi enviado sem o campo "E-mail" e devemos notificar o usuário disso. Começamos com um elemento `span` com a classe `error`, para identificar exatamente quais elementos `span` estão exibindo tais mensagens.

```
<p>
  <label for='email'>E-mail:</label>
```

```
  <input type='email' id='email'>
  <span class='hint'>por exemplo, seunome@dominio.com.</span>
  <span class='error'>Preencha o seu e-mail corretamente.</span>
</p>
```

Vamos posicionar a mensagem exatamente abaixo do `input`.

```
.error {
  color: red;
  display: block;
  font-size: 0.8em;
  font-style: italic;
  margin: 5px 0 0 90px;
}
```

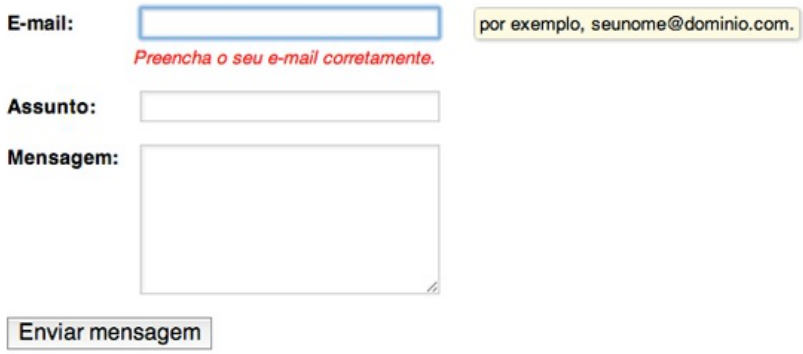


Figura 7.8: A mensagem de erro abaixo do campo de E-mail.

O valor `block` na propriedade `display` vai garantir que o elemento fique abaixo dos demais, do `label`, `input` e o outro `span`. Os valores de margem posicionam este `span` abaixo da caixa de texto, com `5px` de distância entre eles. A cor vermelha e o texto em itálico (graças ao `font-style`), garantem um certo destaque em relação ao outros elementos que estão em tons de preto e cinza.

Agora basta replicar o mesmo exemplo para os outros campos do formulário - "Assunto" e "Mensagem", garantindo o espaço reservado para suas mensagens de erro de validações!

Dessa maneira, bastaria que quando a validação fosse feita, seja no lado do servidor ou no lado do cliente, houvesse uma regra para deixar esse novo `span` visível quando acontece o erro de validação.

7.7 Levando o usuário direto ao que importa com o autofocus

Podemos adicionar mais um truque ao formulário: o `autofocus`. Ao carregar a página, o cursor irá focar o campo de "E-mail" para facilitar o trabalho dos usuários que querem utilizar o formulário. A implementação é bem simples, bastando adicionar um atributo ao campo:

```
<input type='email' id='email' autofocus>
```

Sim, apenas `autofocus`. Não precisamos adicionar nenhum valor ao atributo para ativá-lo. Dessa forma o campo de "E-mail" receberá o foco da janela e a mensagem de ajuda também aparecerá ao carregar a página do formulário.

Apenas alguns pontos importantes que se deve prestar atenção ao usar este atributo: caso você tenha mais de um campo com `autofocus`, eventualmente o último campo presente no HTML receberá o foco. E caso seja necessário descer a barra de rolagem da janela para chegar ao campo, o navegador irá descer a página para exibir o campo focado, pulando qualquer conteúdo anterior ao formulário. Com exceção de navegadores em dispositivos móveis (como o Mobile Safari do iOS 5 e o navegador do Android 2.3), todos as últimas versões dos navegadores de desktop possuem suporte ao `autofocus`.

7.8 A flexibilidade do atributo placeholder

Comentei a respeito do atributo `placeholder` e como utilizamos meros elementos para substituir o que este atributo foi criado para fazer. Agora vamos inverter o cenário - utilizaremos o `placeholder` de campos do formulário para, visualmente, substituir o seu `label` - essa abordagem pode ser útil para formulários menores e simples, quando se precisa economizar alguns pixels da sua página.

O HTML

O HTML para este formulário é bastante simples - não vamos precisar de parágrafos para cada campo, mas vamos ter o `label` de cada campo presente, mesmo considerando que não vamos exibir eles na interface final:

```
<form method='post' action=''>
  <label for='username'>Usuário</label>
```

```
  <input type='text' id='username'>
  <label for='password'>Senha</label>
  <input type='password' id='password'>
  <button>Entrar no site</button>
</form>
```

Existem duas razões para termos o `label` do campo de "Usuário" e de "Senha": primeiro, precisamos disponibilizar tais elementos para auxiliar leitores de telas para deficientes visuais a navegarem pela nossa página. E segundo, caso o navegador que for utilizado para acessar a página não carregue todo o nosso CSS ou não possua suporte ao atributo `placeholder`, o formulário ainda será funcional para quem o utilizar.

Com isso esclarecido, vamos continuar e adicionar os textos para o `placeholder` de cada campo.

```
<form method='post' action=''>
  <label for='username'>Usuário</label>
  <input type='text' id='username' placeholder='Usuário'>
  <label for='password'>Senha</label>
  <input type='password' id='password' placeholder='Senha'>
  <button>Entrar no site</button>
</form>
```

Com os textos em posição, vamos ao CSS.

Aplicando o CSS necessário

Nosso primeiro passo aplicando o CSS neste formulário será esconder o `label` de cada campo e empilhar o campo de "Usuário" junto ao de "Senha", empurrando o botão para baixo:

```
label {
  display: none;
}

input {
  display: block;
}

button {
  margin-top: 5px;
}
```

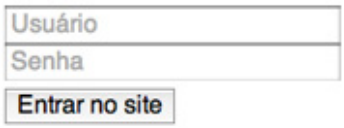


Figura 7.9: Os campos com o atributo 'placeholder'.

E com um pouco mais de código, embelezamos o formulário com bordas e sombras.

```
input {
  border-radius: 3px 3px 0 0;
  border: 1px solid #999;
  box-shadow: 1px 1px 5px rgba(0,0,0,0.2);
  font-size: 0.8em;
  padding: 5px;
}

input[type='password'] {
  border-radius: 0 0 3px 3px;
  border-top: none;
}
```

Figura 7.10: Nosso formulário um pouco mais apresentável.

Bastante simples, certo? Arredondamos apenas os cantos superiores (deixando os dois últimos valores do `border-radius` com o valor `0`), aplicamos uma borda cinza e uma leve sombra utilizando `rgba`. E para o campo de "Senha", invertemos os valores do `border-radius`.

Podemos alterar o estilo do texto dos campos, mas o mesmo não é aplicado ao texto exibido do `placeholder`. Para isso, precisamos utilizar um pseudo-seletor específico para este texto. Devido a problemas de compatibilidades, navegadores diferentes dão um nome diferente para o mesmo seletor, então precisamos especificar todos eles, assim:

```
::-webkit-input-placeholder {
  color: red;
}

::-moz-placeholder {
  color: red;
}

-ms-input-placeholder {
  color: red;
}
```

Figura 7.11: Aplicando o pseudo-seletor para placeholders.

O `-webkit-input-placeholder` é direcionado para os navegadores baseados no WebKit - como Safari e o Google Chrome. Os demais, `-moz-placeholder` e `-ms-input-placeholder`, atendem o Mozilla Firefox e o Internet Explorer 10, respectivamente. Cada navegador processará um seletor diferente, e precisamos repetir as propriedades entre eles.

Apesar da repetição, ganhamos total controle sobre o estilo do placeholder - podemos alterar cores, tamanhos e fundos. Por exemplo, podemos deixar o texto digitado no campo em negrito e o texto do `placeholder` em itálico e cinza, ajudando a diferenciar entre um campo preenchido e um campo em branco.

```
input {
  font-weight: bold;
```

```
}

::-webkit-input-placeholder {
  color: #666;
  font-style: italic;
  font-weight: normal;
}

:-moz-placeholder {
  color: #666;
  font-style: italic;
  font-weight: normal;
}

-ms-input-placeholder {
  color: #666;
  font-style: italic;
  font-weight: normal;
}
```

Figura 7.12: A versão final dos campos do formulário.

O formulário está completo - com o `placeholder` em uso e o `label` de cada campo escondido - apenas visualmente, utilizando CSS - deixando o formulário ocupar um espaço mínimo.

Temos apenas um ponto, opcional, a melhorar: em navegadores que não suportam o uso de `placeholder` - como versões do Internet Explorer anteriores a 10 - nosso formulário não exibe apenas os 2 campos, sem texto algum indicando sobre o que eles são.

A seguir, vamos adotar uma estratégia para identificar os casos onde não se tem suporte ao atributo e ainda sim podemos permitir que os usuários utilizando tais

navegadores consigam usar o formulário normalmente.

Detectando funcionalidades com JavaScript e ajustando o estilo

Para identificar que o navegador em uso não possui suporte ao `placeholder` vamos utilizar um pouco de JavaScript, mas, em vez de tentar identificar em qual navegador estamos fazendo malabarismos com o `navigator.userAgent`, será mais seguro testar se o navegador possui realmente suporte a funcionalidade que precisamos. Eis o por quê: **podemos escrever um código específico para atender o Internet Explorer, mas isso não atingirá versões antigas do Mozilla Firefox, por exemplo. Detectando a funcionalidade em vez que possuir uma lista de versões de navegadores que precisam de uma correção garante um código mais simples e conciso**.

Vamos escrever um teste específico para o atributo `placeholder` e caso você precise testar outras funcionalidades que possam não existir talvez será necessário usar alguma outra lógica específica. A biblioteca 10.4 **Modernizr** foi criada para agrupar diversos testes de funcionalidades diferentes para facilitar a nossa vida. Mas neste caso vamos escrever o código nós mesmos, por ser um teste tão simples.

```
function supportsPlaceholder() {
  return 'placeholder' in document.createElement('input') &&
         'placeholder' in document.createElement('textarea');
}
```

A função é bem simples: caso a propriedade `placeholder` exista nos elementos `input` e `textarea`, ela retornará `true`. Caso algum dos elementos (ou ambos) não possua o atributo, ela retornará `false`. O teste não servirá de nada se não o utilizarmos, certo? Vamos fazer o seguinte: caso o navegador *não* passe no teste (retornando `false` a chamada desta função), vamos adicionar uma classe ao elemento `html` (o elemento raiz da página) para, através do CSS, aplicarmos um estilo diferente para navegadores sem suporte ao `placeholder`. Utilizando o jQuery, biblioteca mais usada para trabalhar com o DOM, ficamos com o seguinte código:

```
jQuery(function($) {
  function supportsPlaceholder() {
    return 'placeholder' in document.createElement('input') &&
           'placeholder' in document.createElement('textarea');
  }
```

```
  if(!supportsPlaceholder()) {
    $('html').addClass('no-placeholder');
  }
});
```

Assim, o nosso CSS poderá aplicar regras específicas para páginas com a classe `no-placeholder`. Neste caso podemos exibir o `label` de cada campo e fazer outras mudanças no estilo do formulário.

```
.no-placeholder input {
  border-radius: 3px;
  border-top: 1px solid #999;
}

.no-placeholder label {
  display: block;
}
```

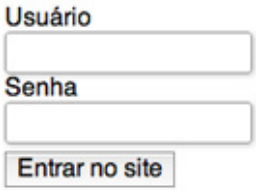


Figura 7.13: Como formulário será visualizado em navegadores sem suporte.

Assim, o formulário se mantém funcional quando o navegador usado não tiver suporte ao que precisamos e os navegadores atuais terão o comportamento padrão que desenvolvemos anteriormente.

7.9 Aplicando CSS3 em botões

Talvez você tenha reparado nos exemplos anteriores que não chegamos a trabalhar o estilo dos botões dos formulários. Deixei o assunto para ser tratado em separado por considerá-lo uma arte a parte.

Nos capítulos anteriores entramos em detalhes sobre gradientes, sombras e bordas, que são utilizados em conjunto para aprimorar o visual de botões e links sem a necessidade de se usar imagens para isso.

Criando um botão completo

De volta ao formulário de contato que criamos anteriormente e utilizando o que aprendemos sobre gradientes e bordas arredondadas, vamos começar com um sutil botão branco:

```
button {
  background-color: #E5E5E5;
  background-image: linear-gradient(top, #FFF, #E5E5E5);
  border: 1px solid #AAA;
  border-radius: 3px;
  color: #000;
  font-size: 0.8em;
  font-weight: bold;
  padding: 5px 10px;
}
```

Definimos um gradiente usando branco (`#FFF`) e cinza (`E5E5E5`), deixando o cinza como cor de fundo caso não exista suporte a gradientes. Uma borda um pouco mais escura que o cinza do gradiente, levemente arredondada e alguns toques de tamanho de fonte e espaçamento para chegar ao seguinte resultado:

Figura 7.14: O botão com gradiente e bordas.

Simples, não? É importante não usar cores muito diferentes no gradiente ou deixar o botão grande demais para chamar mais atenção que os demais elementos do formulário. Mas ainda temos trabalho a fazer neste botão!

Para ficar mais elegante, podemos adicionar estilos para os estados de uso do botão, usando pseudo seletores como o `:hover` e `:active`, para garantir a resposta visual a interação com o formulário. Vamos começar pelo `:hover`, alterando as cores do botão.

```
button:hover {
  background-color: #285582;
  background-image: linear-gradient(top, #3775B3, #285582);
```

```
  border-color: #204569;
}
```

Figura 7.15: Ao posicionar o mouse em cima do botão, ele muda do branco e cinza para o azul.

Do branco sutil ao azul. Devido a essa inversão brusca, a cor do texto do botão precisa ser mudada também, assim:

```
button:hover {
  background-color: #285582;
  background-image: linear-gradient(top, #3775B3, #285582);
  border-color: #204569;
  color: #FFF;
  text-shadow: -1px -1px 0 rgba(0, 0, 0, 0.6);
}
```

Figura 7.16: Bem melhor a cor branca no botão azul, não acha?

Além de mudar a cor para branco, adicionamos uma sombra no texto para ajudar na leitura - geralmente não precisamos usar mais do que 1 ou 2 pixels quando se trata de `text-shadow`, basta ajustar qual o ângulo desejado. O uso do `rgba` aqui pode ser trocado pela cor sólida - `#000` - se quiser. Por último, fechamos com estilo do botão com um par de sombras:

```
button:hover {
  background-color: #285582;
  background-image: linear-gradient(top, #3775B3, #285582);
  border-color: #204569;
  box-shadow: inset 0 1px 0 rgba(255, 255, 255, 0.3), 0 0 3px #3775B3;
```

```
  color: #FFF;
  text-shadow: -1px -1px 0 rgba(0, 0, 0, 0.6);
}
```

Figura 7.17: O estado de 'hover' finalizado com as sombras.

A propriedade `box-shadow` contém duas sombras bastante distintas - a primeira é uma sombra interna (atente-se ao `inset`) para clarear um pouco o topo do botão. A segunda, em volta do elemento, para demonstrar um certo brilho ao seu redor.

Agora, para o estado `:active`, vamos escurecer *bastante* o botão, dando a ilusão do elemento estar realmente pressionado pelo mouse.

```
button:active {
  background-color: #204569;
  background-image: none;
  border-color: #1A3754;
  box-shadow: inset 0 2px 1px rgba(0,0,0, 0.15);
}
```

Trocamos o par de sombras por uma sombra escura interna, removemos o gradiente azul do botão e escurecemos as cores das bordas e dos fundos.

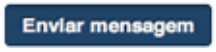


Figura 7.18: O estilo do botão quando pressionado.

Pronto, temos os 3 principais estados do nosso botão devidamente estilizados combinando as propriedades de gradiente e sombras. É muito fácil ajustar os tons para alguma outra cor, como um vermelho ou verde - simplesmente alterando os hexadecimais usados no código.

Regras básicas para definir estados de botões

É muito comum ver pela internet diversos sites que aplicam as mais diversas mudanças em estilos para `:hover` e `:active`, e nem sempre o resultado é agradável. Tyler Galpin, um designer canadense, sugere no seu blog três regras básicas para esses casos, cabendo a cada um escolher qual das três seguir.

Para o caso de `:hover`:

1) O botão deve ficar mais claro;

2) O botão deve ficar mais escuro;

3) O botão deve mudar de cor.

E o estado de `:active`:

1) O botão deve se mover 1 ou 2 pixels para baixo;

2) O botão deve ter o seu gradiente (caso presente) invertido;

3) O botão deve ficar ainda mais escuro.

No exemplo anterior escolhi a terceira opção de cada um, mas você pode praticar e ver os resultados ao usar alguma das outras regras sugeridas. Em seu blog, Tyler também comenta sobre o que se deve evitar nestes casos e outras dicas importantes sobre botões - http://galp.in/blog/2011/08/02/the-ui-guide-part-1-buttons/.

Reaplicando padrões em outros cenários

No formulário de login que criamos anteriormente, podemos usar as mesmas técnicas do botão anterior para criar um botão verde bastante arredondado. Vamos começar com CSS simples, sem utilizar gradientes ou sombras.

```
button {
  background-color: #6DA033;
  border: 1px solid #588129;
  color: white;
  font-size: 0.8em;
  font-weight: bold;
  padding: 5px 10px;
}
```

Figura 7.19: Um simples botão com fundo e bordas verdes.

Agora, deixamos o botão um pouco mais interessante com `border-radius`, `text-shadow` e `box-shadow`.

```
button {
  background-color: #6DA033;
  border-radius: 13px;
  border: 1px solid #588129;
  box-shadow: inset 0 1px 0 rgba(255, 255, 255, 0.4);
  color: white;
  font-size: 0.8em;
  font-weight: bold;
  padding: 5px 10px;
  text-shadow: -1px -1px 0 rgba(0, 0, 0, 0.6);
}
```

Figura 7.20: Melhorando o botão verde.

O valor de `13px` para o `border-radius` é por que o botão tem `27px` de altura - então usamos aproximadamente metade desse valor. Não podemos utilizar `50%` por que esse valor iria levar em consideração a largura do elemento, fazendo que ele tomasse a forma de uma elipse. As sombras são para realçar o texto (com o `text-shadow` preto acima do texto branco) e a borda superior. Agora, é a vez do gradiente.

```
button {
  background-color: #6DA033;
  background-image: linear-gradient(bottom, #6DA033 47%, #87C442 55%);
  border-radius: 13px;
  border: 1px solid #588129;
```

```
  box-shadow: inset 0 1px 0 rgba(255, 255, 255, 0.4);
  color: white;
  font-size: 0.8em;
  font-weight: bold;
  padding: 5px 10px;
  text-shadow: -1px -1px 0 rgba(0, 0, 0, 0.6);
}
```

Figura 7.21: Nosso botão verde com um gradiente.

Diferente do gradiente anterior, agora vamos informar as posições das cores - 47% e 55% . O primeiro tom de verde se mantém até 47% da altura do gradiente, e o segundo tom começa a partir dos 55%. O espaço entre eles - os 8% restantes - são preenchidos com a transição de uma cor para outra. Para a versão de `:hover` do botão, seguindo as indicações comentadas anteriormente, vamos escurecer 10% das cores do gradiente e das bordas do botão.

```
button:hover {
  background-color: #618F2D;
  background-image: linear-gradient(bottom, #618F2D 47%, #79B238 55%);
  border-color: #4E7324;
}
```

Figura 7.22: O estado de 'hover' para o botão verde.

E o `:active`? Podemos reutilizar a técnica para escurecer bastante o botão que usamos anteriormente, apenas ajustando as cores para o verde que está sendo aplicado agora.

```
button:active {
  background-color: #618F2D;
```

```
  background-image: none;
  border-color: #405E1E;
  box-shadow: inset 0 2px 1px rgba(0,0,0, 0.15);
}
```

Figura 7.23: Escurecendo o botão verde utilizando o 'box-shadow'.

E...pronto! Os mesmos padrões, mas aplicados com cores diferentes e leves mudanças de estilo, levam a elementos agradáveis e funcionais. Apesar de usarmos bastante CSS 3 aqui, os botões degradam muito bem em navegadores que possam não ter suporte a sombras ou gradientes. Isso se deve principalmente à presença do `background-color` e do `padding` usado para manter o tamanho dos botões.

Capítulo 8

Efeitos 101: Trabalhando com animações e transições

Uma das minhas partes favoritas do CSS 3 são a quantidade de efeitos que conseguimos criar apenas com CSS nas nossas páginas combinando algumas propriedades novas - `transform`, `transition` e `animation`. Sem eles, todos esses efeitos e animações de encher os olhos costumam ser feitos por bibliotecas de JavaScript como o jQuery ou o script.aculo.us. Além dessa dependência adicional, problemas de performance e de renderização de elementos costumam surgir em navegadores antigos ou ao trabalhar com elementos complexos. Passando a responsabilidade de todo o trabalho pesado para os navegadores, conseguimos efeitos de maior qualidade com uma taxa menor de bugs.

Apesar do suporte experimental, pois no momento todos essas 3 propriedades precisam ser usadas com prefixos específicos para cada os navegadores que os suportam, diversos sites e aplicativos usam esses recursos para efeitos sensacionais, sendo alguns deles os sites do último modelo do iPhone e do Google Nexus e a interface

web do Twitter. Não existe há razão de ter medo ou receio em usar eles em seus projetos. Vamos repassar exemplos clássicos de usos para a propriedade `transform`, como adicionar transições entre propriedades em elementos existentes, o nível de refinamento que temos em animações e como executar transformações em 3 dimensões.

8.1 Transformando elementos

Vamos começar com transformações em duas dimensões. A propriedade `transform` aceita diversas funções de efeitos diferentes definidos pela especificação da W3C, tais como:

1) **translate**: utilizado para reposicionar um elemento, movendo sua posição no eixo X e Y pelos valores informados.

2) **scale**: utilizada para redimensionar um elemento, aceitando valores referentes ao tamanho do elemento - `scale(2)` dobrará o tamanho de um elemento.

3) **skew**: permite distorcer as posições de um elemento, esticando e torcendo de acordo com os valores passados.

4) **rotate**: usada para rotacionar o ângulo dos elementos.

5) **matrix**: o mais complexo de todos, permite executar uma matriz de transformações.

As funções `translate`, `scale` e `skew` possuem versões `X` e `Y`, que permitem modificar apenas um eixo e não ambos de uma vez. As funções de `skew` e `rotate` trabalham com uma nova medida, graus, utilizando o sufixo `deg`, como em `transform: rotate(90deg)`.

A melhor forma de compreender o uso de tais funções é na práticas. Vamos utilizar algumas delas no exemplo a seguir.

8.2 Os efeitos rotate, scale, skew e translate em uma galeria de fotos

No começo de 2012, a Plataformatec contratou 4 desenvolvedores - incluindo eu - para reforçar a sua equipe. Em homenagem a isso, vamos criar uma galeria de fotos

para apresentar os novos membros do time. Começando com um HTML com a lista de contratados e suas fotos:

```
<h1>Reforços da Plataformatec em 2012</h1>

<ul>
  <li>
    <img src='pics/galdino.png' alt='Carlos Galdino'>
    <span>Carlos Galdino</span>
  </li>
  <li>
    <img src='pics/erich.png' alt='Erich Kist'>
    <span>Erich Kist</span>
  </li>
  <li>
    <img src='pics/lucas.png' alt='Lucas Mazza'>
    <span>Lucas Mazza</span>
  </li>
  <li>
    <img src='pics/rondy.png' alt='Rondy Sousa'>
    <span>Rondy Sousa</span>
  </li>
</ul>
```

E um pouco de CSS para alinhar as fotos e realçar os nomes:

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

li {
  float: left;
  margin-left: 10px;
  position: relative;
}

span {
  color: #153755;
  display: block;
  font-weight: bold;
  margin-top: 5px;
```

```
  text-align: center;
}
```

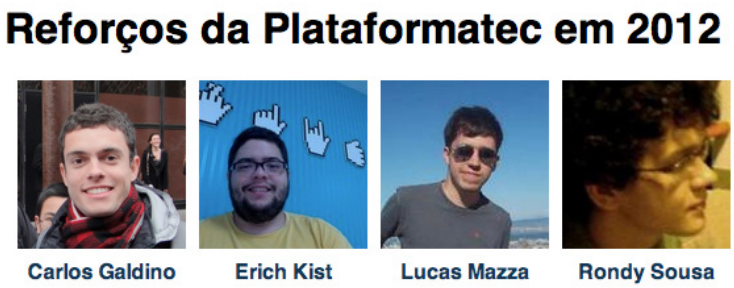


Figura 8.1: A galeria com as 4 fotos no lugar.

Para dar uma descontraída na galeria, vamos girar as fotos utilizando a função `rotate`. Sendo que as fotos ímpares (do Carlos e a minha) irão para um lado e as pares (do Erich e do Rondy) para outro. Utilizando o seletor de `nth-child`, o código para isso fica bem enxuto:

```
li:nth-child(even) {
  transform: rotate(5deg);
}

li:nth-child(odd) {
  transform: rotate(-5deg);
}
```

Figura 8.2: A função 'rotate' em ação, aplicada nas fotos.

Passando um valor negativo podemos rotacionar as imagens no sentido anti-horário, enquanto valores positivos são para o sentido horário. Outro efeito que conseguimos adicionar facilmente é ao passar com o cursor por cima das fotos, aumentar o tamanho da imagem e do texto, através do uso do `scale`.

```
li:hover {
  transform: scale(1.3);
  z-index: 1;
}
```

Figura 8.3: Combinando o 'hover' com o 'scale'.

Desta forma, a `img` e o `span` irão voltar a posição normal e ficarão 30% maiores - e o uso do `z-index` vai posicionar a foto em cima das demais. Podemos testar

outras funções como o `skew` e o `translate` aqui, mas elas não são aplicações tão interessantes quanto o `rotate` e o `scale` no nosso cenário. Mas vamos conferir como elas ficariam na nossa galeria.

```
li {
  transform: skew(10deg, 10deg);
}
```

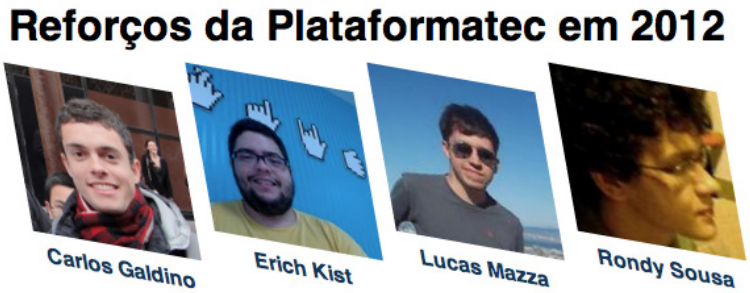


Figura 8.4: Talvez o 'skew' não seja o mais interessante para o nosso caso.

```
li:nth-child(even) {
  transform: translateY(10px);
}

li:nth-child(odd) {
  transform: translateY(-10px);
}
```

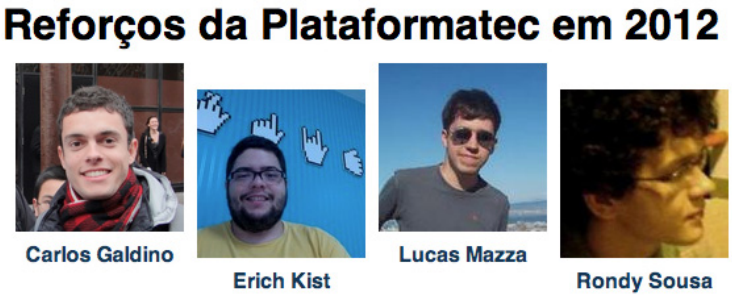


Figura 8.5: Movimente os elementos com a função 'translate'.

Em cada caso é necessário estudar qual o melhor efeito a ser usado. O `skew` pode ser útil para desenhar cubos em 3D utilizando 3 elementos, e o `translate` é uma ótima alternativa ao uso de `position: absolute` junto de `top` e `left`.

Em navegadores sem suporte a `transform`, a galeria seria exibida da forma que ela foi criada inicialmente - sem efeitos nenhum. Caso você precise trabalhar com navegadores sem suporte, sempre teste as suas páginas sem os efeitos de `transform` para confirmar que elas continuam funcionais.

8.3 Transições de estilos

A propriedade `transition` permite você "animar" a mudança - ou seja, a transição - entre outras propriedades dos seus elementos quando elas mudam de valor. Seja esta mudança feita através de JavaScript, alterando as classes aplicadas a um elemento, ou pelas mudanças causadas por pseudo seletores, como `:focus` e `:hover`.

Transições são uma ótima adição a interfaces já existentes, já que dependem da mudança de estado existente, como *containers* deslizantes e modais que aparecem e desaparecem das telas. Se você está acostumado a usar animações em JavaScript com o jQuery, por exemplo, é possível substituir o uso de funções como `slideDown` e `fadeToggle` por transições entre classes específicas para modificar a altura ou a opacidade, tornando esses efeitos mais rápidos e garantido melhor qualidade aos efeitos.

Vamos voltar a esse assunto do JavaScript mais tarde, mas primeiro vamos ver as transições em ação para melhorar a experiência de interfaces que já desenvolvemos anteriormente.

8.4 Transições na galeria de fotos

Você se lembra da galeria de fotos dos reforços da Plataformatec que criamos anteriormente para usar a propriedade `transform` com a função `rotate`? Vamos aprimorar o efeito de `scale` adicionado ao `:hover` das fotos com uma transição - e isso será absurdamente simples - basta aplicar a seguinte linha ao seletor da `li`:

```
li {
  transition: transform 0.2s linear;
}
```

Abra a galeria no seu navegador e passe o mouse pelas fotos novamente. Incrível, certo? Apenas uma linha de CSS e ganhamos essa animação - mas essa linha contém bastante informação sobre o que queremos fazer. A propriedade `transform` é um atalho para as 4 propriedades que compõe a animação, da mesma forma como usamos `margin` para definir 4 margens diferentes em uma propriedade só, sendo elas:

1) **transition-property**: para qual propriedade a transição será aplicada. É possível utilizar o valor `all` para referenciar todas as propriedades do elemento;

2) **transition-duration**: A duração que a transição deve ter, em segundos;

3) **transition-timing-function**: Como a velocidade da transição deverá ser calculada, os valores mais comuns são `linear`, `ease`, `ease-in`, `ease-out` e `ease-in-out`;

4) **transition-delay**: tempo de espera que precede a transição - geralmente omitimos essa propriedade para utilizar o valor padrão dela, 0.

Com isso explicado, fica fácil entender o que a nossa definição faz: aplicamos a transição a propriedade `transform`, com `0.2s` de duração e a transição deverá ser executada em uma velocidade `linear`. Apesar de existir um mar de opções possíveis que conseguimos combinar com essas propriedades, é fácil definir um padrão de valores para a duração e a velocidade e repetir os valores em diversos lugares e projetos - melhor do que se preocupar constantemente em otimizar as transições para cada uso possível.

Revisitando um velho amigo

Vamos voltar ao formulário de contato criado no capítulo anterior - onde as mensagens de ajuda são exibidas conforme os campos recebem foco - e adicionar transições as mensagens. Diferente do exemplo anterior, precisamos fazer alguns ajustes ao estilo do formulário e podemos combinar transições diferentes para criar um efeito melhor.

Figura 8.6: O formulário de contato criado no capítulo anterior.

Primeiramente, precisamos trocar a definição de `display: none;` das mensagens - **não podemos usar essa propriedade nas nossas animações, pois elementos que estão escondidos desta forma não recebem os efeitos de transição quando mudamos o valor dela para** `block`. Para contornar isso, precisamos esconder as mensagens de outra maneira, utilizando a propriedade `opacity`. Então, o CSS da classe `.hint` ficará assim:

```
.hint {
  background-color: #FFFBE4;
  border-radius: 3px;
  border: 1px solid #CCC;
  box-shadow: 1px 1px 3px rgba(0,0,0,0.2);
  display: inline-block; /* Colocamos o `inline-block` de volta. */
  font-size: 0.8em;
  margin-left: 20px;
  opacity: 0; /* Remove a opacidade do elemento para escondê-lo. */
```

```
  padding: 3px;
}
```

No seletor para as mensagens que são exibidas no estado de `:focus` dos campos, vamos alterar a opacidade do elemento para exibir ele corretamente:

```
input:focus + .hint, textarea:focus + .hint {
  opacity: 1;
}
```

Alteramos a forma que escondemos e exibimos a mensagem, e agora é necessário adicionar a definição da transição a ser usada.

```
.hint {
  transition: opacity 0.5s;
}
```

Figura 8.7: A transição de opacidade sendo aplicada a mensagem.

Uma transição de meio segundo, linear, na propriedade `opacity`. Teste novamente a exibição das mensagens e veja a transição em ação. Mas ainda podemos incrementar a transição, adicionando a margem lateral na jogada. Vamos mover a definição de `margin-left` da classe `.hint` para o seletor de `:focus`, deixando o CSS assim:

```
input:focus + .hint, textarea:focus + .hint {
  margin-left: 20px;
  opacity: 1;
}
```

Agora, além do `opacity`, a propriedade `margin-left` também sofrerá mudanças, e precisamos adicionar uma definição de `transition` para ela:

```
.hint {
  transition: opacity 0.5s, margin-left 0.5s;
}
```

Ou, utilizando o atalho do `all`, definindo apenas uma propriedade e uma duração.

```
.hint {
  transition: all 0.5s;
}
```

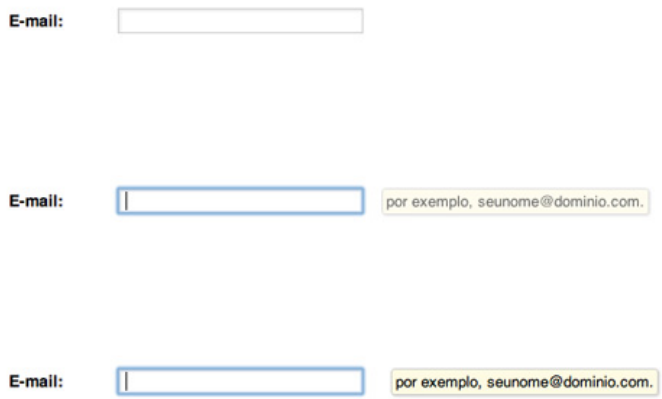


Figura 8.8: Ambas propriedades sendo 'animadas' pela transição criada.

Testando novamente o formulário, podemos ver a nova versão da nossa transição, cuja intenção é demonstrar que a mensagem está "entrando" na interface do usuário, e saindo quando outro campo recebe foco. Apesar das mudanças necessárias no código escrito anteriormente, conseguimos adicionar as transições no formulário sem perder o comportamento antigo.

8.5 Um detalhe importante sobre transições e JavaScript

IT'S DANGEROUS TO GO
ALONE! TAKE THIS.

Se você pretende substituir as animações via JavaScript por transições utilizando CSS, preciso te falar duas coisas: Primeira, essa é uma ótima ideia, e segunda, você precisa aprender mais um truque para ter controle sobre a sequência de coisas que são executadas. Via JavaScript, geralmente, podemos definir uma função de `callback` a ser executada quando a animação terminar, podendo encadear animações, mudar o conteúdo de elementos e mais - mas e como fazer isso quando se usa transições?

Para isso, existe um novo evento disponível via JavaScript chamado `transitionend`. Ele será disparado quando a transição de uma propriedade foi finalizada. Assim como no CSS, onde precisamos utilizar um prefixo específico como `-moz-`, `-webkit-` e `-o`, cada navegador, no momento, possui um nome específico para o evento de `transitionend`: Para o Safari e o Google Chrome, baseados no WebKit, é preciso usar o nome `webkitTransitionEnd`, para o Opera `oTransitionEnd` e para o Internet Explorer 10 o nome a ser utilizado é `MSTransitionEnd`. As últimas versões do Mozilla Firefox já utilizam o nome `transitionend`, apesar da necessidade de se usar o prefixo no CSS.

Um exemplo rápido disso: Digamos que você quer executar um código específico após remover a opacidade do seu elemento `#popup`. Um código que provavelmente seria escrito usando as funções do jQuery assim:

```
$('#popup').fadeOut(function() {
  // o '#popup' já está com opacidade 0;
});
```

Pode ser escrito assim, utilizando o `transitionend` em um navegador baseado no WebKit, e uma classe `.hidden` que altera a opacidade:

```
$('#popup').on('webkitTransitionEnd', function() {
  // A transição do '#popup' foi finalizada.
})

$('#popup').addClass('hidden');
```

8.6 Transformações em 3D

As funções de `transform` em 2D são interessantes, mas não se comparam aos efeitos criados pelas funções de três dimensões. Capazes de usar todo o poder de renderização dos navegadores, com essas funções podemos criar efeitos similares ao de aplicações nativas do Windows ou do OS X.

Algumas das novas funções disponíveis:

1) **translateZ**, que assim como seus similares `translateX` e `translateY`, permite a reposição dos elementos dentro de um eixo específico;

2) **rotateX**, **rotateY** podem ser usadas para rotacionar os elementos nos eixos verticais e horizontais;

3) **perspective**, que deixa possível manipular a perspectiva dos elementos em meio as transformações;

4) **matrix3d**, **translate3d** e **rotate3d**, versões melhoradas de funções que vimos anteriormente, agora com suporte a 3D.

Além das funções novas, uma nova propriedade foi adicionada para manipular o efeito da transformação quando aplicada a elementos com outros elementos dentro dele, o `transform-style`. Por padrão, os elementos filhos não compartilham do efeito 3D existente no elemento que os contêm - o que leva a um comportamento inesperado do posicionamento - e podemos mudar isso utilizando o `transform-style` com o valor `preserve-3d`, aplicando os efeitos aos elementos internos do que está sendo transformado.

Apesar do suporte escasso de navegadores, existem cenários onde utilizar efeitos em 3D são uma ótima ferramenta para melhorar a interação dos seus projetos e roubar elogios pela beleza dos efeitos. Páginas voltadas para dispositivos móveis ou cuja audiência majoritária utiliza navegadores atualizados - como o Firefox e o Chrome - são ótimas pedidas. Aconselho isso, poia alternativas aos efeitos para navegadores sem suporte provavelmente vão requerer mudanças maiores na sua interface para permitir o uso em casos onde não há suporte as funções de 3D.

8.7 Girar formulários com apenas um clique

Vamos transitar entre dois formulários utilizando animações 3D - geralmente vemos isso sendo feito com efeitos mais simples, mas o trabalho extra para chegar no que

vamos criar não é tão grande quanto se pode imaginar. Vamos começar com o código para os formulários.

Primeiro, um formulário de cadastro tradicional:

```
<section class='signup'>
  <form>
    <h2>Cadastre-se agora!</h2>
    <p>
      <label for="account_email">E-mail</label>
      <input type='email' id="account_email">
    </p>
    <p>
      <label for="account_password">Senha</label>
      <input type='password' id="account_password">
    </p>
    <p>
      <label for="account_password_confirmation">
        Confirme a sua senha
      </label>
      <input type='password' id="account_password_confirmation">
    </p>
    <p><button>Enviar!</button></p>
    <a href='javascript:;' class='toggle'>Já estou cadastrado no site</a>
  </form>
</section>
```

E um formulário para login:

```
<section class='signin'>
  <form>
    <h2>Entrar no site</h2>
    <p>
      <label for="user_email">E-mail</label>
      <input type='email' id="user_email">
    </p>
    <p>
      <label for="user_password">Senha</label>
      <input type='password' id="user_password">
    </p>
    <p>
      <button>Enviar!</button>
    </p>
```

```
    <a href='javascript:;' class='toggle'>Quero me cadastrar</a>
  </form>
</section>
```

Junto deles, um pouco de CSS (devidamente inspirado em exemplos anteriores) para adicionar estilo aos campos e posicionar todos os elementos:

```
form {
  font-size: 0.8em;
  padding: 10px;
}

label {
  display: block;
  font-weight: bold;
}

input {
  border: 1px solid #CCC;
  box-shadow: inset 2px 2px 2px #EEE;
  font-size: 0.9em;
  padding: 2px 5px;
}
```

Figura 8.9: O formulário de cadastro e o de login.

Nada muito diferente do que nos já vimos anteriormente, certo? Os links de "Quero me cadastrar" e `Já estou cadastrado no site` serão utilizados para transitar entre os formulários, com uma leve ajuda de JavaScript e do jQuery. Agora, vamos colocar os nossos formulários dentro de 2 elementos - pode soar que isto é desnecessário, mas eles terão um papel importante a cumprir.

```
<div class='container'>
  <div class='card'>
  <!-- As seções de 'login' e 'signup' vão aqui... -->
  </div>
</div>
```

E claro, um pouco de CSS para esses elementos.

```
.container {
  height: 300px;
```

```
  position: relative;
  width: 200px;
}

.card {
  border-radius: 5px;
  border: 1px solid #CCC;
  height: 100%;
  position: absolute;
  width: 100%;
}

.card section {
  background-color: #FFF;
  border-radius: 5px;
  position: absolute;
}
```

Figura 8.10: As seções devidamente posicionadas uma em cima da outra.

Definimos o elemento que vai conter as seções e seus formulários, e posicionados exatamente uma em cima da outra. Para poder "girar" os formulários de um

lado para outro, precisamos colocar um deles de costas, certo? Então, vamos usar o `rotateY`!

```
.signup {
  transform: rotateY(180deg);
}
```

Figura 8.11: Podemos ver o formulário de cadastro invertido ao fundo.

Feito! Mas o formulário aparece invertido no fundo, o que pode deixar as coisas bastante estranhas. Então precisamos esconder as "costas" do formulário, atualizado um dos seletores que criamos anteriormente:

```
.card section {
  backface-visibility: hidden;
  background-color: #FFF;
  border-radius: 5px;
  position: absolute;
}
```

Figura 8.12: Com o 'backface-visibility' em ação, a seção de cadastro não aparece.

Com a propriedade `backface-visibility` definida como `hidden`, as costas dos elementos (no nosso caso, das seções `.signup` e `.signin`) ficaram invisíveis. O que precisamos agora é mudar de um formulário para outro, utilizando um pedaço de código JavaScript, utilizando o jQuery:

```
jQuery(function($) {
  $('section').on('click', 'a', function(e) {
    $('.card').toggleClass('flipped');
  })
});
```

Toda vez que algum link dentro de uma `section` for clicado, um dos *containers* que adicionados anteriormente - a `div .card` - irá receber a classe `flipped`, ou irá perder essa classe caso ela esteja presente. Mas não adicionamos o CSS específico dessa classe - que será bem simples:

```
.card.flipped {
  transform: rotateY(180deg);
}
```

Assim, todo o elemento será rotacionado por inteiro - então o formulário que estava de costas ficará de frente, e o que estava na frente ficará para trás. Se você testar

isso e tentar voltar para o formulário de login, verá que não conseguimos clicar nos elementos do formulário de cadastro. Isso é porquê a `div .card` não está no mesmo espaço das seções dos formulários - o caso do `transform-style` que comentei anteriormente. Basta corrigir isso e podemos transitar de um formulário para outro a vontade.

```
.card {
  border-radius: 5px;
  border: 1px solid #CCC;
  height: 100%;
  position: absolute;
  transform-style: preserve-3d;
  width: 100%;
}
```

Figura 8.13: Agora conseguimos chegar no formulário de cadastro.

Pronto - conseguimos mudar de um formulário a outro, mas não temos animação nenhuma - hora de uma transição entrar em jogo, para o elemento `.card`. Uma transição de um segundo é o suficiente para o nosso exemplo.

```
.card {
  border-radius: 5px;
```

```
  border: 1px solid #CCC;
  height: 100%;
  position: absolute;
  transform-style: preserve-3d;
  transition: transform 1s;
  width: 100%;
}
```

Agora estamos chegando a algo interessante, mas ainda não ajustamos um ponto importante do uso de 3D - a perspectiva. Não existe aquela ideia de profundidade e que os elementos estão girando em apenas um eixo dentro do espaço definido. Para determinar a "profundidade" das nossas páginas, precisamos utilizar a propriedade `perspective`. Como os elementos que estão sendo utilizados estão relativos ao nosso `.container`, é no CSS nele que devemos definir a profundidade do nosso plano.

```
.container {
  height: 300px;
  perspective: 500px;
  position: relative;
  width: 200px;
}
```

Figura 8.14: A animação final, com o efeito de perspectiva.

`500px` é um valor de praxe para esse cenário. É fácil notar a diferença que isso fez para o efeito que criamos - conseguimos notar os formulários girando na ilusão de profundidade que criamos via CSS. Se quiser mudar o sentido da animação, basta trocar o uso do `rotateY` por `rotateX`, e os formulários irão girar na horizontal.

8.8 Utilizando animações

Agora que já vimos sobre `transform` e `transition` e conseguimos criar alguns efeitos interessantes com eles, por quê se preocupar em ver sobre a propriedade `animation`? A propriedade `animation` consegue ir além das transformações e transições e permite um controle maior sobre a execução das animações, repetições e até adiar os efeitos por alguns segundos. Além de diversas opções de configuração, como o `animation-delay` para um tempo de espera antes da animação, o `animation-play-state` para definir se a animação está sendo executada ou não e o `animation-iteration-count` para definir quantas vezes ela deve ser executada, podemos definir diversos pontos de quebra nos efeitos que queremos fazer, e não apenas o início e fim como fazemos com as transições. O que permite toda essa flexibilidade é a definição dessas regras utilizando `@keyframes`, onde definimos propriedades em pontos específicos das animações, utilizando porcentagens. Vamos colocar a mão na massa e entender melhor o uso dos `@keyframes`.

8.9 Começando com keyframes

Nossa primeira animação será fazer uma entrada digna de cinema para um título da sua página. Para isso só vamos precisar de um `h1`.

```
<h1>Bem vindo!</h1>
```

E claro, jogar um pouco de CSS em cima para acompanhar.

```
h1 {
  color: #000;
  font-size: 1.4em;
  text-align: center;
}
```

Um começo bastante simples, agora vamos adicionar a animação, que irá trabalhar a opacidade e o tamanho do elemento, utilizando `transform`.

```
h1 {
  animation: appear 2s 0 linear;
  color: #000;
  font-size: 1.4em;
  text-align: center;
}
```

Assim como o `transform` e o `transition` são utilizados como atalhos para definir várias opções de uma vez só, o `animation` pode compor diversos valores. No nosso caso, estamos utilizando a animação chamada **appear** (`animation-name`), com 2 segundos de duração (`animation-duration`), a ser executada imediatamente (com o `animation-delay` em 0) e com um progresso linear (`animation-timing-function`). Mas, em nenhum momento definimos do que é composta essa animação. Precisamos definir um grupo de regras para ela utilizando uma seção de `@keyframes`, assim:

```
@keyframes appear {
  0% {
    opacity: 0;
    transform: scale(2);
  }
  70% {
    opacity: 1;
    transform: scale(1);
  }
  100% {
    transform: scale(1.2);
  }
}
```

Dentro da seção de `@keyframes`, é possível definir pedaços de CSS que serão aplicados em um ponto específico da animação fazendo referência a porcentagem do progresso - já que a duração em si é definida em outro ponto. A transição de um ponto e outro será feita da mesma forma que as transições são executadas, utilizando a função de progresso do `animation-timing-function`.

Aqui, estamos começando a animação com o elemento com seu tamanho dobrado (usando o `scale`) e sem opacidade nenhuma, aos 70% da animação (com a nossa animação de 2 segundos, isso seria aos 1.4s) o elemento deve ter a sua opacidade normalizada e o tamanho regular, e a animação será finalizada (ao completar os 2 segundos) com o elemento em uma escala de 1.2. Após isso, qualquer mudança presente na animação irá desaparecer e o elemento voltará ao normal. Para isso, vamos levar a definição de `scale(1.2)` para o estilo do `h1` e adicionar um `transform-origin` para centralizar o efeito de escala.

```
h1 {
  animation: appear 2s 0 linear;
```

```
  color: #000;
  font-size: 1.4em;
  text-align: center;
  transform: scale(1.2);
  transform-origin: 50% 50%;
}
```

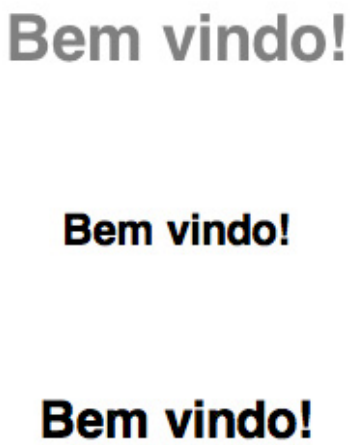


Figura 8.15: Os 3 momentos da animação do nosso 'h1'.

Com o controle sobre diversos pontos das animações e das diversas opções de configuração, conseguimos criar efeitos que não possível apenas com `transition` e `transform`, apesar da necessidade de escrever um número maior de linhas dependendo do resultado desejado.

Animações infinitas

Você provavelmente já viu o universo de gifs animados que são utilizados para informar quando algum processamento está feito por debaixo dos panos e que devemos esperar a página carregar algo novo em algum momento - os chamados **loaders**. Eu fico muito feliz em saber que não precisamos mais de imagens assim e podemos utilizar apenas CSS (e talvez um pouco de JavaScript), para replicar esse componente e conseguir mudar tamanhos e cores com apenas algumas linhas de código. Caso você não acredite nisso, vamos criar uma versão simples de um desses componentes.

Primeiro, precisamos apenas de um span, já que o nosso exemplo será bastante minimalista.

```
<span class='loading' title='Carregando...'>Carregando...</span>
```

```
.loading {
  background-color: #666;
  border-radius: 50%;
  font: 0/0 serif;
  position: absolute;
}
```

Definimos um fundo escuro, bordas redondas o suficiente para garantir que o elemento apareça como um círculo, e uma fonte zerada para esconder o texto do span - Mas e o tamanho? E os efeitos? Tudo isso irá na nossa animação, com as regras abaixo:

```
@keyframes pulse {
  0% {
    width: 0;
    height: 0;
    opacity: 1;
    top: 50px;
    left: 50px;
  }
  100% {
    width: 30px;
    height: 30px;
    opacity: 0;
    top: 35px;
    left: 35px;
  }
}
```

Enquanto a nossa esfera vai aumentando de tamanho - de 0 a 35 px - ela vai sendo apagada pela mudança de opacidade. As mudanças de top e left garantem que ela ficará centralizada no seu lugar e não irá se mover por ai devido a mudança de tamanho. Para utilizar essa animação **pulse** no nosso loader com o efeito desejado, precisamos definir que ele será animado *para sempre*.

Sim, *para sempre*. Assim:

```
.loading {
  animation: pulse 1.5s infinite;
  background-color: #666;
  border-radius: 50%;
  font: 0/0 serif;
  position: absolute;
}
```

Figura 8.16: A animação infinita do nosso 'loader'.

Podemos usar `infinite` como um valor para o `animation-iteration-count`, e a animação será executada para sempre - podemos apenas adicionar um pouco de JavaScript para esconder e exibir o loader quando for necessário. Caso seja necessário mudar a cor ou aumentar o tamanho para adequar a interfaces diferentes, só precisamos alterar algumas linhas de CSS, sem precisar criar imagens novas em um editor e executar mais requisições HTTP para servir as imagens.

Capítulo 9

O universo fora dos desktops e notebooks

A popularização e evolução dos *smartphones* e *tablets* nos últimos 4 anos teve um grande impacto nas tecnologias e na forma em que desenvolvemos os nossos projetos para a web.

Garantir uma experiência adequada para o número crescente de usuários que utilizam esses dispositivos se tornou uma vantagem comercial e um diferencial para produtos, empresas e desenvolvedores - ninguém deseja excluir um grupo de usuários devido ao tipo de equipamento que eles utilizam para acessar a web.

Para entrar neste mundo e abraçar a onda do *mobile*, é preciso uma evolução tanto técnica - mudanças de ferramentas e tecnologias utilizadas - quanto mental - o seu processo de desenvolvimento precisa se adequar a este cenário.

9.1 O que é "Responsive Web Design" e porquê você deve se preocupar

Ethan Marcotte criou o termo **Responsive Web Design** em um artigo escrito no site *A List Apart*[5], que então se tornou um livro, que é referência no assunto. A combinação de elementos fluídos, com dimensões relativas ao espaço disponível no aparelho do usuário, e regras específicas para definir estilos específicos para tamanhos de tela diferentes e funcionalidades diferentes (como, por exemplo, a resolução fantástica do iPhone 4 e do iPad de 3ª geração).

Sites responsivos adequam os seus elementos e o seu comportamento para cenários, como disponibilizando fluxo de navegação que se adequará ao tamanho da tela utilizada ou mudando as interações do usuário caso ele use um dispositivo sensível a toque ou não. Além de precisar identificar o tamanho, capacidade e funcionalidades disponíveis, é importante que a disposição dos elementos seja flexível o bastante para se adequar a qualquer dispositivo.

O primeiro passo para se começar a trabalhar com designs responsivos é fazer a transição de elementos fixos - de dimensões fixadas em `px` - para elementos fluídos - que se baseiam em porcentagens e `em`. Desta forma, a sua página consegue se expandir ou diminuir de acordo com a largura disponível, em vez de se fixar em algo similar a `960px` de largura máxima.

A conversão de tamanhos fixos para fluídos é feita com base em uma fórmula bastante simples: `"tamanho fixo"/ "contexto"= "tamanho fluído"`. Com ela você consegue converter dimensões em `px` para porcentagens (para larguras) ou `em` (para fontes), assim:

```
body {
  /* Utilizando o tamanho de fonte padrão, geralmente 16px */
  font: normal 100% Helvetica, Arial, sans-serif;
}

h1 {
  font-size: 1.5em; /* 24px / 16px = 1.5em */
}
```

A princípio, esta definição terá o mesmo efeito que definir o tamanho da fonte do `h1` como `24px`. Porém, em dispositivos que o tamanho padrão de fonte seja diferente, como em *smartphones*, `font-size` do elemento irá acompanhar esta mudança de

tamanho: em um dispositivo que utilize 12px a fonte do elemento terá 18px de altura, e não 24px.

Esta mesma abordagem pode ser usada para definir as regras do seu grid - tornando-o um grid fluído - e o espaçamento entre elementos. Desta forma você consegue fazer os seus layouts acompanharem as diferenças de largura entre diversos dispositivos, e então você pode tratar as peculiaridades de cada contexto utilizando *media queries*.

9.2 O funcionamento dos media queries

Outra ferramenta do *Responsive Web Design* são os *media queries* - uma regra específica para aplicar um bloco de CSS caso a regra seja atendida. Similar ao uso de CSS para estilos específicos de impressão (seção 4.8), você também pode indicar regras associadas ao tamanho, orientação ou a resolução de tela.

```
@media only screen
and (min-device-width : 320px)
and (max-device-width : 480px) {
/*
  Esta regra será aplicada em
  aparelhos com uma largura de 320px a 480px,
  o que atende a maioria de smartphones.
*/
}

@media only screen
and (min-device-width : 768px)
and (max-device-width : 1024px)
and (orientation : landscape) {
/*
  Além de verificar a largura do dispositivo,
  você pode conferir a orientação, utilizando
  "landscape" ou "portrait".
*/
}
```

Utilizando *media queries*, você pode definir estilos específicos para tamanhos diferentes, o que é melhor do que focar em um tipo de dispositivo específico. Diversos modelos diferentes de *smartphones* podem ser atendidos por um mesmo *media query*, o que também ajuda a atender novos modelos que irão surgir no futuro.

Da mesma forma que um CSS de impressão, você pode sobrescrever definições globais de CSS para adequar a sua página para tamanhos diferentes. Assim você consegue reordenar elementos, reduzir o tamanho de imagens ou exibir elementos alternativos para melhorar a experiência de uso dos seus usuários.

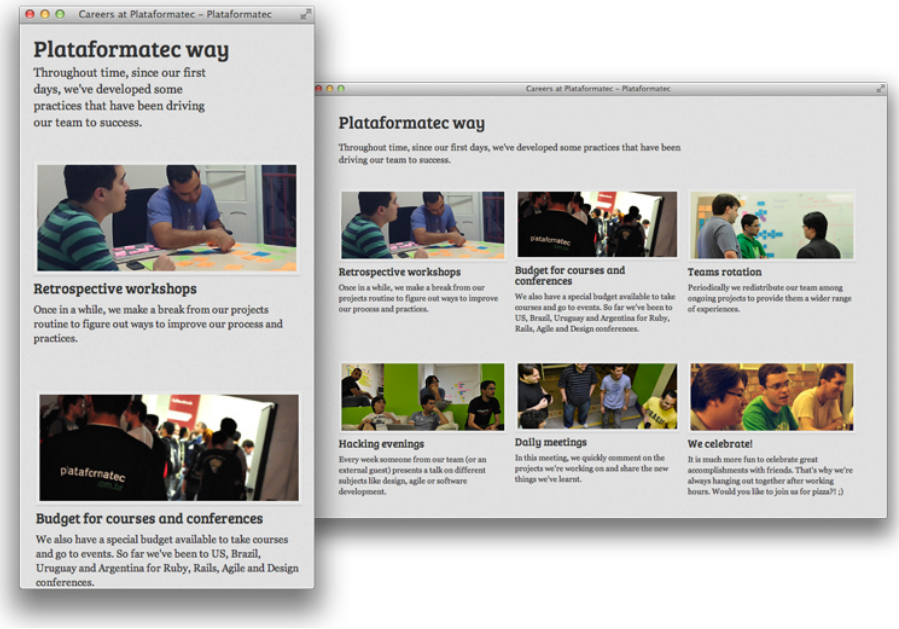


Figura 9.1: Um exemplo do site da Plataformatec (http://plataformatec.com.br): a seção de fotos muda de três colunas para apenas uma coluna quando a largura da página é reduzida.

O site *Media Queries* (http://mediaqueri.es/) possui uma coletânea de sites que possuem diversos formatos para *smartphones* e *tablets*, uma ótima referência para se inspirar com outros trabalhos.

9.3 Não é uma questão de aparelhos

Você pode considerar que toda esta atenção para adequar os seus projetos para diversos dispositivos como o último iPad ou os *smartphones* do Google, mas esta filosofia se aplica para o mundo de *notebooks* e *desktops* que estamos tão acostumados a ver.

Com diversas resoluções diferentes por aí, indo de 11" até 27", e a inclusão do

Retina display na terceira geração da linha do MacBook Pro, é possível refinar a experiência de uso para tirar proveito da abundância de pixels ou da qualidade da resolução do usuário. Basta possuir um grupo de estilos para larguras acima de 1824px, por exemplo:

```
@media only screen and (min-width : 1824px) {
/*
  Aqui você pode adicionar estilos para se aproveitar
  de uma tela maior: aumentando fontes, ícones, e
  largura dos seus elementos
*/
}
```

9.4 Por um futuro melhor

A mudança da web para estes diversos dispositivos e formas de uso só está começando: o futuro ainda guarda um universo de aparelhos novos, tecnologias e mudanças na forma em que nos conectamos pela web, e a indústria da internet ainda precisa evoluir para acompanhar estas mudanças.

Nós não podemos prever o que virá por aí, mas podemos nos preparar para seja lá o que o futuro nos reserva.

Em 2011 foi criado o *Future Friendy*[4], um manifesto assinado por diversas figuras carimbadas do mundo do desenvolvimento web. Lá existem 3 princípios que podemos seguir:

- Você deve compreender e abraçar a imprevisibilidade;
- Pense e comporte-se de uma forma amigável para o futuro;
- Ajude os outros a fazer o mesmo.

O manifesto indica que se deve focar no conteúdo que está em suas mãos - o que realmente importa para os seus usuários e os seus clientes - buscando servir ele da forma mais flexível possível, o que facilita a inclusão de novos formatos e aparelhos que surgirem com o passar do tempo, sem deixar de tirar proveito de aspectos específicos para prover uma experiência adequada.

Todos queremos que o nosso trabalho dure por anos, e para alcançar isso, precisamos trabalhar para que as nossas criações sobrevivam às mudanças de tecnologia. Imagine ter que reescrever sites inteiros daqui a 2 ou 3 anos porque a versão atual não é adequada para o cenário atual de dispositivos e casos de uso?

Capítulo 10

Ferramentas - Frameworks, Plugins e Pré-processadores

Até agora, vimos vários recursos interessantes de CSS 3 e HTML5 para que possamos criar sites e aplicações com visuais elegantes. Porém, muitas das vezes, precisamos criar os estilos do zero, pensar nas cores dos botões, nas divisões de colunas do página, no estilo das tabelas ou até mesmo coisas que podem ser mais trabalhosas, como cuidar para que a página funcione em browsers mais antigos, que não possuam os novos recursos de CSS 3 e HTML5.

Para nos ajudar nessas situações, temos disponível diversas ferramentas que nos auxiliam nesse trabalho, como o Twitter Bootstrap, HTML5 Boilerplate, -prefix-free entre outras. Saber o que cada uma nos oferece é muito importante, assim podemos escolher qual ferramenta melhor se encaixa com o projeto que está sendo desenvolvido.

10.1 Twitter Bootstrap

Um dos projetos de maior sucesso no GitHub, o Twitter Bootstrap é um toolkit criado por designers do Twitter e adotado por diversas empresas e equipes para acelerar o início de projetos por já incluir diversos componentes e facilidades de CSS, HTML e JavaScript. O projeto inclui um grid fluído (que você deve ter lido a respeito na seção 6.7), diversos botões e campos de formulários, abas de navegação, plugins em JavaScript e muito mais.

O projeto é desenvolvido usando LESS, um pré-processador de CSS que facilita a escrita de todo o CSS do Bootstrap, mas você pode baixar o arquivo CSS final e utilizar ele em qualquer de seus projetos.

Você pode usar o estilo padrão do Twitter Bootstrap sem precisar escrever nenhuma linha de CSS - apenas utilizando o HTML e as classes que a documentação do projeto indica. Depois disso, basta sobrescrever o que for necessário para adequar o visual às necessidades do seu projeto, como cores, tamanhos, bordas etc.

O Bootstrap é uma ótima pedida para desenvolvedores que não possuem tanto conhecimento (ou tempo para dedicar) em design e CSS. Se o quesito originalidade for importante para você, dedique um tempo para personalizar o visual do Bootstrap para o seu projeto. Toda a documentação e exemplos podem ser encontrados no site http://twitter.github.com/bootstrap/.

Com o sucesso do projeto, algumas alternativas surgiram com o tempo, disponibilizando soluções similares para os mesmos casos de uso do Bootstrap:

- Foundation, feito pela ZURB - Toolkit similar ao Bootstrap criado por uma empresa com mais de 10 anos de experiência no ramo de web design - http://foundation.zurb.com/.
- HTML Kickstart, criado por Joshua Gatcke - http://www.99lime.com/.
- Skeleton - um toolkit minimalista, que possui um grid responsivo e alguns componentes simples, como botões e abas. - http://getskeleton.com/.

Vale a pena baixar cada um desses frameworks, testar os seus componentes, grids e funcionalidades para descobrir se algum deles se encaixa mais com seu gosto ou até mesmo se você prefere criar todo o CSS por si próprio.

10.2 HTML5 Boilerplate

Enquanto frameworks como o Bootstrap e o Foundation possuem diversos componentes como grids, botões usando gradientes, sombras e outras novidades do CSS3, o HTML5 Boilerplate, que não é exatamente um framework, segue por outro caminho: um template simples, construído em cima de boas práticas para garantir compatibilidade e performance.

O HTML5 Boilerplate usa o Normalize.css (que vimos na seção 4.1) como reset básico, classes utilitárias adicionais (para corrigir floats e substituição de imagens, por exemplo), jQuery disponibilizado pelo CDN do Google e mais. Apesar de não disponibilizar nenhum componente gráfico, ele é uma base sólida para começar um projeto novo.

CDN?

Um CDN - `Content delivery network` - é uma rede de distribuição de arquivos com o objetivo de servir conteúdo contando com alta disponibilidade e performance, com servidores estrategicamente localizados ao redor do mundo, sendo uma ótima solução para servir arquivos estáticos como imagens, scripts e vídeos.

O CDN do Google é famoso por disponibilizar as principais bibliotecas de JavaScript usadas atualmente, como o jQuery, o MooTools e o Prototype. Além disso, é possível contratar a infraestrutura de empresas como a Amazon e Akamai para servir seus próprios arquivos.

Veja o código, mais detalhes e tutoriais sobre como aproveitar o projeto ao máximo no site http://html5boilerplate.com/.

10.3 Plugins em JavaScript

As vezes apenas HTML e CSS não é o suficiente, pois no meio de incompatibilidade com navegadores e APIs novas que não foram implementadas por completo, é possível preencher essas lacunas com JavaScript ou identificar o que o podemos ou não fazer com um navegador específico usando JavaScript para analisar o *User Agent* ou testando as APIs disponíveis.

10.4 Modernizr

Modernizr é uma biblioteca que permite identificar o que é suportado e o que não é suportado no navegador dos nossos usuários e adequar a experiência do seu site para isso, seja via CSS ou JavaScript. Por exemplo, identificar o suporte a tag `<audio>` e a qual formato se deve usar, como `ogg`, `mp3` ou `m4a`. Ou, para navegadores que não suportam animações e transformações, aplicar um estilo alternativo para que o seu layout continue funcional para seus usuários.

O Modernizr permite duas abordagens: a primeira é através da adição de classes ao elemento `html`, identificando as funcionalidades disponíveis (em navegadores sem suporte a interação por toque, como os desktops, ele adiciona a classe `no-touch`) ou pela API em JavaScript no objeto `Modernizr`:

```
if(Modernizr.canvas) {
  // Ok, podemos usar Canvas aqui.
} else {
  // Canvas não está disponível neste navegador.
}
```

Para mais detalhes das funcionalidades que o Modernizr pode detectar e para baixar o código para adicionar ao seu projeto, visite http://modernizr.com/.

-prefix-free

É fato que nenhum desenvolvedor gosta de repetir o mesmo código para satisfazer 3 ou 4 prefixos diferentes toda vez que se vai usar um gradiente ou uma animação em CSS.

Para solucionar isso direto nos navegadores, Lea Verou, uma engenheira de front-end grega, desenvolveu uma biblioteca que altera o CSS da sua página, adicionando apenas os prefixos necessários. A maior vantagem disso é que você não precisa escrever as suas folhas de estilo se preocupando com qual prefixo você precisa usar para cada propriedade nova do CSS3.

Se interessou no -prefix-free? O site da biblioteca é http://leaverou.github.com/prefixfree/. Interessado ou não, não deixe de visitar o blog da Lea Verou, lotado de outras ferramentas e posts interessantes sobre JavaScript e CSS3 - http://lea.verou.me/.

10.5 Polyfills

Existem dezenas de outras bibliotecas que ajudam a preencher os buracos de funcionalidades ausentes em diversos navegadores diferentes que precisamos enfrentar, como suporte a SVG, LocalStorage, WebSockets, Geolocalização. As bibliotecas que implementam essas APIs em navegadores antigos ou que não possuem suporte receberam o nome de **Polyfills**. [6]

Na Wiki do Modernizr no GitHub existe uma lista extensa de diversas implementações de Polyfills feitas pela comunidade, https://github.com/Modernizr/Modernizr/wiki/HTML5-Cross-browser-Polyfills.

10.6 Pré-processadores

Infelizmente, CSS não possui o dinamismo e alguns recursos de linguagens de programação como Ruby, Python ou JavaScript, o que faz muitas vezes com que o nosso código fique extenso e repetitivo. Para amenizar isso, existem linguagens intermediárias que adicionam alguns recursos interessantes na linguagem e ajudam a criar mais estilos, com menos códigos. Essas linguagens intermediárias são chamadas de pré-processadores e no momento da escrita desse livro, posso recomendar três: **LESS**, **Sass** e **Stylus**.

Os três possuem muitas funcionalidades em comum, o que ajuda a transitar entre cada pré-processador para avaliar todos e escolher o seu favorito. Veja algumas das principais funcionalidades a seguir:

Seletores aninhados

Para reduzir a duplicação de seletores, os pré-processadores permitem que você aninhe os seletores de elementos contidos em outro elemento (como no caso abaixo).

```
// Aninhando os seletores, não precisamos escrever
// 'header nav' e 'header a' para alterar o estilo apenas dos elementos
// presentes dentro de 'header'.

header {
  padding: 10px 5px;

  nav {
    float: left;
  }

  a {
    color: red;
  }
}
```

Com este exemplo, conseguimos gerar o seguinte CSS:

```
header {
  padding: 10px 5px;
}
header nav {
  float: left;
```

```
}
header a {
  color: red;
}
```

Variáveis

Valores em comum, como uma cor ou um tamanho específico, que você tende a repetir com frequência na sua folha de estilo podem ser extraídos para uma variável. Assim, caso você precise mudar todos os vermelhos do seu site para roxo você não terá tanto trabalho.

```
// No LESS, você pode definir variáveis com '@nomedavariavel: valor',
// e usá-las em qualquer parte do seu código, além de poder manipular
// o seu valor com cálculos ou outras funções

@darkred: #990000;
@grid: 960px;

body {
  color: @darkred;
  width: @grid;
}

.sidebar {
  width: @grid / 3;
}

.container {
  width: @grid / 2;
}
```

Quando processamos este exemplo utilizando o LESS, as definições de variáveis geram os seguintes seletores:

```
body {
  color: #990000;
  width: 960px;
}
.sidebar {
  width: 320px;
```

```
}
.container {
  width: 480px;
}
```

Mixins

Mixins trazem muita flexibilidade a sua folha de estilo para repetir definições similares, executar cálculos ou gerar prefixos específicos sem repetir as mesmas definições diversas vezes.

```
// Os mixins do Stylus permitem que você escreva CSS sem se preocupar
// se está usando um mixin ou uma propriedade comum, como no caso abaixo
// onde o 'transform' cria todas as variações com os prefixos
// necessários.
// Caso você não precise mais dos prefixos '-moz' ou '-webkit',
// por exemplo, basta alterar o mixin e gerar o seu arquivo '.css'
// novamente.

transform() {
  -moz-transform:arguments;
  -ms-transform: arguments;
  -o-transform:arguments;
  -webkit-transform: arguments;
  transform:arguments;
}

.slider {
  transform: width 0.2s linear;
}

.popup {
  transform: opacity 0.5s;
}
```

Com esta definição de mixin, o Stylus consegue gerar o seguinte código:

```
.slider {
  -moz-transform: width 0.2s linear;
  -ms-transform: width 0.2s linear;
  -o-transform: width 0.2s linear;
  -webkit-transform: width 0.2s linear;
```

```
  transform: width 0.2s linear;
}
.popup {
  -moz-transform: opacity 0.5s;
  -ms-transform: opacity 0.5s;
  -o-transform: opacity 0.5s;
  -webkit-transform: opacity 0.5s;
  transform: opacity 0.5s;
}
```

Mas cuidado, pré-processadores não são só unicórnios e arco-íris. O uso excessivo ou inadequado das suas funcionalidades pode trazer problemas para você e a sua equipe. Muito aninhamento complica a leitura do seu código e gera seletores muito maiores do que você realmente precisa. Muitos mixins e variáveis distanciam demais o código que você escreve do código gerado, o que complica o *debug* e a resolução dos problemas que aparecerem durante o desenvolvimento.

Com o devido conhecimento de CSS, os pré-processadores podem ajudar bastante a tornar o seu código fonte mais flexível e inteligente, seja manipulando cores para facilitar a criação de gradientes, escrevendo prefixos de navegadores específicos como `-moz` e `-webkit` ou reduzindo a duplicação de código com mixins ou funções.

Para fazer a sua escolha, sugiro uma pesquisa rápida e um *test-drive* nos pré-processadores do seu interesse. Confira qual se adequa melhor as ferramentas e linguagens que você já usa no seu dia a dia. Confira os sites de cada um para começar a estudar melhor:

- LESS - http://lesscss.org/;
- Sass - http://sass-lang.com/;
- Stylus - http://learnboost.github.com/stylus/.

10.7 É tudo CSS e HTML

Os itens citados acima são frutos do trabalho de desenvolvedores como eu e você, com grande interesse em CSS e HTML. Antes de ter essas ferramentas em mãos, todos eles passaram por apertos e problemas clássicos que se encontram no dia a dia de desenvolvimento. Antes de entrar de cabeça nessa festa, lembre-se de praticar um pouco sem tais ferramentas, apenas você e seu editor favorito.

Desenvolvendo as coisas "na mão" você consegue entender melhor a razão de tais soluções terem sido criadas e quais os problemas que elas solucionam. Quem sabe, no processo, você pode criar ferramentas e soluções próprias e compartilhar elas com a comunidade.

Capítulo 11

Não pare por aqui

Abordamos diversos assuntos sobre HTML e CSS neste livro, mas o universo em torno destas tecnologias é vasto demais para se comprimir todas as referências, explicações e técnicas em um livro. E tudo está evoluindo a passos largos, o que significa que precisamos acompanhar estes assuntos e conferir as novidades que surgem para aprimorar e facilitar o trabalho de desenvolvedores como eu e você. Para isso, posso lhe indicar as fontes que eu e diversos outros profissionais utilizamos para nos manter atualizados, e acredito que alguns deles podem lhe ser bastante úteis.

Referências na internet

Uma fonte incrível de conteúdo sobre desenvolvimento web é a comunidade criada em torno do **CSS-Tricks** (http://css-tricks.com/), mantido pelo Chris Coyier, desenvolvedor e designer americano, que ajuda diversos profissionais a criar websites melhores desde 2007. Outro lugar bastante interessante para se encontrar publicações nos diversos assuntos da web é a revista digital **A List Apart** (http://www.alistapart.com/), que também possui uma editora de livros para desenvol-

vedores, chamada **A Book Apart**, e prepara uma série de eventos todo ano chamado **An Event Apart**.

Para referências rápidas e explicações enxutas sobre funcionalidades novas, tecnologias associadas a HTML5 e tudo mais, eu indico dois sites incríveis para responder perguntas como *"Quais navegadores atualmente suportam animações?"* ou *"Como eu posso simular o uso de Media Queries no Internet Explorer?"* o **When can I use...** (http://caniuse.com/) e o **HTML5 Please** (http://html5please.com/). Atualizados conforme novas especificações surgem e novas versões de navegadores são lançadas, ambos os sites possuem as referências necessárias para saber como e por onde começar a trabalhar com coisas novas.

Outro site bastante interessante para se ter como referência é o **Dive into HTML5** (http://diveintohtml5.info/). Originalmente criado por Mark Pilgrim e com diversas contribuições de outros desenvolvedores, o Dive into HTML5 é uma referência completa que vai desde as tags introduzidas no HTML5 quanto a APIs de JavaScript disponíveis e a desenho de gráficos em `canvas`. O mesmo pode se dizer do **HTML5 Doctor** (http://html5doctor.com/about/), outro site bastante completo de referências e artigos sobre o assunto.

Acompanhe quem entende do assunto

Eu acredito que o mundo da tecnologia é feito por aqueles que o compõem, e não apenas pelas empresas presentes no mercado. Por isso, eu considero muito interessante o trabalho desenvolvido por diversas pessoas pelo mundo que nos últimos anos, independente do cargo ou empresa que ela se encontrasse, compartilhando conteúdo, criando projetos e palestras sobre diversos assuntos relacionados a web e auxiliando o trabalho de muitos outros.

Sugiro que você pesquise e acompanhe os trabalhos, blogs e slides de alguns da lista de talentosos profissionais abaixo. Um canal comum entre todos eles é o Twitter, então adicionei os seus usuários no Twitter para você acompanhar o que eles andam comentando por lá:

- **Paul Irish** (@paul_irish), **Mathias Bynens** (@mathias), **Divya Manian** (@divya), **Nicolas Gallagher** (@necolas) e todos os outros envolvidos nos projetos do HTML5 Boilerplate, HTML5 Please e os outros relacionados. Você pode conferir tudo que essa galera está criando no perfil da organização no GitHub, https://github.com/h5bp.

- **Lea Verou** (@leaverou), criadora do -prefix-free, Dabblet (http://dabblet.com)

e diversas outras ferramentas relacionadas a CSS 3. Além disso, ela viaja o mundo apresentando suas palestras de diversos assuntos.

- **Ethan Marcotte** (@beep), autor do livro *Responsive Web Design* e **Luke Wroblewski** (@lukew), autor do *Mobile First* da A Book Apart, livros que se tornaram leituras obrigatórias em seus respectivos assuntos.
- **Brad Frost** (@brad_frost), outro entendido no assunto de design responsivo de usabilidade, que trabalhou em diversos projetos interessantes de aplicações para dispositivos móveis.
- **Ryan Singer** (@rjs), designer da 37signals, que possui ótimas palestras e artigos sobre o fluxo de trabalho e o desenvolvimento de interfaces para a web.
- **Eduardo Shiota** (@shiota), designer que além de palestrar em diversos eventos aqui no Brasil também ministra um curso sobre Responsive Web Design no Howtocode (http://howtocode.com.br/workshops/responsive-design).
- **Sérgio Lopes** (@sergio_caelum), que além de atuar como desenvolvedor na Caelum também é coordenador e instrutor de seus cursos de web design.
- **Kyle Neath** (@kneath), **Ben Bleikamp** (@bleikamp) e toda a equipe de design do GitHub, responsáveis por diversos projetos e interfaces sensacionais.

Outros livros

Um caminho interessante para se aprofundar em outros assuntos relacionados a HTML e CSS é mergulhar em bons livros sobre o que você quer tanto aprender. A editora **A Book Apart** possui uma ótima coleção de livros no campo de web design, abordando diversos assuntos, desde HTML e CSS (*HTML5 for Web Designers*, *CSS 3 for Web Designers* e o incrível *Responsive Web Design*), tipografia para a web (*On Web Typography*) e usabilidade e design (*Designing for Emotion*). São livros pequenos porém bastante objetivos em suas explicações e ricos em conteúdo.

Caso se interesse por JavaScript, a O'Reilly possui diversos livros sobre a linguagem e sua utilização nos dias de hoje, abordando desde assuntos básicos e introdutórios a guias avançados. Dos diversos títulos da editora, os principais na minha opinião são: *JavaScript: The Good Parts*, *JavaScript Web Applications* e *jQuery Pocket Reference*.

Índice Remissivo

-prefix-free, 185
::after, 41
::before, 41
:hover, 17
3d, 161

alt, 18
animation, 170
article, 29
aside, 29

background, 15
background-color, 15
border, 13
border-radius, 67
Box model|hyperpage, 38
box-sizing, 38

CDN, 184
clear, 99
clearfix, 106
color, 16
column, 118

data-*, 29
display, 96

figcaption, 71
figure, 71
flexbox, 119
float, 20, 97
Font Awesome, 64
font-family, 11
font-size, 12
Fontello, 64
FontSpring, 63
FontSquirrel, 63
footer, 29
Formulários, 123

Google Web Fonts, 63
grid, 117

header, 29
HTML5 Boilerplate, 183
html5shiv, 33

IcoMoon, 64
Iconic, 64
ImageOptim, 19
img, 18
impressão, 52

LESS, 186

margin, 13
Modernizr, 184

nav, 29
Normalize.css, 37

onTransitionEnd, 160
OOCSS, 50

padding, 13
Pictos, 64
placeholder, 134
Polyfills, 185
position, 107, 110

Pré-processadores, 186
preserve-3d, 168
Press Start 2P, 63
pseudo seletor, 17
pseudo-elementos, 41

reset, 35
Responsive Web Design, 176
rodapé, 22

Sass, 186
section, 29
Shifticons, 64
src, 18
Stylus, 186

transform, 150
transition, 155
Twitter Bootstrap, 182
Typekit, 63

width, 11
Wufoo, 125

z-index, 113

Referências Bibliográficas

[1] Ross Allen. Css box-shadow can slow down scrolling. http://nerds.airbnb.com/box-shadows-are-expensive-to-paint, 2011.

[2] Chris Coyier. Fighting the space between inline block elements. http://css-tricks.com/fighting-the-space-between-inline-block-elements/, 2012.

[3] Nicolas Gallagher. About normalize.css. http://nicolasgallagher.com/about-normalize-css/, 2012.

[4] Brad Frost Jeremy Keith Lyza D. Gardner Scott Jehl Stephanie Rieger Jason Grigsby Bryan Rieger Josh Clark Tim Kadlec Brian LeRoux Andrea Trasatti Luke Wroblewski, Scott Jenson. Future friendly. http://futurefriend.ly/, 2011.

[5] Ethan Marcotte. Responsive web design. http://www.alistapart.com/articles/responsive-web-design, 2010.

[6] Remy Sharp. What is a polyfill? http://remysharp.com/2010/10/08/what-is-a-polyfill/, 2010.

[7] Nicole Sullivan. The media object saves hundreds of lines of code. http://www.stubbornella.org/content/2010/06/25/the-media-object-saves-hundreds-of-lines-of-code/, 2010.