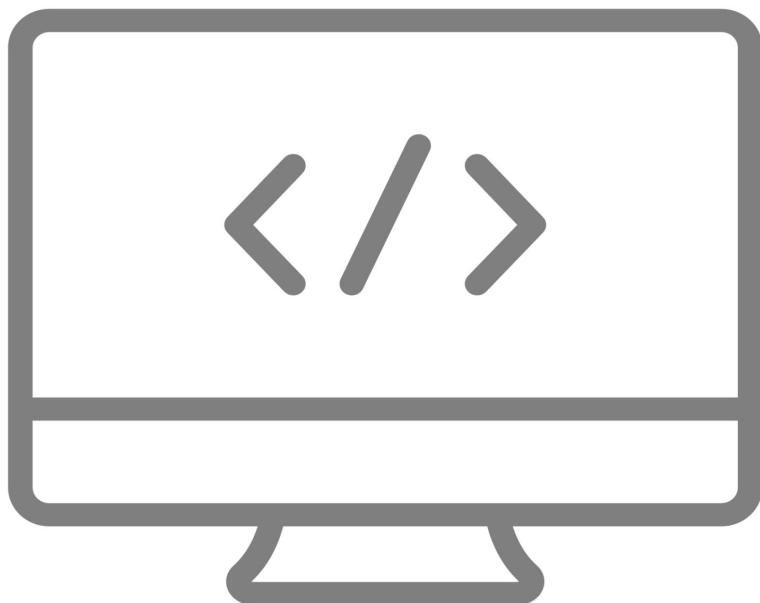


Desenvolvimento Web com HTML, CSS e JavaScript

Curso WD-43





Conheça também:



alura

alura.com.br



Casa do Código
Livros e Tecnologia

casadocodigo.com.br

Blog da Caelum



blog.caelum.com.br

Newsletter



caelum.com.br/newsletter

Facebook



facebook.com.br/caelumbr

Twitter



twitter.com/caelum

Sumário

1 Sobre o curso - O complexo mundo do front-end	1
1.1 O curso e os exercícios	1
1.2 O projeto de e-commerce	2
1.3 Tirando dúvidas com instrutor	2
1.4 Tirando dúvidas online no GUJ	2
1.5 Bibliografia	2
1.6 Para onde ir depois?	3
2 Introdução a HTML e CSS	4
2.1 Exibindo informações na Web	4
2.2 Sintaxe do HTML	7
2.3 Estrutura de um documento HTML	7
2.4 Tags HTML	9
2.5 Imagens	10
2.6 A estrutura dos arquivos de um projeto	10
2.7 Editores e IDEs	11
2.8 Primeira página	11
2.9 Exercícios: Primeiros passos com HTML	13
2.10 Estilizando com CSS	16
2.11 Sintaxe e inclusão de CSS	17
2.12 Propriedades tipográficas e fontes	19
2.13 Alinhamento e decoração de texto	20
2.14 Imagem de fundo	20
2.15 Bordas	20
2.16 Exercícios: Primeiros passos com CSS	21
2.17 Cores na Web	23
2.18 Listas HTML	24
2.19 Espaçamento, margem e dimensões	25
2.20 Exercícios: Listas e margens	27

2.21 Links HTML	28
2.22 Exercícios: Links	29
2.23 Elementos estruturais	30
2.24 CSS: Seletores de ID e filho	30
2.25 Fluxo do documento e float	31
2.26 Exercícios: Seletores CSS e flutuação de elementos	32
2.27 O futuro e presente da Web com o HTML5	34
3 HTML semântico e posicionamento no CSS	35
3.1 O processo de desenvolvimento de uma tela	35
3.2 O projeto Mirror Fashion	37
3.3 Analisando o Layout	38
3.4 HTML semântico	40
3.5 Pensando no header	41
3.6 Estilização com classes	42
3.7 Exercícios: Header semântico	44
3.8 CSS Reset	45
3.9 Block vs Inline	46
3.10 Exercícios: Reset e display	47
3.11 Position: static, relative, absolute e fixed	48
3.12 Exercícios: Posicionamento	51
3.13 Para saber mais: Suporte HTML5 no Internet Explorer antigo	51
3.14 Exercícios opcionais	52
4 Mais HTML e CSS	53
4.1 Analisando o miolo da página	53
4.2 Formulários	53
4.3 Posicionamento com float e clear	53
4.4 Decoração de texto com CSS	55
4.5 Cascata e herança no CSS	56
4.6 Para saber mais: o valor inherit	56
4.7 Exercícios: Menu e destaque	57
4.8 Display inline-block	60
4.9 Exercícios: Painéis flutuantes	61
4.10 Seletores de atributo do CSS3	63
4.11 Rodapé	65
4.12 Substituição por Imagem	65
4.13 Estilização e posicionamento do rodapé	66
4.14 Exercícios: Rodapé	67

4.15 Exercícios opcionais	69
5 CSS Avançado	70
5.1 Seletores avançados	70
5.2 Pseudo-classes	72
5.3 Pseudo elementos	75
5.4 Exercícios: Seletores, pseudo-classes e pseudo-elementos	76
5.5 Exercícios opcionais	79
5.6 CSS3: border-radius	80
5.7 CSS3: text-shadow	81
5.8 CSS3: box-shadow	82
5.9 Opacidade e RGBA	84
5.10 Prefixos	87
5.11 CSS3: Gradientes	87
5.12 Progressive Enhancement	89
5.13 Exercícios: Visual CSS3	90
5.14 CSS3 Transitions	91
5.15 CSS3 Transforms	93
5.16 Exercícios: CSS3 transform e transition	95
5.17 Para saber mais: especificidade de seletores CSS	97
6 Web para dispositivos móveis	100
6.1 Site mobile ou mesmo site?	100
6.2 CSS media types	102
6.3 CSS3 media queries	103
6.4 Viewport	104
6.5 Exercícios: Adaptações para mobile	105
6.6 Responsive Web Design	110
6.7 Mobile-first	110
6.8 Exercícios opcionais: Versão tablet	111
7 Progressive enhancement e mobile-first	113
7.1 Formulário de compra	113
7.2 Submissão do formulário	115
7.3 Exercícios: Formulário da página de produto	116
7.4 Design mobile-first	118
7.5 Progressive enhancement	119
7.6 Box model e box-sizing	119
7.7 Exercícios: Página de produto	120

7.8 Evoluindo o design para desktop	125
7.9 Media queries de conteúdo	125
7.10 Exercícios: Responsive design	126
7.11 HTML5 Input range	128
7.12 Exercícios: Seletor de tamanho	129
7.13 Tabelas	130
7.14 Exercícios: Detalhes	132
7.15 Exercícios opcionais: Fundo	134
8 Bootstrap e formulários HTML5	135
8.1 Bootstrap e frameworks de CSS	135
8.2 Estilo e componentes base	135
8.3 A página de checkout da Mirror Fashion	136
8.4 Exercícios: Página de checkout	137
8.5 Formulários a fundo	140
8.6 Novos componentes do HTML5	144
8.7 Novos atributos HTML5 em elementos de formulário	148
8.8 Ícones	149
8.9 Exercícios: Formulários	150
8.10 Validação HTML5	153
8.11 Exercícios: Validação com HTML5	154
8.12 Grid responsivo do Bootstrap	154
8.13 Exercícios: Grids	156
8.14 Para saber mais: componentes JS do Bootstrap	158
8.15 Exercícios opcionais: Navbar e JavaScript	159
8.16 Para saber mais: outros frameworks CSS	161
9 JavaScript e interatividade na Web	163
9.1 Porque usamos JavaScript?	163
9.2 Um pouquinho da história do JavaScript	163
9.3 Características da linguagem	164
9.4 Console do navegador	164
9.5 Sintaxe básica	165
9.6 A tag script	167
9.7 DOM: sua página no mundo JavaScript	168
9.8 Funções e os eventos do DOM	170
9.9 Exercícios: Mostrando tamanho do produto com javascript	171
9.10 Funções Anônimas	172
9.11 Manipulando strings	173

9.12 Manipulando números	173
9.13 Concatenações	173
9.14 Exercícios: Calculando o total da compra	175
9.15 Array	176
9.16 Blocos de Repetição	177
9.17 Funções temporais	178
9.18 Exercícios opcionais: Banner rotativo	179
9.19 Para saber mais: vários callbacks no mesmo elemento	181
9.20 Para saber mais: controlando as validações HTML5	181
10 jQuery	183
10.1 jQuery - A função \$	183
10.2 jQuery Selectors	184
10.3 Filtros customizados e por DOM	185
10.4 Utilitário de iteração do jQuery	185
10.5 Características de execução	186
10.6 Mais produtos na home	186
10.7 Exercícios: jQuery na home	187
10.8 Plugins jQuery	189
10.9 Exercícios: Plugin	190
11 Apêndice - Integrações com serviços Web	191
11.1 Web 2.0 e integrações	191
11.2 iframes	191
11.3 Vídeo embutido com YouTube	191
11.4 Exercícios: iframe	192
11.5 Exercícios opcionais: Google Maps	192
11.6 Fontes customizadas com @font-face	193
11.7 Serviços de web fonts	193
11.8 Exercícios: Google Web Fonts	193
12 Apêndice - Otimizações de front-end	195
12.1 HTML e HTTP - Como funciona a World Wide Web?	196
12.2 Princípios de programação distribuída	198
12.3 Ferramentas de diagnóstico - YSlow e PageSpeed	198
12.4 Compressão e minificação de CSS e JavaScript	199
12.5 Compressão de imagens	201
12.6 Diminuir o número de requests	203
12.7 Juntar arquivos CSS e JS	204

12.8 Image Sprites	205
12.9 Para saber mais	207
12.10 Exercícios: Otimizações Web	207
13 Apêndice - LESS	209
13.1 Variáveis	209
13.2 Contas	210
13.3 Hierarquia	210
13.4 Funções de cores e palhetas automáticas	211
13.5 Reaproveitamento com mixins	212
13.6 Executando o LESS	212
13.7 Para saber mais: recursos avançados e alternativas	213
13.8 Exercícios: LESS	213
14 Apêndice - Subindo sua aplicação no cloud	217
14.1 Como escolher um provedor	217
14.2 O Jelastic Cloud Locaweb	217
14.3 Criando a conta	218
14.4 Importando dados no MySQL	218
14.5 Preparando o projeto	219
14.6 Enviando o projeto e inicializando servidor	220
15 Apêndice - Mais integrações com serviços Web	222
15.1 Botão de curtir do Facebook	222
15.2 Exercícios: Facebook	223
15.3 Para saber mais: Twitter	223
15.4 Para saber mais: Google+	224
15.5 Exercícios opcionais: Twitter e Google+	224

SOBRE O CURSO - O COMPLEXO MUNDO DO FRONT-END

"Ação é a chave fundamental para todo sucesso" -- Pablo Picasso

Vivemos hoje numa era em que a Internet ocupa um espaço cada vez mais importante em nossas vidas pessoais e profissionais. O surgimento constante de Aplicações Web, para as mais diversas finalidades, é um sinal claro de que esse mercado está em franca expansão e traz muitas oportunidades. Aplicações corporativas, comércio eletrônico, redes sociais, filmes, músicas, notícias e tantas outras áreas estão presentes na Internet, fazendo do navegador (o *browser*) o software mais utilizado de nossos computadores.

Esse curso pretende abordar o desenvolvimento de *front-end* (interfaces) para Aplicações Web e Sites que acessamos por meio do navegador de nossos computadores, utilizando padrões atuais de desenvolvimento e conhecendo a fundo suas características técnicas. Discutiremos as implementações dessas tecnologias nos diferentes navegadores, a adoção de *frameworks* que facilitam e aceleram nosso trabalho, além de dicas técnicas que destacam um profissional no mercado. HTML, CSS e JavaScript serão vistos em profundidade.

Além do acesso por meio do navegador de nossos computadores, hoje o acesso à Internet a partir de dispositivos móveis representa um grande avanço da plataforma, mas também implica em um pouco mais de atenção ao trabalho que um programador *front-end* tem que realizar. No decorrer do curso, vamos conhecer algumas dessas necessidades e como utilizar os recursos disponíveis para atender também a essa nova necessidade.

1.1 O CURSO E OS EXERCÍCIOS

Esse é um curso prático que começa nos fundamentos de HTML e CSS, incluindo tópicos relacionados às novidades das versões HTML5 e CSS3. Depois, é abordada a linguagem de programação JavaScript, para enriquecer nossas páginas com interações e efeitos, tanto com JavaScript puro quanto com a biblioteca jQuery, hoje padrão de mercado.

Durante o curso, serão desenvolvidas páginas de um Site de comércio eletrônico. Os exercícios foram projetados para apresentar gradualmente ao aluno quais são as técnicas mais recomendadas e utilizadas quando assumimos o papel do **Programador *front-end***, quais são os desafios mais comuns e

quais são as técnicas e padrões recomendados para atingirmos nosso objetivo, transformando imagens estáticas (os *layouts*) em código que os navegadores entendem e exibem como páginas da Web.

Os exercícios propostos são fundamentais para o acompanhamento do curso, desde os mais iniciais, já que são incrementados no decorrer das aulas. Igualmente importante é a participação ativa nas discussões e debates em sala de aula.

1.2 O PROJETO DE E-COMMERCE

Durante o curso, vamos produzir um site para um e-commerce de moda chamado **Mirror Fashion**. Construiremos várias páginas da loja com intuito de aprender os conceitos de HTML, CSS e JS.

Os conteúdos e o design da loja já foram pré-definidos. Vamos, aqui, focar na implementação, papel do programador front-end.

1.3 TIRANDO DÚVIDAS COM INSTRUTOR

Durante o curso, tire todas as suas dúvidas com o instrutor. HTML, CSS e JavaScript, apesar de parecerem simples e básicos, têm muitas características complexas que não podem deixar de ser totalmente compreendidas pelo aluno. Os instrutores também estão disponíveis para tirar as dúvidas do aluno após o término do treinamento, basta entrar em contato direto com o instrutor ou com a Caelum, teremos o prazer em ajudá-lo.

Se você está acompanhando essa apostila em casa, pense também em fazer o curso presencial na Caelum. Você terá contato direto com o instrutor para esclarecer suas dúvidas, aprender mais tópicos além da apostila, e trocar experiências.

1.4 TIRANDO DÚVIDAS ONLINE NO GUJ

Recomendamos fortemente a busca de recursos e a participação ativa na comunidade por meio das listas de discussão relacionadas ao conteúdo do curso.

O **GUJ.com.br** é um site de perguntas e respostas para desenvolvedores de software que abrange diversas áreas, sendo que front-end é um dos principais focos. A comunidade do GUJ tem mais de 150 mil usuários e 1 milhão e meio de mensagens. É o lugar ideal pra você tirar suas dúvidas e encontrar outros desenvolvedores.

<http://www.guj.com.br>

1.5 BIBLIOGRAFIA

Além do conhecimento disponível na Internet pela comunidade, existem muitos livros interessantes

sobre o assunto. Algumas referências:

- **HTML5 e CSS3: Domine a web do futuro** - Lucas Mazza, editora Casa do Código;
- **A Web Mobile: Design Responsivo e além para uma Web adaptada ao mundo mobile** - Sérgio Lopes, editora Casa do Código;
- **A Arte E A Ciência Do CSS** - Adams & Cols;
- **Pro JavaScript Techniques** - John Resig;
- **The Smashing Book** - smashingmagazine.com

1.6 PARA ONDE IR DEPOIS?

Este curso é parte da **Formação Front-end** da Caelum que engloba também o treinamento **Web Apps com JavaScript Moderno, DOM e jQuery**. Você pode obter mais informações aqui:

<https://www.caelum.com.br/formacao-frontend>

Se o seu desejo é entrar mais a fundo no desenvolvimento Web, incluindo a parte *server-side*, oferecemos o curso **Desenvolvimento Web com PHP e MySQL**, a **Formação Java** e a **Formação .NET** que abordam três caminhos possíveis para esse mundo.

Mais informações em:

- <https://www.caelum.com.br/curso-php-mysql>
- <https://www.caelum.com.br/formacao-java>
- <https://www.caelum.com.br/formacao-dotnet>

INTRODUÇÃO A HTML E CSS

"Quanto mais nos elevamos, menores parecemos aos olhos daqueles que não sabem voar." -- Friedrich Wilhelm Nietzsche

2.1 EXIBINDO INFORMAÇÕES NA WEB

A única linguagem que o navegador consegue interpretar para a exibição de conteúdo é o HTML. Para iniciar a exploração do HTML, vamos imaginar o seguinte caso: o navegador realizou uma requisição e recebeu como corpo da resposta o seguinte conteúdo:

Mirror Fashion

Bem-vindo à Mirror Fashion, sua loja de roupas e acessórios.

Confira nossas promoções.

Receba informações sobre nossos lançamentos por email.

Navegue por todos nossos produtos em catálogo.

Compre sem sair de casa.

Para conhecer o comportamento dos navegadores quanto ao conteúdo descrito antes, vamos reproduzir esse conteúdo em um arquivo de texto comum, que pode ser criado com qualquer editor de texto puro. Salve o arquivo como **index.html** e abra-o a partir do navegador à sua escolha.



Parece que obtemos um resultado um pouco diferente do esperado, não? Apesar de ser capaz de exibir texto puro em sua área principal, algumas regras devem ser seguidas caso desejemos que esse texto seja exibido com alguma formatação, para facilitar a leitura pelo usuário final.

Usando o resultado acima podemos concluir que o navegador por padrão:

- Pode não exibir caracteres acentuados corretamente;
- Não exibe quebras de linha.

Para que possamos exibir as informações desejadas com a formatação, é necessário que cada trecho de texto tenha uma **marcação** indicando qual é o significado dele. Essa marcação também influencia a maneira com que cada trecho do texto será exibido. A seguir é listado o texto com uma marcação correta:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Mirror Fashion</title>
    <meta charset="utf-8">
  </head>
  <body>
    <h1>Mirror Fashion</h1>
    <h2>Bem-vindo à Mirror Fashion, sua loja de roupas e acessórios.</h2>
    <ul>
      <li>Confira nossas promoções.</li>
      <li>Receba informações sobre nossos lançamentos por email.</li>
```

```
<li>Navegue por todos nossos produtos em catálogo.</li>
<li>Compre sem sair de casa.</li>
</ul>
</body>
</html>
```

Reproduza o código anterior em um novo arquivo de texto puro e salve-o como **index-2.html**. Não se preocupe com a sintaxe, vamos conhecer detalhadamente cada característica do HTML nos próximos capítulos. Abra o arquivo no navegador.



Agora, o resultado é exibido de maneira muito mais agradável e legível. Para isso, tivemos que utilizar algumas marcações do HTML. Essas marcações são chamadas de **tags**, e elas basicamente dão **significado** ao texto contido entre sua abertura e fechamento.

Apesar de estarem corretamente marcadas, as informações não apresentam nenhum atrativo estético e, nessa deficiência do HTML, reside o primeiro e maior desafio do programador front-end.

O HTML foi desenvolvido para exibição de documentos científicos. Para termos uma comparação, é como se a Web fosse desenvolvida para exibir monografias redigidas e formatadas pela Metodologia do Trabalho Científico da ABNT. Porém, com o tempo e a evolução da Web e de seu potencial comercial, tornou-se necessária a exibição de informações com grande riqueza de elementos gráficos e de interação.

Começaremos por partes, primeiro entenderemos como o HTML funciona, para depois aprendermos

estilos, elementos gráficos e interações.

2.2 SINTAXE DO HTML

O HTML é um conjunto de **tags** responsáveis pela marcação do conteúdo de uma página no navegador. No código que vimos antes, as tags são os elementos a mais que escrevemos usando a sintaxe `<nomedatag>`. Diversas tags são disponibilizadas pela linguagem HTML e cada uma possui uma funcionalidade específica.

No código de antes, vimos por exemplo o uso da tag `<h1>`. Ela representa o título principal da página.

```
<h1>Mirror Fashion</h1>
```

Note a sintaxe. Uma tag é definida com caracteres `<` e `>`, e seu nome (`h1` no caso). Muitas tags possuem conteúdo, como o texto do título ("*Mirror Fashion*"). Nesse caso, para determinar onde o conteúdo acaba, usamos uma *tag de fechamento* com a barra antes do nome: `</h1>`.

Algumas tags podem receber **atributos** dentro de sua definição. São parâmetros usando a sintaxe de `nome=valor`. Para definir uma imagem, por exemplo, usamos a tag `` e, para indicar qual imagem carregar, usamos o atributo `src`:

```

```

Repare que a tag `img` não possui conteúdo por si só. Nesses casos, **não** é necessário usar uma tag de fechamento como antes no `h1`.

2.3 ESTRUTURA DE UM DOCUMENTO HTML

Um documento HTML válido precisa seguir obrigatoriamente a estrutura composta pelas tags `<html>`, `<head>` e `<body>` e a instrução `<!DOCTYPE>`. Vejamos cada uma delas:

A tag `<html>`

Na estrutura do nosso documento, antes de tudo, inserimos uma tag `<html>`. Dentro dessa tag, é necessário declarar outras duas tags: `<head>` e `<body>`. Essas duas tags são "irmãs", pois estão no mesmo nível hierárquico em relação à sua tag "pai", que é `<html>`.

```
<html>
  <head></head>
  <body></body>
</html>
```

A tag `<head>`

A tag `<head>` contém informações sobre nosso documento que são de interesse somente do navegador, e não dos visitantes do nosso site. São informações que não serão exibidas na área do documento no navegador.

A especificação obriga a presença da tag de conteúdo `<title>` dentro do nosso `<head>`, permitindo especificar o título do nosso documento, que normalmente será exibido na *barra de título* da janela do navegador ou na *aba* do documento.

Outra configuração muito utilizada, principalmente em documentos cujo conteúdo é escrito em um idioma como o português, que tem caracteres como acentos e cedilha, é a configuração da codificação de caracteres, chamada de **encoding** ou **charset**.

Podemos configurar qual codificação queremos utilizar em nosso documento por meio da configuração de charset na tag `<meta>`. Um dos valores mais comuns usados hoje em dia é o **UTF-8**, também chamado de **Unicode**. Há outras possibilidades, como o **latin1**, muito usado antigamente.

O **UTF-8** é a recomendação atual para encoding na Web por ser amplamente suportada em navegadores e editores de código, além de ser compatível com praticamente todos os idiomas do mundo. É o que usaremos no curso.

```
<html>
  <head>
    <meta charset="utf-8">
    <title>Mirror Fashion</title>
  </head>
  <body>

  </body>
</html>
```

A tag `<body>`

A tag `<body>` contém o corpo do nosso documento, que é exibido pelo navegador em sua janela. É necessário que o `<body>` tenha ao menos um elemento "filho", ou seja, uma ou mais tags HTML dentro dele.

```
<html>
  <head>
    <meta charset="utf-8">
    <title>Mirror Fashion</title>
  </head>
  <body>
    <h1>A Mirror Fashion</h1>
  </body>
</html>
```

Nesse exemplo, usamos a tag `<h1>`, que indica um título.

A instrução DOCTYPE

O `DOCTYPE` não é uma tag HTML, mas uma instrução especial. Ela indica para o navegador qual **versão do HTML** deve ser utilizada para renderizar a página. Utilizaremos `<!DOCTYPE html>`, que indica para o navegador a utilização da versão mais recente do HTML - a versão 5, atualmente.

Há muitos comandos complicados nessa parte de `DOCTYPE` que eram usados em versões anteriores do HTML e do XHTML. Hoje em dia, nada disso é mais importante. O recomendado é **sempre usar a última versão do HTML**, usando a declaração de `DOCTYPE` simples:

```
<!DOCTYPE html>
```

2.4 TAGS HTML

O HTML é composto de diversas tags, cada uma com sua função e significado. O HTML 5, então, adicionou muitas novas tags, que veremos ao longo do curso.

Nesse momento, vamos focar em tags que representam **títulos, parágrafo e ênfase**.

Títulos

Quando queremos indicar que um texto é um título em nossa página, utilizamos as tags de *heading* em sua marcação:

```
<h1>Mirror Fashion</h1>
<h2>Bem-vindo à Mirror Fashion, sua loja de roupas e acessórios.</h2>
```

As tags de *heading* são tags de conteúdo e vão de `<h1>` a `<h6>`, seguindo a ordem de importância, sendo `<h1>` o título principal, o mais importante, e `<h6>` o título de menor importância.

Utilizamos, por exemplo, a tag `<h1>` para o nome, título principal da página, e a tag `<h2>` como subtítulo ou como título de seções dentro do documento.

A ordem de importância, além de influenciar no tamanho padrão de exibição do texto, tem impacto nas ferramentas que processam HTML. As ferramentas de indexação de conteúdo para buscas, como o Google, Bing ou Yahoo! levam em consideração essa ordem e relevância. Os navegadores especiais para acessibilidade também interpretam o conteúdo dessas tags de maneira a diferenciar seu conteúdo e facilitar a navegação do usuário pelo documento.

Parágrafos

Quando exibimos qualquer texto em nossa página, é recomendado que ele seja sempre conteúdo de alguma tag filha da tag `<body>`. A marcação mais indicada para textos comuns é a tag de **parágrafo**:

```
<p>Nenhum item na sacola de compras.</p>
```

Se você tiver vários parágrafos de texto, use várias dessas tags `<p>` para separá-los:

```
<p>Um parágrafo de texto.</p>
<p>Outro parágrafo de texto.</p>
```

Marcações de ênfase

Quando queremos dar uma ênfase diferente a um trecho de texto, podemos utilizar as marcações de ênfase. Podemos deixar um texto "mais forte" com a tag `` ou deixar o texto com uma "ênfase acentuada" com a tag ``. Também há a tag `<small>`, que diminui o tamanho do texto.

Por padrão, os navegadores renderizarão o texto dentro da tag `` em negrito e o texto dentro da tag `` em itálico. Existem ainda as tags `` e `<i>`, que atingem o mesmo resultado visualmente, mas as tags `` e `` são mais indicadas por definirem nossa intenção de significado ao conteúdo, mais do que uma simples indicação visual. Vamos discutir melhor a questão do significado das tags mais adiante.

```
<p>Compre suas roupas e acessórios na <strong>Mirror Fashion</strong>.</p>
```

2.5 IMAGENS

A tag `` define uma imagem em uma página HTML e necessita de dois atributos preenchidos: `src` e `alt`. O primeiro é um atributo obrigatório e aponta para o local da imagem, já o segundo é um texto alternativo que aparece caso a imagem não possa ser carregada ou visualizada. O atributo `alt` não é obrigatório, mas é importante ser preenchido para que leitores de tela e robôs de busca, como o Google, consigam ler o conteúdo da imagem.

O HTML 5 introduziu duas novas tags específicas para imagem: `<figure>` e `<figcaption>`. A tag `<figure>` define uma imagem com a conhecida tag ``. Além disso, permite adicionar uma legenda para a imagem por meio da tag `<figcaption>`.

```
<figure>
  
  <figcaption>Fuzz Cardigan por R$ 129,90</figcaption>
</figure>
```

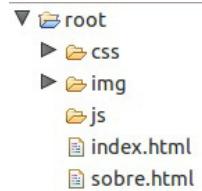
2.6 A ESTRUTURA DOS ARQUIVOS DE UM PROJETO

Como todo tipo de projeto de software, existem algumas recomendações quanto à organização dos arquivos de um site. Não há nenhum rigor técnico quanto a essa organização e, na maioria das vezes, você vai adaptar as recomendações da maneira que for melhor para o seu projeto.

Como um site é um conjunto de páginas Web sobre um assunto, empresa, produto ou qualquer outra coisa, é comum todos os arquivos de um site estarem dentro de uma só pasta e, assim como um livro, é recomendado que exista uma "capa", uma página inicial que possa indicar para o visitante quais são as outras páginas que fazem parte desse projeto e como ele pode acessá-las, como se fosse o **índice** do site.

Esse índice, não por coincidência, é convenção adotada pelos servidores de páginas Web. Se desejamos que uma determinada pasta seja servida como um site e dentro dessa pasta existe um arquivo chamado **index.html**, esse arquivo será a página inicial a menos que alguma configuração determine outra página para esse fim.

Dentro da pasta do site, no mesmo nível que o `index.html`, é recomendado que sejam criadas mais algumas pastas para manter separados os arquivos de imagens, as folhas de estilo e os scripts. Para iniciar um projeto, teríamos uma estrutura de pastas como a demonstrada na imagem a seguir:



Muitas vezes, um site é servido por meio de uma aplicação Web e, nesses casos, a estrutura dos arquivos depende de como a aplicação necessita dos recursos para funcionar corretamente. Porém, no geral, as aplicações também seguem um padrão bem parecido com o que estamos adotando para o nosso projeto.

2.7 EDITORES E IDES

Existem editores de texto como *Gedit* (www.gnome.org), *Sublime* (<http://www.sublimetext.com>), *Atom* (<http://atom.io>) e *Notepad++* (<http://notepad-plus-plus.org>), que possuem realce de sintaxe e outras ferramentas para facilitar o desenvolvimento de páginas. Há também IDEs (Integrated Development Environment), que oferecem recursos como autocompletar e pré-visualização, como *Eclipse* (<https://www.eclipse.org>) e *Visual Studio* (<https://visualstudio.microsoft.com>).

2.8 PRIMEIRA PÁGINA

A primeira página que desenvolveremos para a *Mirror Fashion* será a *Sobre*, que explica detalhes sobre a empresa, apresenta fotos e a história.

Recebemos o design já pronto, assim como os textos. Nosso trabalho, como desenvolvedores de front-end, é codificar o HTML e CSS necessários para esse resultado.

A Mirror Fashion

A **Mirror Fashion** é a maior empresa de comércio eletrônico no segmento de moda em todo o mundo. Fundada em 1932, possui filiais em 124 países, sendo líder de mercado com mais de 90% de participação em 118 deles.

Nosso centro de distribuição fica em [Jacarezinho, no Paraná](#). De lá, saem 48 aviões que distribuem nossos produtos às casas do mundo todo. Nossa centro de distribuição:



Centro de distribuição da Mirror Fashion

Compre suas roupas e acessórios na Mirror Fashion. Acesse [nossa loja](#) ou entre em contato se tiver dúvidas. Conheça também nossa [história](#) e nossos [diferenciais](#).

História

A fundação em 1932 ocorreu no momento da descoberta econômica do interior do Paraná. A família Pelho, tradicional da região, investiu todas as suas economias nessa nova iniciativa, revolucionária para a época. O fundador *Eduardo Simões Pelho*, dotado de particular visão administrativa, guiou os negócios da empresa durante mais de 50 anos, muitos deles ao lado de seu filho *E. S. Pelho Filho*, atual CEO. O nome da empresa é inspirado no nome da família.

O crescimento da empresa foi praticamente instantâneo. Nos primeiros 5 anos, já atendia 18 países. Bateu a marca de 100 países em apenas 15 anos de existência. Até hoje, já atendeu 740 milhões de usuários diferentes, em bilhões de diferentes pedidos.

O crescimento em número de funcionários é também assombroso. Hoje, é a maior empregadora do Brasil, mas mesmo após apenas 5 anos de sua existência, já possuía 30 mil funcionários. Fora do Brasil, há 240 mil funcionários, além dos 890 mil brasileiros nas instalações de Jacarezinho e nos escritórios em todo país.

Dada a importância econômica da empresa para o Brasil, a família Pelho já recebeu diversos prêmios, homenagens e condecorações. Todos os presidentes do Brasil já visitaram as instalações da Mirror Fashion, além de presidentes da União Européia, Ásia e o secretário-geral da ONU.



Família Pelho

Diferenciais

- Menor preço do varejo, garantido
- Se você achar uma loja mais barata, leva o produto de graça
- Pague em reais, dólares, euros ou bitcoins
- Todas as compras com frete grátis para o mundo todo
- Maior comércio eletrônico de moda do mundo
- Atendimento via telefone, email, chat, twitter, facebook, carta, fax e telegrama
- Presente em 124 países
- Mais de um milhão de funcionários em todo o mundo

2.9 EXERCÍCIOS: PRIMEIROS PASSOS COM HTML

1. Ao longo do curso, usaremos diversas imagens que o nosso designer preparou. Nesse ponto, vamos importar todas as imagens e demais arquivos que usaremos no decorrer do curso.

- Copie a pasta **mirror-fashion** de dentro da pasta **Caelum/43** para a área de trabalho de sua máquina.
- Verifique a pasta **img**, cheia de arquivos do design do site. Além desta pasta, teremos as pastas **js** e **css**, que ainda não usaremos.

EM CASA

Você pode baixar um ZIP com todos os arquivos necessários para o curso aqui:

<https://s3.amazonaws.com/apostilas-caelum/caelum-arquivos-curso-web.zip>

2. Dentro da pasta **mirror-fashion**, vamos criar o arquivo **sobre.html** com a estrutura básica contendo o DOCTYPE e as tags `html` , `head` e `body` :

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Sobre a Mirror Fashion</title>
  </head>
  <body>
    ... conteúdo da página ...
  </body>
</html>
```

3. A página deve ter uma imagem com o logo da empresa, um título e um texto falando sobre ela.

O texto para ser colocado na página está no arquivo **sobre.txt** disponível na pasta **Caelum/43/textos**. São vários parágrafos que devem ser adaptados com o HTML apropriado.

Após copiar o texto do arquivo **sobre.txt**, coloque cada um dos parágrafos dentro de uma tag `<p>` . Coloque também o título *História* dentro de uma tag `<h2>` .

Use a tag `` para o logo e a tag `<h1>` para o título. Seu HTML deve ficar assim, no final:

```


<h1>A Mirror Fashion</h1>

<p>
  A Mirror Fashion é a maior empresa de comércio eletrônico no segmento
```

```
de moda em todo o mundo. Fundada em 1932, possui filiais  
em 124 países.....  
</p>
```

4. Um texto corrido sem ênfases é difícil de ler. Use negritos e itálicos para destacar partes importantes.

Use a tag `` para a ênfase mais forte em negrito, por exemplo para destacar o nome da empresa no texto do primeiro parágrafo:

```
<p>A <strong>Mirror Fashion</strong> é a maior empresa comércio eletrônico.....</p>
```

Use também a ênfase com `` que deixará o texto em itálico. Na parte da *História*, coloque os nomes das pessoas e da família em ``.

5. A página deve ter duas imagens. A primeira apresenta o centro da *Mirror Fashion* e deve ser inserida **após o segundo parágrafo do texto**. A outra, é uma imagem da *Família Pelho* e deve ser colocada **após o subtítulo da História**.

As imagens podem ser carregadas com a tag ``, apontando seu caminho. Além disso, no HTML5, podemos usar as tags `<figure>` e `<figcaption>` para destacar a imagem e colocar uma legenda em cada uma.

A imagem do centro de distribuição está em **img/centro-distribuicao.png**:

```
<figure>  
    
  <figcaption>Centro de distribuição da Mirror Fashion</figcaption>  
</figure>
```

A imagem da família é a **img/familia-pelho.jpg** e deve ser colocada na parte de *História*:

```
<figure>  
    
  <figcaption>Família Pelho</figcaption>  
</figure>
```

6. Confira se o seu código final está como a seguir:

```
<!DOCTYPE html>  
<html>  
  <head>  
    <meta charset="utf-8">  
    <title>Sobre a Mirror Fashion</title>  
  </head>  
  
  <body>  
      
    <h1>A Mirror Fashion</h1>  
  
    <p>  
      A <strong>Mirror Fashion</strong> é a maior empresa comércio eletrônico no segmento de  
      moda em todo o mundo. Fundada em 1932, possui filiais em 124 países, sendo líder de  
      mercado com mais de 90% de participação em 118 deles.  
    </p>
```

```

<p>
    Nosso centro de distribuição fica em Jacarezinho, no Paraná. De lá, saem 48 aviões que
    distribuem nossos produtos às casas do mundo todo. Nosso centro de distribuição:
</p>

<figure>
    
    <figcaption>Centro de distribuição da Mirror Fashion</figcaption>
</figure>

<p>
    Compre suas roupas e acessórios na Mirror Fashion. Acesse nossa loja ou entre em contato
    se tiver dúvidas. Conheça também nossa história e nossos diferenciais.
</p>

<h2>História</h2>

<figure>
    
    <figcaption>Família Pelho</figcaption>
</figure>

<p>
    A fundação em 1932 ocorreu no momento da descoberta econômica do interior do Paraná. A
    família Pelho, tradicional da região, investiu todas as suas economias nessa nova
    iniciativa, revolucionária para a época. O fundador <em>Eduardo Simões Pelho</em>,
    dotado de particular visão administrativa, guiou os negócios da empresa durante mais
    de 50 anos, muitos deles ao lado de seu filho <em>E. S. Pelho Filho</em>, atual CEO.
    O nome da empresa é inspirado no nome da família.
</p>

<p>
    O crescimento da empresa foi praticamente instantâneo. Nos primeiros 5 anos, já atendia
    18 países. Bateu a marca de 100 países em apenas 15 anos de existência. Até hoje, já
    atendeu 740 milhões de usuários diferentes, em bilhões de diferentes pedidos.
</p>

<p>
    O crescimento em número de funcionários é também assombroso. Hoje, é a maior
    empregadora do Brasil, mas mesmo após apenas 5 anos de sua existência, já
    possuía 30 mil funcionários. Fora do Brasil, há 240 mil funcionários, além
    dos 890 mil brasileiros nas instalações de Jacarezinho e nos escritórios em
    todo país.
</p>

<p>
    Dada a importância econômica da empresa para o Brasil, a família Pelho já
    recebeu diversos prêmios, homenagens e condecorações. Todos os presidentes
    do Brasil já visitaram as instalações da Mirror Fashion, além de presidentes
    da União Européia, Ásia e o secretário-geral da ONU.
</p>
</body>
</html>

```

7. Verifique o resultado no navegador. Devemos já ver o conteúdo e as imagens na sequência, mas sem um design muito interessante.

BOA PRÁTICA - INDENTAÇÃO

Uma prática sempre recomendada, ligada à limpeza e utilizada para facilitar a leitura do código, é o uso correto de **recuos**, ou **indentação**, no HTML. Costumamos alinhar elementos "irmãos" na mesma margem e adicionar alguns espaços ou um *tab* para elementos "filhos".

A maioria dos exercícios dessa apostila utiliza um padrão recomendado de recuos.

BOA PRÁTICA - COMENTÁRIOS

Quando iniciamos nosso projeto, utilizamos poucas tags HTML. Mais tarde adicionaremos uma quantidade razoável de elementos, o que pode gerar uma certa confusão. Para manter o código mais legível, é recomendada a adição de comentários antes da abertura e após do fechamento de tags estruturais (que conterão outras tags). Dessa maneira, nós podemos identificar claramente quando um elemento está **dentro** dessa estrutura ou **depois** dela.

```
<!-- início do cabeçalho -->
<header>
  <p>Esse parágrafo está "dentro" da área principal.</p>
</header>
<!-- fim do cabeçalho -->

<p>Esse parágrafo está "depois" da área principal.</p>
```

2.10 ESTILIZANDO COM CSS

Quando escrevemos o HTML, marcamos o conteúdo da página com tags que melhor representam o significado daquele conteúdo. Aí quando abrimos a página no navegador é possível perceber que ele mostra as informações com estilos diferentes.

Um h1, por exemplo, fica em negrito numa fonte maior. Parágrafos de texto são espaçados entre si, e assim por diante. Isso quer dizer que o navegador tem um *estilo padrão* para as tags que usamos. Mas, claro, pra fazer sites bonitões vamos querer *customizar o design dos elementos* da página.

Antigamente, isso era feito no próprio HTML. Se quisesse um título em vermelho, era só fazer:

```
<h1><font color="red">Mirror Fashion anos 90</font></h1>
```

Além da tag ``, várias outras tags de estilo existiam. Mas isso é passado. **Tags HTML para estilo são má prática** hoje em dia e jamais devem ser usadas.

Em seu lugar, surgiu o **CSS**, que é uma outra linguagem, separada do HTML, com objetivo único de

cuidar da estilização da página. A vantagem é que o CSS é bem mais robusto que o HTML para estilização, como veremos. Mas, principalmente, escrever formatação visual misturado com conteúdo de texto no HTML se mostrou algo bem impraticável. O CSS resolve isso separando as coisas; regras de estilo não aparecem mais no HTML, apenas no CSS.

2.11 SINTAXE E INCLUSÃO DE CSS

A sintaxe do CSS tem estrutura simples: é uma declaração de propriedades e valores separados por um sinal de dois pontos ":", e cada propriedade é separada por um sinal de ponto e vírgula ";" da seguinte maneira:

```
{color: blue;  
background-color: yellow;}
```

O elemento que receber essas propriedades será exibido com o texto na cor azul e com o fundo amarelo. Essas propriedades podem ser declaradas de três maneiras diferentes.

Atributo style

A primeira delas é com o atributo `style` no próprio elemento:

```
<p style="color: blue; background-color: yellow;">  
O conteúdo desta tag será exibido em azul com fundo amarelo no navegador!  
</p>
```

Mas tínhamos acabado de discutir que uma das grandes vantagens do CSS era manter as regras de estilo fora do HTML. Usando esse atributo `style` não parece que fizemos isso. Justamente por isso não se recomenda esse tipo de uso na prática, mas sim os que veremos a seguir.

A tag style

A outra maneira de se utilizar o CSS é declarando suas propriedades dentro de uma tag `<style>`.

Como estamos declarando as propriedades visuais de um elemento em outro lugar do nosso documento, precisamos indicar de alguma maneira a qual elemento nos referimos. Fazemos isso utilizando um **seletor CSS**. É basicamente uma forma de buscar certos elementos dentro da página que receberão as regras visuais que queremos.

No exemplo a seguir, usaremos o seletor que pega todas as tags `p` e altera sua cor e background:

```
<!DOCTYPE html>  
<html>  
<head>  
<meta charset="utf-8">  
<title>Sobre a Mirror Fashion</title>  
<style>  
  p {  
    color: blue;  
    background-color: yellow;
```

```

        }
    </style>
</head>
<body>
    <p>
        O conteúdo desta tag será exibido em azul com fundo amarelo!
    </p>
    <p>
        <strong>Também</strong> será exibido em azul com fundo amarelo!
    </p>
</body>
</html>

```

O código dentro da tag `<style>` indica que estamos alterando a cor e o fundo de todos os elementos com tag `p`. Dizemos que selecionamos esses elementos pelo nome de sua tag, e aplicamos certas propriedades CSS apenas neles.

Arquivo externo

A terceira maneira de declararmos os estilos do nosso documento é com um arquivo externo, geralmente com a extensão `.css`. Para que seja possível declarar nosso CSS em um arquivo à parte, precisamos indicar em nosso documento HTML uma ligação entre ele e a folha de estilo (arquivo com a extensão `.css`).

Além da melhor organização do projeto, a folha de estilo externa traz ainda as vantagens de manter nosso HTML mais limpo e do reaproveitamento de uma mesma folha de estilos para diversos documentos.

A indicação de uso de uma folha de estilos externa deve ser feita dentro da tag `<head>` do nosso documento HTML:

```

<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Sobre a Mirror Fashion</title>
        <link rel="stylesheet" href="estilos.css">
    </head>
    <body>
        <p>
            O conteúdo desta tag será exibido em azul com fundo amarelo!
        </p>
        <p>
            <strong>Também</strong> será exibido em azul com fundo amarelo!
        </p>
    </body>
</html>

```

E dentro do arquivo `estilos.css` colocamos apenas o conteúdo do CSS:

```

p {
    color: blue;
    background-color: yellow;
}

```

2.12 PROPRIEDADES TIPOGRÁFICAS E FONTES

Da mesma maneira que alteramos cores, podemos alterar o texto. Podemos definir fontes com o uso da propriedade `font-family`.

A propriedade `font-family` pode receber seu valor com ou sem aspas. No primeiro caso, passaremos o nome do arquivo de fonte a ser utilizado, no último, passaremos a família da fonte.

Por padrão, os navegadores mais conhecidos exibem texto em um tipo que conhecemos como "serif". As fontes mais conhecidas (e comumente utilizadas como padrão) são "Times" e "Times New Roman", dependendo do sistema operacional. Elas são chamadas de **fontes serifadas** pelos pequenos ornamentos em suas terminações.

Podemos alterar a família de fontes que queremos utilizar em nosso documento para a família "sans-serif" (sem serifas), que contém, por exemplo, as fontes "Arial" e "Helvetica". Podemos também declarar que queremos utilizar uma família de fontes "monospace" como, por exemplo, a fonte "Courier".

```
h1 {  
    font-family: serif;  
}  
  
h2 {  
    font-family: sans-serif;  
}  
  
p {  
    font-family: monospace;  
}
```

É possível, e muito comum, declararmos o nome de algumas fontes que gostaríamos de verificar se existem no computador, permitindo que tenhamos um controle melhor da forma como nosso texto será exibido. Normalmente, declaramos as fontes mais comuns, e existe um grupo de fontes que são consideradas "seguras" por serem bem populares.

Em nosso projeto, as fontes não têm ornamentos, vamos declarar essa propriedade para todo o documento por meio do seu elemento `body`:

```
body {  
    font-family: "Arial", "Helvetica", sans-serif;  
}
```

Nesse caso, o navegador verificará se a fonte "Arial" está disponível e a utilizará para renderizar os textos de todos os elementos do nosso documento que, por cascata, herdarão essa propriedade do elemento `body`.

Caso a fonte "Arial" não esteja disponível, o navegador verificará a disponibilidade da próxima fonte declarada, no nosso exemplo a "Helvetica". Caso o navegador não encontre também essa fonte, ele

solicita qualquer fonte que pertença à família "sans-serif", declarada logo a seguir, e a utiliza para exibir o texto, não importa qual seja ela.

Temos outras propriedades para manipular a fonte, como a propriedade `font-style`, que define o estilo da fonte que pode ser: `normal` (normal na vertical), `italic` (inclinada) e `oblique` (oblíqua).

2.13 ALINHAMENTO E DECORAÇÃO DE TEXTO

Já vimos uma série de propriedades e subpropriedades que determinam o tipo e estilo da fonte. Vamos conhecer algumas maneiras de alterarmos as disposições dos textos.

Uma das propriedades mais simples, porém muito utilizada, é a que diz respeito ao alinhamento de texto: a propriedade `text-align`.

```
p {  
    text-align: right;  
}
```

O exemplo anterior determina que todos os parágrafos da nossa página tenham o texto alinhado para a direita. Também é possível determinar que um elemento tenha seu conteúdo alinhado ao centro ao definirmos o valor `center` para a propriedade `text-align`, ou então definir que o texto deve ocupar toda a largura do elemento aumentando o espaçamento entre as palavras com o valor `justify`. O padrão é que o texto seja alinhado à esquerda, com o valor `left`, porém é importante lembrar que essa propriedade propaga-se em cascata.

É possível configurar também uma série de espaçamentos de texto com o CSS:

```
p {  
    line-height: 3px; /* tamanho da altura de cada linha */  
    letter-spacing: 3px; /* tamanho do espaço entre cada letra */  
    word-spacing: 5px; /* tamanho do espaço entre cada palavra */  
    text-indent: 30px; /* tamanho da margem da primeira linha do texto */  
}
```

2.14 IMAGEM DE FUNDO

A propriedade `background-image` permite indicar um arquivo de imagem para ser exibido ao fundo do elemento. Por exemplo:

```
h1 {  
    background-image: url(sobre-background.jpg);  
}
```

Com essa declaração, o navegador vai requisitar um arquivo `sobre-background.jpg`, que deve estar na mesma pasta do arquivo CSS onde consta essa declaração.

2.15 BORDAS

As propriedades do CSS para definirmos as **bordas** de um elemento nos apresentam uma série de opções. Podemos, para cada borda de um elemento, determinar sua cor, seu estilo de exibição e sua largura. Por exemplo:

```
body {  
    border-color: red;  
    border-style: solid;  
    border-width: 1px;  
}
```

A propriedade `border` tem uma forma resumida para escrever os mesmos estilos que adicionamos acima, mas de uma maneira mais simples:

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

Para que o efeito da cor sobre a borda surta efeito, é necessário que a propriedade `border-style` tenha qualquer valor diferente do padrão `none`.

Conseguimos fazer também comentários no CSS usando a seguinte sintaxe:

```
/* deixando o fundo ridículo */  
body {  
    background-color: gold;  
}
```

2.16 EXERCÍCIOS: PRIMEIROS PASSOS COM CSS

- Aplicaremos um pouco de estilo em nossa página usando CSS. Dentro da pasta `css`, **crie um arquivo** `sobre.css`, que conterá nosso código de estilo para essa página.

Em primeiro lugar, precisamos carregar o arquivo `sobre.css` dentro da página `sobre.html`, com a tag `<link>` que deve ser adicionada dentro da tag `<head>`:

```
<head>  
    <meta charset="utf-8">  
    <title>Sobre a Mirror Fashion</title>  
    <link rel="stylesheet" href="css/sobre.css">  
</head>
```

No arquivo `sobre.css`, para o elemento `<body>`, altere a sua cor e sua fonte base por meio das propriedades `color` e `font-family`:

```
body {  
    color: #333333;  
    font-family: "Lucida Sans Unicode", "Lucida Grande", sans-serif;  
}
```

O título principal deve ter um fundo estampado com a imagem `img/sobre-background.jpg`. Use a propriedade `background-image` pra isso. Aproveite e coloque uma borda sutil nos subtítulos, para ajudar a separar o conteúdo.

```
h1 {  
    background-image: url(..../img/sobre-background.jpg);  
}  
  
h2 {  
    border-bottom: 2px solid #333333;  
}
```

Acerte também a renderização das figuras. Coloque um fundo cinza, uma borda sutil, deixe a legenda em itálico com `font-style` e alinhe a imagem e a legenda no centro com `text-align`.

```
figure {  
    background-color: #f2edeb;  
    border: 1px solid #ccc;  
    text-align: center;  
}  
  
figcaption {  
    font-style: italic;  
}
```

Teste o resultado no navegador. Nossa página começa a pegar o estilo da página final!



2. (opcional) Teste outros estilos de bordas em vez do `solid` que vimos no exercício anterior. Algumas possibilidades: `dashed` , `dotted` , `double` , `groove` e outros.

Teste também outras possibilidades para o `text-align`, como `left`, `right` e `justify`.

2.17 CORES NA WEB

Propriedades como `background-color`, `color`, `border-color`, entre outras aceitam uma cor como valor. Existem várias maneiras de definir cores quando utilizamos o CSS.

A primeira, mais simples e ingênua, é usando o nome da cor:

```
h1 {  
    color: red;  
}  
  
h2 {  
    background-color: yellow;  
}
```

O difícil é acertar a exata variação de cor que queremos no design. Por isso, é bem incomum usarmos cores com seus nomes. O mais comum é definir a cor com base em sua composição RGB.

RGB é um sistema de cor bastante comum aos designers. Ele permite especificar até 16 milhões de cores com uma combinação de três cores base: Vermelho (Red), Verde (Green), Azul (Blue). Podemos escolher a intensidade de cada um desses três canais básicos, numa escala de 0 a 255.

Um amarelo forte, por exemplo, tem 255 de Red, 255 de Green e 0 de Blue (255, 255, 0). Se quiser um laranja, basta diminuir um pouco o verde (255, 200, 0). E assim por diante.

No CSS, podemos escrever as cores tendo como base sua composição RGB. Aliás, no CSS3 - que veremos melhor depois - há até uma sintaxe bem simples pra isso:

```
h3 {  
    color: rgb(255, 200, 0);  
}
```

Essa sintaxe funciona nos browsers mais modernos mas não é a mais comum na prática, por questões de compatibilidade. O mais comum é a **notação hexadecimal**, que acabamos usando no exercício anterior ao escrever `#f2edeb`. Essa sintaxe tem suporte universal nos navegadores e é mais curta de escrever, apesar de ser mais enigmática.

```
h3 {  
    background-color: #f2edeb;  
}
```

No fundo, porém, as duas formas são baseadas no sistema RGB. Na notação hexadecimal (que começa com `#`), temos 6 caracteres, os primeiros 2 indicam o canal Red, os dois seguintes, o Green, e os dois últimos, Blue; ou seja, RGB. E usamos a matemática pra escrever menos, trocando a base numérica de decimal para hexadecimal.

Na base hexadecimal, os algarismos vão de zero a quinze (ao invés do zero a nove da base decimal)

comum). Para representar os algarismos de dez a quinze, usamos letras de A a F. Nessa sintaxe, portanto, podemos utilizar números de 0-9 e letras de A-F.

Há uma conta por trás dessas conversões, mas seu editor de imagens deve ser capaz de fornecer ambos os valores para você sem problemas. Um valor 255 vira FF na notação hexadecimal. A cor **#f2edeb**, por exemplo, é equivalente a **rgb(242, 237, 237)**, um cinza claro.

Vale aqui uma dica quanto ao uso de cores hexadecimais, toda vez que os caracteres presentes na composição da base se repetirem, estes podem ser simplificados. Então um número em hexadecimal **3366ff**, pode ser simplificado para **36f**.

2.18 LISTAS HTML

Não são raros os casos em que queremos exibir uma listagem em nossas páginas. O HTML tem algumas tags definidas para que possamos fazer isso de maneira correta. A lista mais comum é a lista não-ordenada.

```
<ul>
  <li>Primeiro item da lista</li>
  <li>
    Segundo item da lista:
    <ul>
      <li>Primeiro item da lista aninhada</li>
      <li>Segundo item da lista aninhada</li>
    </ul>
  </li>
  <li>Terceiro item da lista</li>
</ul>
```

Note que, para cada item da lista não-ordenada, utilizamos uma marcação de item de lista ****. No exemplo acima, utilizamos uma estrutura composta na qual o segundo item da lista contém uma nova lista. A mesma tag de item de lista **** é utilizada quando demarcamos uma lista ordenada.

```
<ol>
  <li>Primeiro item da lista</li>
  <li>Segundo item da lista</li>
  <li>Terceiro item da lista</li>
  <li>Quarto item da lista</li>
  <li>Quinto item da lista</li>
</ol>
```

As listas ordenadas também podem ter sua estrutura composta por outras listas ordenadas como no exemplo que temos para as listas não-ordenadas. Também é possível ter listas ordenadas aninhadas em um item de uma lista não-ordenada e vice-versa.

Existe um terceiro tipo de lista que devemos utilizar para demarcar um glossário, quando listamos termos e seus significados. Essa lista é a **lista de definição**.

```
<dl>
  <dt>HTML</dt>
  <dd>
```

```

HTML é a linguagem de marcação de textos utilizada
para exibir textos como páginas na Internet.

</dd>
<dt>Navegador</dt>
<dd>
    Navegador é o software que requisita um documento HTML
    através do protocolo HTTP e exibe seu conteúdo em uma
    janela.
</dd>
</dl>

```

Para estilizar o formato padrão das listas ordenadas e não-ordenadas, podemos utilizar a propriedade `list-style-type` no CSS:

```

ul {
    /* alterar para circulo antes de cada <li> da lista não-ordenada */
    list-style-type: circle;
}

ol {
    /* alterar para uma sequência alfabética antes de cada <li> da lista ordenada */
    list-style-type: upper-alpha;
}

```

2.19 ESPAÇAMENTO, MARGEM E DIMENSÕES

Utilizamos a propriedade `padding` para **espaçamento**, `margin` para **margem**, `height` e `width` para alterar dimensões dos elementos. Vejamos cada uma e como elas diferem entre si.

Padding

A propriedade **padding** é utilizada para definir um espaçoamento interno em alguns elementos (por espaçoamento interno queremos dizer a distância entre o limite do elemento, sua borda, e seu respectivo conteúdo) e tem as subpropriedades listadas a seguir:

- `padding-top`
- `padding-right`
- `padding-bottom`
- `padding-left`

Essas propriedades aplicam uma distância entre o limite do elemento e seu conteúdo acima, à direita, abaixo e à esquerda respectivamente. Essa ordem é importante para entendermos como funciona a *shorthand property* do padding.

Podemos definir todos os valores para as subpropriedades do padding em uma única propriedade, chamada exatamente de `padding`, e seu comportamento é descrito nos exemplos a seguir:

Se passado somente um valor para a propriedade `padding`, esse mesmo valor é aplicado em todas as direções.

```
p {  
  padding: 10px;  
}
```

Se passados dois valores, o primeiro será aplicado acima e abaixo (equivalente a passar o mesmo valor para `padding-top` e `padding-bottom`) e o segundo será aplicado à direita e à esquerda (equivalente ao mesmo valor para `padding-right` e `padding-left`).

```
p {  
  padding: 10px 15px;  
}
```

Se passados três valores, o primeiro será aplicado acima (equivalente a `padding-top`), o segundo será aplicado à direita e à esquerda (equivalente a passar o mesmo valor para `padding-right` e `padding-left`) e o terceiro valor será aplicado abaixo do elemento (equivalente a `padding-bottom`).

```
p {  
  padding: 10px 20px 15px;  
}
```

Se passados quatro valores, serão aplicados respectivamente a `padding-top`, `padding-right`, `padding-bottom` e `padding-left`. Para facilitar a memorização dessa ordem, basta lembrar que os valores são aplicados em **sentido horário**.

```
p {  
  padding: 10px 20px 15px 5px;  
}
```

Margin

A propriedade `margin` é bem parecida com a propriedade `padding`, exceto que ela adiciona espaço após o limite do elemento, ou seja, é um espaçamento além do elemento em si. Além das subpropriedades listadas a seguir, há a *shorthand property* `margin` que se comporta da mesma maneira que a *shorthand property* do `padding` vista no tópico anterior.

- `margin-top`
- `margin-right`
- `margin-bottom`
- `margin-left`

Há ainda uma maneira de permitir que o navegador defina qual será a dimensão da propriedade `padding` ou `margin` conforme o espaço disponível na tela: definimos o valor `auto` para os espaçamentos que quisermos.

No exemplo a seguir, definimos que um elemento não tem nenhuma margem acima ou abaixo de seu conteúdo e que o navegador define uma margem igual para ambos os lados de acordo com o espaço disponível:

```
p {  
    margin: 0 auto;  
}
```

Dimensões

É possível determinar as dimensões de um elemento, por exemplo:

```
p {  
    background-color: red;  
    height: 300px;  
    width: 300px;  
}
```

Todos os parágrafos do nosso HTML ocuparão 300 pixels de altura e de largura, com cor de fundo vermelha.

2.20 EXERCÍCIOS: LISTAS E MARGENS

1. Ainda na página **sobre.html**, crie um subtítulo chamado **Diferenciais** e, logo em seguida, uma lista de diferenciais. Use `<h2>` para o subtítulo, `` para a lista e `` para os itens da lista.

Dica: você pode copiar o texto dos diferenciais do arquivo **diferenciais.txt**.

```
<h2>Diferenciais</h2>  
  
<ul>  
    <li>Menor preço do varejo, garantido</li>  
    <li>Se você achar uma loja mais barata, leva o produto de graça</li>  
    ....  
</ul>
```

Teste o resultado no navegador.

2. Podemos melhorar a apresentação da página acertando alguns espaçamentos usando várias propriedades do CSS, como `margin`, `padding` e `text-indent`.

```
h1 {  
    padding: 10px;  
}  
  
h2 {  
    margin-top: 30px;  
}  
  
p {  
    padding: 0 45px;  
    text-indent: 15px;  
    text-align: justify;  
}  
  
figure {  
    padding: 15px;  
    margin: 30px;  
}
```

```
figcaption {
    margin-top: 10px;
}
```

Veja o resultado no navegador.

3. Para centralizar o `body` como no design, podemos usar o truque de colocar um tamanho fixo e `margens auto` na esquerda e na direita:

```
body {
    margin-left: auto;
    margin-right: auto;
    width: 940px;
}
```

Verifique mais uma vez o resultado.



2.21 LINKS HTML

Quando precisamos indicar que um trecho de texto se refere a um outro conteúdo, seja ele no mesmo documento ou em outro endereço, utilizamos a tag de âncora `<a>`.

Existem três diferentes usos para as âncoras. Um deles é a definição de links:

```
<p>
    Visite o site da <a href="http://www.caelum.com.br">Caelum</a>.
</p>
```

Note que a âncora está demarcando apenas a palavra **Caelum** de todo o conteúdo do parágrafo exemplificado. Isso significa que, ao clicarmos com o cursor do mouse na palavra **Caelum**, o navegador redirecionará o usuário para o site da **Caelum**, indicado no atributo `href`.

Outro uso para a tag de âncora é a demarcação de destinos para links dentro do próprio documento, o que chamamos de *bookmark*.

```
<p>Mais informações <a href="#info">aqui</a>.</p>
<p>Conteúdo da página...</p>

<h2 id="info">Mais informações sobre o assunto:</h2>
<p>Informações...</p>
```

De acordo com o exemplo acima, ao clicarmos sobre a palavra **aqui**, demarcada com um link, o usuário será levado à porção da página onde o *bookmark info* é visível. *Bookmark* é o elemento que tem o atributo `id`.

É possível, com o uso de um link, levar o usuário a um *bookmark* presente em outra página.

```
<a href="http://www.caelum.com.br/curso/wd43/#contato">  
    Entre em contato sobre o curso  
</a>
```

O exemplo acima fará com que o usuário que clicar no link seja levado à porção da página indicada no endereço, especificamente no ponto onde o *bookmark contato* seja visível.

O outro uso para a tag de âncora é a demarcação de destinos para links dentro do próprio site, mas não na mesma página que estamos. Por exemplo, estamos na página **sobre.html** e queremos um link para a página **index.html**.

```
<p>Acesse <a href="index.html">nossa loja</a>.</p>
```

2.22 EXERCÍCIOS: LINKS

1. No segundo parágrafo do texto, citamos o centro de distribuição na cidade de *Jacarezinho, no Paraná*. Transforme esse texto em um **link externo** apontando para o mapa no Google Maps.

Use a tag `<a>` para criar link para o Google Maps:

```
<a href="https://maps.google.com.br/?q=Jacarezinho">  
    Jacarezinho, no Paraná  
</a>
```

Teste a página no navegador e acesse o link.

2. Durante o curso, vamos criar várias páginas para o site da Mirror Fashion, como uma página inicial, chamada **index.html**.

Queremos, nessa página de *Sobre* que estamos fazendo, linkar para essa nova página. Por isso, vamos criá-la agora na pasta `mirror-fashion` com a estrutura básica e um parágrafo indicando em qual página o usuário está. Não se preocupe, ela será incrementada em breve.

Crie a página **index.html**:

```
<!DOCTYPE html>  
<html>  
    <head>  
        <meta charset="utf-8">  
        <title>Mirror Fashion</title>  
    </head>  
    <body>  
        <h1>Olá, sou o index.html!</h1>  
    </body>  
</html>
```

Adicione um **link interno** na nossa **sobre.html** apontando para esta página que acabamos de criar. O terceiro parágrafo do texto possui referência a esta página mas ainda está sem link. Crie o link lá:

```
... Acesse <a href="index.html">nossa loja</a>...
```

Repare como apenas envolvemos o texto a ser linkado usando a tag `<a>`.

Veja o resultado no navegador.

3. Se reparar bem, ainda nesse terceiro parágrafo de texto há referências textuais para as outras seções da nossa página, em particular para a *História* e os *Diferenciais*. Para facilitar a navegação do usuário, podemos transformar essas referências em âncoras para as respectivas seções no HTML.

Para isso, adicione um `id` em cada um dos subtítulos para identificá-los:

```
<h2 id="historia">História</h2>  
...  
<h2 id="diferenciais">Diferenciais</h2>
```

Agora que temos os `ids` dos subtítulos preenchidos, podemos criar uma âncora para eles no terceiro parágrafo do texto:

```
... Conheça também nossa <a href="#historia">história</a> e  
nossos <a href="#diferenciais">diferenciais</a>....
```

Veja o resultado em seu navegador.

2.23 ELEMENTOS ESTRUTURAIS

Já vimos muitas tags para casos específicos como, por exemplo, `h1` para títulos, `p` para parágrafos, `img` para imagens, `ul` para listas etc. Além dessas, ainda vamos ver várias outras, mas é claro que não existe uma tag diferente para cada coisa do universo. O conjunto de tags do HTML é bem vasto mas também é limitado.

Invariavelmente você vai cair algum dia num cenário onde não consegue achar a tag certa para aquele conteúdo. Nesse caso, pode usar as tags `<div>` e ``, que funcionam como coringas. São tags sem nenhum significado especial e por padrão, não possuem estilo algum, mas podem servir para agrupar um certo conteúdo, tanto um bloco da página quanto um pedaço de texto.

2.24 CSS: SELETORES DE ID E FILHO

Já vimos como selecionar elementos no CSS usando simplesmente o nome da tag:

```
p {  
    color: red;  
}
```

Apesar de simples, é uma maneira muito limitada de selecionar. Às vezes não queremos pegar *todos* os parágrafos da página, mas apenas algum determinado.

Existem, portanto, maneiras mais avançadas de selecionarmos um ou mais elementos do HTML usando os seletores CSS. Vamos ver seletores CSS quase que ao longo do curso todo, inclusive alguns bem avançados e modernos do CSS3. Por enquanto, vamos ver mais 2 básicos além do seletor por nome de tag.

Seletor de ID

É possível aplicar propriedades visuais a um elemento selecionado pelo valor de seu atributo `id`. Para isso, o seletor deve iniciar com o caractere "#" seguido do valor correspondente.

```
#cabecalho {  
    color: white;  
    text-align: center;  
}
```

O seletor acima fará com que o elemento do nosso HTML que tem o atributo `id` com valor "cabecalho" tenha seu texto renderizado na cor branca e centralizado. Note que não há nenhuma indicação para qual tag a propriedade será aplicada. Pode ser tanto uma `<div>` quanto um `<p>`, até mesmo tags sem conteúdo como uma ``, desde que essa tenha o atributo `id` com o valor "cabecalho".

Como o atributo `id` deve ter valor único no documento, o seletor deve aplicar suas propriedades declaradas somente àquele único elemento e, por cascata, a todos os seus elementos filhos.

Seletor hierárquico

Podemos ainda utilizar um seletor hierárquico que permite aplicar estilos aos elementos filhos de um elemento pai:

```
#rodape img {  
    margin-right: 30px;  
    vertical-align: middle;  
    width: 94px;  
}
```

No exemplo anterior, o elemento pai `rodape` é selecionado pelo seu `id`. O estilo será aplicado apenas nos elementos `img` filhos do elemento com `id=rodape`.

2.25 FLUXO DO DOCUMENTO E FLOAT

Suponhamos que, por uma questão de design, a imagem da família Pelho deva vir ao lado do parágrafo e conforme a imagem abaixo:

História

A fundação em 1932 ocorreu no momento da descoberta econômica do interior do Paraná. A família Pelho, tradicional da região, investiu todas as suas economias nessa nova iniciativa, revolucionária para a época. O fundador *Eduardo Simões Pelho*, dotado de particular visão administrativa, guiou os negócios da empresa durante mais de 50 anos, muitos deles ao lado de seu filho *E. S. Pelho Filho*, atual CEO. O nome da empresa é inspirado no nome da família.

O crescimento da empresa foi praticamente instantâneo. Nos primeiros 5 anos, já atendia 18 países. Bateu a marca de 100 países em apenas 15 anos de existência. Até hoje, já atendeu 740 milhões de usuários diferentes, em bilhões de diferentes pedidos.

O crescimento em número de funcionários é também assombroso. Hoje, é a maior empregadora do Brasil, mas mesmo após apenas 5 anos de sua existência, já possuía 30 mil funcionários. Fora do Brasil, há 240 mil funcionários, além dos 890 mil brasileiros nas instalações de Jacarezinho e nos escritórios em todo país.

Dada a importância econômica da empresa para o Brasil, a família Pelho já recebeu diversos prêmios, homenagens e condecorações. Todos os presidentes do Brasil já visitaram as instalações da Mirror Fashion, além de presidentes da União Européia, Ásia e o secretário-geral da ONU.



Família Pelho

Isso não acontece por padrão. Repare que, observando as tags HTML que usamos até agora, os elementos da página são desenhados um em cima do outro. É como se cada elemento fosse uma caixa (box) e o padrão é empilhar essas caixas verticalmente. Mais pra frente vamos entender melhor esse algoritmo, mas agora o importante é que ele atrapalha esse nosso design.

Temos um problema: a tag `<figure>` ocupa toda a largura da página e aparece empilhada no **fluxo do documento**, não permitindo que outros elementos sejam adicionados ao seu lado.

Este problema pode ser solucionado por meio da propriedade **float**. Esta propriedade permite que tiremos um certo elemento do fluxo vertical do documento, o que faz com que o conteúdo abaixo dele flua ao seu redor. Na prática, vai fazer exatamente o layout que queremos.

Em nosso exemplo, o conteúdo do parágrafo tentará fluir ao redor da nossa imagem com **float**. Perceba que houve uma perturbação do fluxo HTML, pois agora a nossa imagem parece existir fora do fluxo.

2.26 EXERCÍCIOS: SELETORES CSS E FLUTUAÇÃO DE ELEMENTOS

1. Temos uma `<figure>` com a imagem do centro de distribuição que queremos centralizar na página (`margin auto`) e acertar o tamanho (`width`).

Para aplicar essas regras apenas a esse `figure` e não a todos da página, vamos usar o ID. Coloque um `id` nessa `<figure>` para podermos estilizá-la especificamente via CSS:

```
<figure id="centro-distribuicao">  
....
```

Adicionando o CSS:

```
#centro-distribuicao {  
    margin-left: auto;  
    margin-right: auto;  
    width: 550px;  
}
```

Teste no navegador. Compare o resultado com a outra figura que não recebeu o mesmo estilo.

Nosso centro de distribuição fica em [Jacarezinho, no Paraná](#). De lá, saem 48 aviões que distribuem nossos produtos às casas do mundo todo. Nossa centro de distribuição:



Compre suas roupas e acessórios na Mirror Fashion. Acesse [nossa loja](#) ou entre em contato se tiver dúvidas. Conheça também nossa [história](#) e nossos [diferenciais](#).

2. Crie um rodapé para a página utilizando a tag `<footer>`, que deve ser inserida como último elemento dentro da tag `<body>`:

```
<footer>
    
    &copy; Copyright Mirror Fashion
</footer>
```

Teste o resultado no navegador.

3. Assim como fizemos para os títulos e subtítulos, estilize o nosso rodapé. Repare no uso do `id` via CSS para selecionarmos apenas o elemento específico que será estilizado. Repare também no uso do `seletor de descendentes` para indicar um elemento que está dentro de outro.

```
footer {
    color: #777;
    margin: 30px 0;
    padding: 30px 0;
}

footer img {
    margin-right: 30px;
    vertical-align: middle;
    width: 94px;
}
```

Teste o resultado final no navegador.

- [Maior comércio eletrônico de moda do mundo](#)
- Atendimento via telefone, email, chat, twitter, facebook, carta, fax e telegrama
- Presente em 124 países
- Mais de um milhão de funcionários em todo o mundo



© Copyright Mirror Fashion

4. Queremos que a imagem da *Família Pelho* esteja flutuando a direita no texto na seção sobre a *História* da empresa. Para isso, use a propriedade `float` no CSS.

Como a `<figure>` com a imagem da família Pelho ainda não possui `id`, adicione um:

```
<figure id="familia-pelho">
```

```
....
```

Agora podemos referenciar o elemento através de seu `id` em nosso CSS, indicando a flutuação e uma margem para espaçamento:

```
#familia-pelho {  
    float: right;  
    margin: 0 0 10px 10px;  
}
```

Teste o resultado. Repare como o texto é renderizado *ao redor* da imagem, bem diferente de antes.

História

A fundação em 1932 ocorreu no momento da descoberta econômica do interior do Paraná. A família Pelho, tradicional da região, investiu todas as suas economias nessa nova iniciativa, revolucionária para a época. O fundador *Eduardo Simões Pelho*, dotado de particular visão administrativa, guiou os negócios da empresa durante mais de 50 anos, muitos deles ao lado de seu filho *E. S. Pelho Filho*, atual CEO. O nome da empresa é inspirado no nome da família.

O crescimento da empresa foi praticamente instantâneo. Nos primeiros 5 anos, já atendia 18 países. Bateu a marca de 100 países em apenas 15 anos de existência. Até hoje, já atendeu 740 milhões de usuários diferentes, em bilhões de diferentes pedidos.

O crescimento em número de funcionários é também assombroso. Hoje, é a maior empregadora do Brasil, mas mesmo após apenas 5 anos de sua existência, já possuía 30 mil funcionários. Fora do Brasil, há 240 mil funcionários, além dos 890 mil brasileiros nas instalações de Jacarezinho e nos escritórios em todo país.

Dada a importância econômica da empresa para o Brasil, a família Pelho já recebeu diversos prêmios, homenagens e condecorações. Todos os presidentes do Brasil já visitaram as instalações da Mirror Fashion, além de presidentes da União Europeia, Ásia e o secretário-geral da ONU.



Família Pelho

5. (opcional) Faça testes com o `float: left` também.

6. (opcional) Teste flutuar a imagem do centro de distribuição. Como o conteúdo fluirá ao seu redor agora? É o que queríamos? Como melhorar?

2.27 O FUTURO E PRESENTE DA WEB COM O HTML5

Nos últimos anos, muito tem se falado sobre a versão mais recente do HTML, o HTML5. Esse projeto é um grande esforço do W3C e dos principais browsers para atender a uma série de necessidades do desenvolvimento da Web como plataforma de sistemas distribuídos e informação descentralizada. Algumas novidades são importantes para a marcação de conteúdo, outras para a estilização com o CSS nível 3 (CSS3) e outras novidades são importantes para interação avançada com o usuário com novas funcionalidades do navegador com JavaScript.

Apesar da especificação já estar completa, existem diferenças entre as implementações adotadas pelos diferentes navegadores ainda hoje. Mesmo assim, o mercado está tomando uma posição bem agressiva quanto à adoção dos novos padrões e hoje muitos projetos já são iniciados com eles.

Em nosso projeto, vamos adotar os padrões do HTML5 e vamos conhecer e utilizar algumas de suas novidades quanto à melhoria da semântica de conteúdo e novas propriedades de CSS que nos permite adicionar efeitos visuais antes impossíveis. Ainda assim, nosso projeto será parcialmente compatível com navegadores obsoletos por conta da técnica *Progressive Enhancement*.

HTML SEMÂNTICO E POSICIONAMENTO NO CSS

"O caos é a rima do inaudito." -- Zack de la Rocha

3.1 O PROCESSO DE DESENVOLVIMENTO DE UMA TELA

Existe hoje no mercado uma grande quantidade de empresas especializadas no desenvolvimento de sites e aplicações web, bem como algumas empresas de desenvolvimento de software ou agências de comunicação que têm pessoas capacitadas para executar esse tipo de projeto.

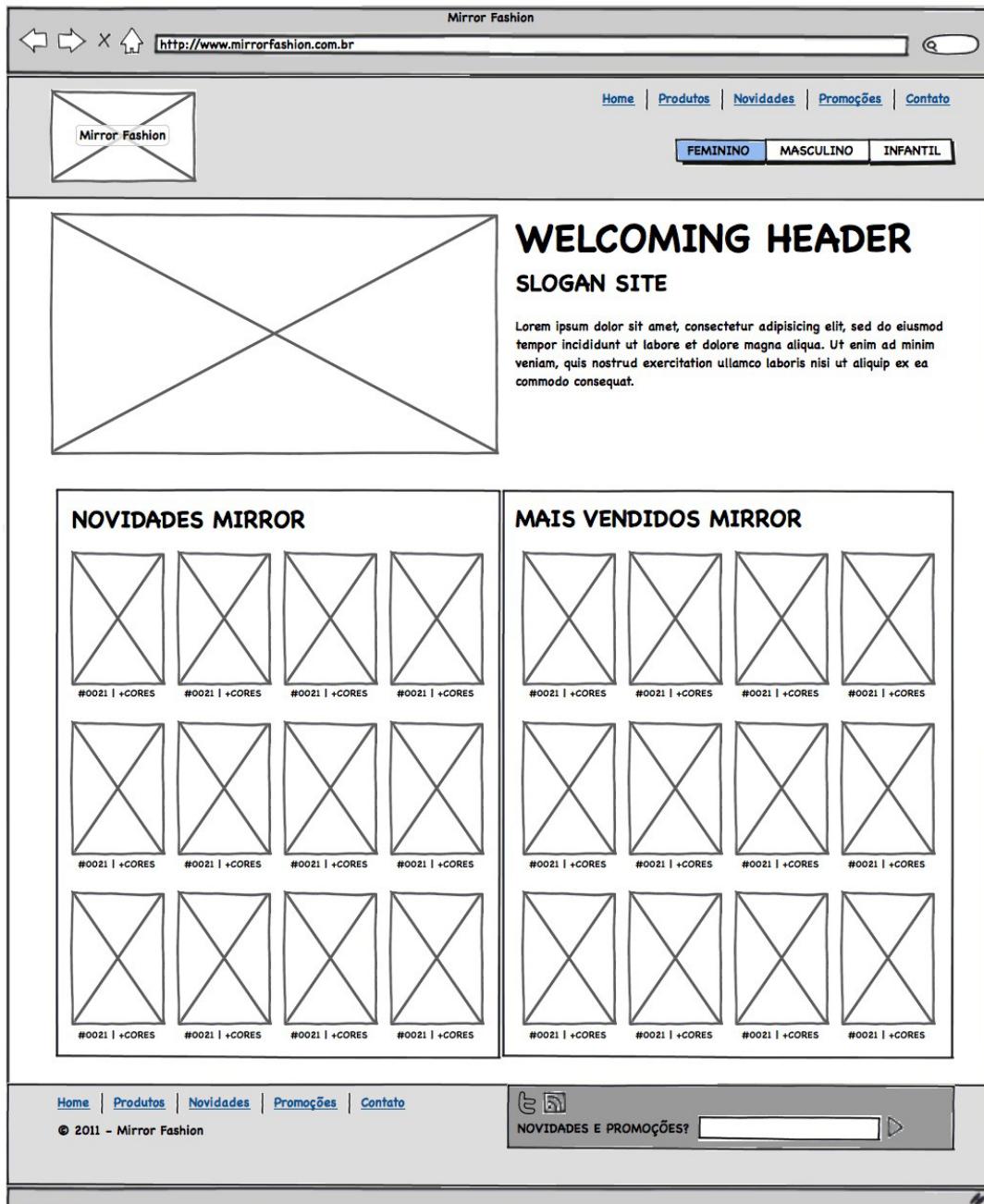
Quando detectada a necessidade do desenvolvimento de um site ou aplicação web, a empresa que tem essa necessidade deve passar todas as informações relevantes ao projeto para a empresa que vai executá-lo. A empresa responsável pelo seu desenvolvimento deve analisar muito bem essas informações e utilizar pesquisas para complementar ou mesmo certificar-se da validade dessas informações.

Um projeto de site ou aplicação web envolve muitas disciplinas em sua execução, pois são diversas características a serem analisadas e diversas as possibilidades apresentadas pela plataforma. Por exemplo, devemos conhecer muito bem as características do público alvo, pois ele define qual a melhor abordagem para definir a navegação, tom linguístico e visual a ser adotado, entre outras. A afinidade do público com a Internet e o computador pode inclusive definir o tipo e a intensidade das inovações que podem ser utilizadas.

Por isso, a primeira etapa do desenvolvimento do projeto fica a cargo da área de User Experience Design (UX) ou Interaction Design (IxD), normalmente composta de pessoas com formação na área de comunicação. Essa equipe, ou pessoa, analisa e endereça uma série de informações da característica humana do projeto, definindo a quantidade, conteúdo e localização de cada informação.

Algumas das motivações e práticas de Design de Interação e Experiência do Usuário são conteúdo do curso **Design de Interação, Experiência do Usuário e Usabilidade**. O resultado do trabalho dessa equipe é uma série de definições sobre a navegação (mapa do site) e um esboço de cada uma das visões, que são as páginas, e visões parciais como, por exemplo, os diálogos de alerta e confirmação da aplicação. Essas visões não adotam nenhum padrão de design gráfico: são utilizadas fontes, cores e imagens neutras, embora as informações escritas devam ser definidas nessa fase do projeto.

Esses esboços das visões são o que chamamos de **wireframes** e guiam o restante do processo de design.



Com os wireframes em mãos, é hora de adicionar as imagens, cores, tipos, fundos, bordas e outras características visuais. Esse trabalho é realizado pelos designers gráficos, que utilizam ferramentas gráficas como Adobe Photoshop, Adobe Fireworks, GIMP, entre outras. O que resulta desse trabalho que o designer realiza em cada wireframe é o que chamamos de **layout**. Os layouts são imagens estáticas já com o visual completo a ser implementado. Apesar de os navegadores serem capazes de exibir imagens estáticas, exibir uma única imagem para o usuário final no navegador não é a forma ideal

de se publicar uma página.

Para que as informações sejam exibidas de forma correta e para possibilitar outras formas de uso e interação com o conteúdo, é necessário que a equipe de **programação front-end** transforme essas imagens em telas visíveis e, principalmente, utilizáveis pelos navegadores. Existem diversas tecnologias e ferramentas para realizar esse tipo de trabalho. Algumas das tecnologias disponíveis são: HTML, Adobe Flash, Adobe Flex, JavaFX e Microsoft Silverlight.

De todas as tecnologias disponíveis, a mais recomendada é certamente o HTML, pois é a linguagem que o navegador entende. Todas as outras tecnologias citadas dependem do HTML para serem exibidas corretamente no navegador e, ultimamente, o uso do HTML, em conjunto com o CSS e o JavaScript, tem evoluído a ponto de podermos substituir algumas dessas outras tecnologias onde tínhamos mais poder e controle em relação à exibição de gráficos, efeitos e interatividade.

3.2 O PROJETO MIRROR FASHION

Durante o curso, vamos produzir algumas páginas de um projeto: um e-commerce de roupas. No capítulo anterior, de introdução, criamos uma página simples de Sobre. Vamos começar agora a projetar o restante, com as páginas mais complexas.

Uma equipe de UX já definiu as páginas, o conteúdo de cada uma delas e produziu alguns *wireframes*. Depois de realizado esse trabalho, a equipe de design já adicionou o visual desejado pelo cliente como resultado final do projeto.

Agora é a nossa vez de **transformar esse layout em HTML**, para que os navegadores possam ler e renderizar o código, exibindo a página para o usuário final.

No capítulo anterior, começamos a codificar a página de **Sobre** da nossa loja, com o intuito de praticar o básico de HTML e CSS.

Nesse momento, vamos focar na construção da parte principal da loja, a **Home Page**, e seguiremos o layout oficial criado pela equipe de design:

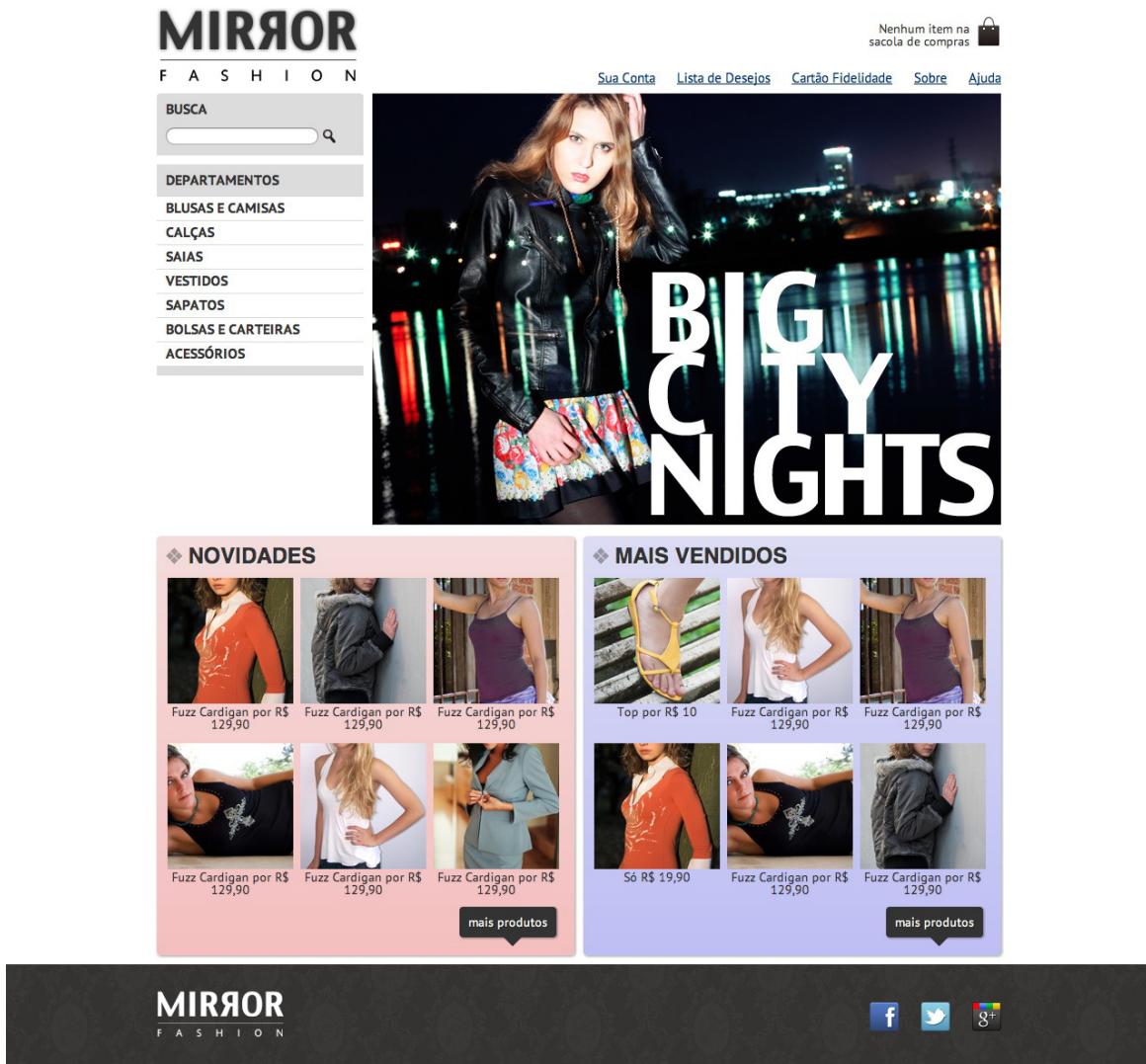


Figura 3.2: Design da Homepage

3.3 ANALISANDO O LAYOUT

Antes de digitar qualquer código, é necessária uma análise do layout. Com essa análise, definiremos as principais áreas de nossas páginas. Um fator muito importante a ser considerado quando fazemos essa análise do layout é o modo como os navegadores interpretam e renderizam o HTML.

O HTML é exibido no navegador de acordo com a *ordem de leitura do idioma da página*. Na maioria dos casos, a leitura é feita da esquerda para a direita, de cima para baixo, da mesma maneira que lemos essa apostila, por exemplo.

Olhe detalhadamente nosso layout e tente imaginar qual a melhor maneira de estruturar nosso HTML para que possamos codificá-lo.

De acordo com o posicionamento de elementos que foi definido desde a etapa de criação dos *wireframes*, todas as páginas no nosso projeto obedecem a uma estrutura similar.

Estrutura da página

Note que há um cabeçalho (uma área que potencialmente se repetirá em mais de uma página) que ocupa uma largura fixa; sendo assim, podemos criar uma seção exclusiva para o cabeçalho.



Outra área que tem uma característica semelhante é o rodapé, pois pode se repetir em mais de uma página. Podemos notar que o fundo do elemento vai de uma ponta à outra da página, porém seu conteúdo segue a mesma largura fixa do restante da página.



A área central, que contém informações diferentes em cada página, não tem nenhum elemento ao fundo. Porém, notemos que sua largura é limitada antes de atingir o início e o fim da página. Notemos que, apesar do fundo do rodapé ir de uma ponta à outra, seu conteúdo também é limitado pela mesma largura do conteúdo.

No caso da home page, o miolo da página pode ainda ser visto como dois blocos diferentes. Há o bloco principal inicial com o menu de navegação e o banner de destaque. E há outro bloco no final com dois painéis com listas de produtos.

Poderíamos definir essas áreas da seguinte maneira:

```
<body>
  <header>
    <!-- Conteúdo do cabeçalho -->
  </header>

  <main>
    <!-- Conteúdo principal -->
  </main>

  <section id="destaques">
    <!-- Painéis com destaque -->
  </section>

  <footer>
    <!-- Conteúdo do rodapé -->
  </footer>
</body>
```

Note que utilizamos o atributo `id` do HTML para identificar a `<section>` de destaque. O atributo `id` deve ser único em cada página, ou seja, só pode haver **um** elemento com o atributo `id="destaques"`. Mesmo se o outro elemento for de outra tag, o `id` não pode se repetir. De acordo com a estrutura acima, nós separamos as quatro áreas principais.

3.4 HTML SEMÂNTICO

As tags que usamos antes - `header` e `footer` - são tags novas do HTML5. Antigamente, numa situação parecida com essa, teríamos criado quatro `div`, uma para cada parte da página, e talvez colocado ids diferentes pra cada uma.

Qual a diferença entre colocar `div` e essas novas tags do HTML5? Visualmente e funcionalmente, nenhuma. A única diferença é o nome da tag e o significado que elas carregam. **E isso é bastante importante.**

Dizemos que a função do HTML é fazer a marcação do conteúdo da página, representar sua estrutura da informação. Não é papel do HTML, por exemplo, cuidar da apresentação final e dos detalhes de design, pois isto é o papel do CSS. O HTML precisa ser **claro e limpo**, focado em marcar o conteúdo.

As novas tags do HTML5 trazem novos **significados semânticos** para usarmos em elementos HTML. Em vez de simplesmente agrupar os elementos do cabeçalho em um `div` genérico e sem significado, usamos uma tag `header` que carrega em si o significado de representar um cabeçalho.

Com isso, temos um HTML com estrutura baseada no significado de seu conteúdo, o que traz uma série de benefícios, como a facilidade de manutenção e compreensão do documento.

Provavelmente, o design da sua página deixa bastante claro qual parte da sua página é o cabeçalho e qual parte é o menu de navegação. Visualmente, são distinguíveis para o usuário comum. Mas e se desabilitarmos o CSS e as regras visuais? Como distinguir esses conteúdos?

Um HTML semântico carrega significado independente da sua apresentação visual. Isso é particularmente importante quando o conteúdo for consumido por clientes não visuais. Há vários desses cenários. Um usuário cego poderia usar um leitor de tela para ouvir sua página. Neste caso, a estrutura semântica do HTML é essencial para ele entender as partes do conteúdo.

Mais importante ainda, robôs de busca como o Google também são leitores não visuais da sua página. Sem um HTML semântico, o Google não consegue, por exemplo, saber que aquilo é um menu e que deve seguir seus links. Ou que determinada parte é só um rodapé informativo, mas não faz parte do conteúdo principal. Semântica é uma importante técnica de SEO - **Search Engine Optimization** - e crítica para marketing digital.

Vamos falar bastante de semântica ao longo do curso e esse é um ingrediente fundamental das técnicas modernas de web design. Veremos mais cenários onde uma boa semântica é essencial.

3.5 PENSANDO NO HEADER

Já sabemos que o nosso cabeçalho será representado pela tag `<header>` do HTML5, semanticamente perfeita para a situação. Mas e o conteúdo dele?

Observe apenas o cabeçalho no layout anterior. Quais blocos de conteúdo você identifica nele?

- O logo principal com o nome da empresa
- Uma mensagem sobre a sacola de compras
- Uma lista de links de navegação da loja

Repare como não destacamos a presença do ícone da sacola. Ele não faz parte do conteúdo, é meramente *decorativo*. O *conteúdo* é a mensagem sobre os itens na sacola. Que tipo de conteúdo é esse? Qual tag usar? É apenas uma frase informativa, um parágrafo de texto. Podemos usar `<p>`:

```
<p>
    Nenhum item na sacola de compras
</p>
```

Mas e a imagem com o ícone? Como é decorativa, pertence ao CSS, como veremos depois. O HTML não tem nada a ver com isso.

Continuando no header, nossa lista de links pode ser uma lista - `` com ``s. Dentro de cada item, vamos usar um link - `<a>` - para a página correspondente. Mas há como melhorar ainda mais: esses links não são links ordinários, são essenciais para a navegação do usuário na página. Podemos sinalizar isso com a nova tag `<nav>` do HTML5, que representa blocos de navegação primários:

```
<nav>
  <ul>
    <li><a href="#">Sua Conta</a></li>
    <li><a href="#">Lista de Desejos</a></li>
    <li><a href="#">Cartão Fidelidade</a></li>
    <li><a href="sobre.html">Sobre</a></li>
    <li><a href="#">Ajuda</a></li>
  </ul>
</nav>
```

O último ponto para fecharmos nosso cabeçalho é o logo. Como representá-lo?

Visualmente, observamos no layout que é apenas uma imagem. Podemos usar então uma tag `` como fizemos antes. Mas e semanticamente, o que é aquele conteúdo? E, principalmente, o que significa aquele logo para clientes não visuais? Como gostaríamos que esse conteúdo fosse indexado no Google?

É muito comum, nesse tipo de situação, usar um `<h1>` com um texto que represente o título da nossa página. Se pensarmos bem, o que queremos passar com o logo é a ideia de que a página é da

Mirror Fashion .

Quando o texto for lido para um cego, queremos essa mensagem lida. Quando o Google indexar, queremos que ele associe nossa página com Mirror Fashion e não com uma imagem "qualquer".

É fácil obter esse resultado colocando a `` dentro do `<h1>`. E para representar o conteúdo textual da imagem (o que vai ser usado pelo leitor de tela e pelo Google), usamos o atributo `alt` da imagem. Esse atributo indica **conteúdo alternativo**, que será usado quando o cliente não for visual e não conseguir enxergar a imagem visualmente.

```
<h1>
  
</h1>
```

Repare como a colocação do H1 e do ALT na imagem não alteram em nada a página visualmente. Estão lá por pura **importância semântica**. E isso é muito bom. O H1 dará o devido destaque semântico para o logo, colocando-o como elemento principal. E o ALT vai designar um conteúdo textual alternativo à imagem.

3.6 ESTILIZAÇÃO COM CLASSES

Para podermos estilizar os elementos que criamos, vamos precisar de uma forma de selecionarmos no CSS cada coisa. Já vimos seletor de tag e por ID. Ou seja, pra estilizar nosso menu `<nav>`, podíamos fazer:

```
nav {
  ...
}
```

Mas imagine que podemos ter muitos NAV na página e queremos ser mais específicos. O ID é uma solução:

```
<nav id="menu-opcoes">
</nav>
```

E, no CSS:

```
#menu-opcoes {
  ...
}
```

Vamos ver uma terceira forma, com o uso de classes. O código é semelhante de quando usamos o atributo `id`, mas agora vamos usar o atributo `class` no HTML e o ponto no CSS:

```
<nav class="menu-opcoes">
  ...
</nav>
```

E, no CSS:

```
.menu-opcoes {  
    ...  
}
```

Mas quando usar ID ou CLASS?

Ambos fariam seu trabalho nesse caso. Mas é bom lembrar que ids são mais fortes, devem ser únicos na página, sempre. Embora esse nosso menu seja único agora, imagine se, no futuro, quisermos ter o mesmo menu em outro ponto da página, mais pra baixo? **Usar classes facilita reuso de código e flexibilidade.**

Além disso, um elemento pode ter *mais de uma classe ao mesmo tempo*, aplicando estilos de várias regras do CSS ao mesmo tempo:

```
<nav class="menu-opcoes menu-cabecalho">  
    ...  
</nav>  
  
.menu-opcoes {  
    // código de um menu de opções  
    // essas regras serão aplicadas ao nav  
}  
  
.menu-cabecalho {  
    // código de um menu no cabeçalho  
    // essas regras TAMBÉM serão aplicadas ao nav  
}
```

No caso do ID, não. Cada elemento só tem um id, único.

Preferimos o uso de classes, no lugar de ids, pra deixar em aberto a possibilidade de reaproveitar aquele elemento em mais de um ponto do código. Vamos fazer o uso de classe na sacola do cabeçalho:

```
<p class="sacola">  
    Nenhum item na sacola de compras  
</p>
```

Reutilizando uma classe para diversos elementos

Pode ser interessante criar uma classe que determina a centralização horizontal de qualquer elemento, sem interferir em suas margens superior e inferior como no exemplo a seguir:

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

Agora, é só adicionar a classe "container" ao elemento, mesmo que ele já tenha outros valores para esse atributo:

```
<div class="info container">  
    <p>Conteúdo que deve ficar centralizado</p>  
</div>
```

3.7 EXERCÍCIOS: HEADER SEMÂNTICO

1. Já temos o arquivo **index.html*** criado. Vamos apagar seu único parágrafo, pois adicionaremos conteúdo que fará sentido.

Crie o arquivo **estilos.css** na pasta **css** do projeto, que será onde escreveremos todos os estilos da página. Adicione na **index.html** a tag `<link>` apontando para **css/estilos.css**:

```
<head>
  <meta charset="utf-8">
  <title>Mirror Fashion</title>
  <link rel="stylesheet" href="css/estilos.css">
</head>
```

2. Próximo passo: criar o cabeçalho. Use as tags semânticas que vimos no curso, como `<header>`, `<nav>`, ``, ``, etc. Crie links para as páginas do menu. E use `<h1>` para representar o título da página com o logo acessível.

```
<body>
  <header>
    <h1></h1>

    <p class="sacola">
      Nenhum item na sacola de compras
    </p>

    <nav class="menu-opcoes">
      <ul>
        <li><a href="#">Sua Conta</a></li>
        <li><a href="#">Lista de Desejos</a></li>
        <li><a href="#">Cartão Fidelidade</a></li>
        <li><a href="sobre.html">Sobre</a></li>
        <li><a href="#">Ajuda</a></li>
      </ul>
    </nav>
  </header>
</body>
```

3. Já podemos testar no navegador. Repare como a página, embora sem estilo visual, é **utilizável**. É assim que os clientes não visuais lerão nosso conteúdo. Qual a importância de uma marcação semântica?
4. Vamos criar a estilização visual básica do nosso conteúdo, sem nos preocuparmos com posicionamento ainda. Ajuste as cores e alinhamento dos textos. Coloque o ícone da sacola com CSS através de uma imagem de fundo do parágrafo:

```
.sacola {
  background-image: url(..../img/sacola.png);
  background-repeat: no-repeat;
  background-position: top right;
  font-size: 14px;
  text-align: right;
  width: 140px;
  padding-right: 35px;
}
```

```
.menu-opcoes ul {  
    font-size: 15px;  
}  
  
.menu-opcoes a {  
color: #003366;  
}
```

Aproveite e configure a cor e a fonte base de todos os textos do site, usando um seletor direto na tag <body> :

```
body {  
color: #333333;  
font-family: "Lucida Sans Unicode", "Lucida Grande", sans-serif;  
}
```

Teste no navegador e veja como nossa página começa a pegar o design.



3.8 CSS RESET

Quando não especificamos nenhum estilo para nossos elementos do HTML, o navegador utiliza uma série de estilos padrão, que são diferentes em cada um dos navegadores. Em um momento mais avançado dos nossos projetos, poderemos enfrentar problemas com coisas que não tínhamos previsto; por exemplo, o espaçamento entre caracteres utilizado em determinado navegador pode fazer com que um texto que, pela nossa definição deveria aparecer em 4 linhas, apareça com 5, quebrando todo o nosso layout.

Para evitar esse tipo de interferência, alguns desenvolvedores e empresas criaram alguns estilos que chamamos de **CSS Reset**. A intenção é setar um valor básico para todas as características do CSS, sobrescrevendo totalmente os estilos padrão do navegador.

Desse jeito podemos começar a estilizar as nossas páginas a partir de um ponto que é o mesmo para todos os casos, o que nos permite ter um resultado muito mais sólido em vários navegadores.

Existem algumas opções para resetar os valores do CSS. Algumas que merecem destaque hoje são as seguintes:

HTML5 Boilerplate

O HTML5 Boilerplate é um projeto que pretende fornecer um excelente ponto de partida para quem pretende desenvolver um novo projeto com HTML5. Uma série de técnicas para aumentar a compatibilidade da nova tecnologia com navegadores um pouco mais antigos estão presentes e o código é totalmente gratuito. Em seu arquivo "style.css", estão reunidas diversas técnicas de CSS Reset. Apesar de consistentes, algumas dessas técnicas são um pouco complexas, mas é um ponto de partida que podemos considerar.

YUI3 CSS Reset

Criado pelos desenvolvedores front-end do Yahoo!, uma das referências na área, esse CSS Reset é composto de 3 arquivos distintos. O primeiro deles, chamado de **Reset**, simplesmente muda todos os valores possíveis para um valor padrão, onde até mesmo as tags `<h1>` e `<small>` passam a ser exibidas com o mesmo tamanho. O segundo arquivo é chamado de **Base**, onde algumas margens e dimensões dos elementos são padronizadas. O terceiro é chamado de **Font**, onde o tamanho dos tipos é definido para que tenhamos um visual consistente inclusive em diversos dispositivos móveis.

Eric Meyer CSS Reset

Há também o famoso CSS Reset de Eric Meyer, que pode ser obtido em <http://meyerweb.com/eric/tools/css/reset/>. É apenas um arquivo com tamanho bem reduzido.

3.9 BLOCK VS INLINE

Os elementos do HTML, quando renderizados no navegador, podem comportar-se basicamente de duas maneiras diferentes no que diz respeito à maneira como eles interferem no documento como um todo: em *bloco* (block) ou em *linha* (inline).

Elementos em bloco são aqueles que ocupam toda a largura do documento, tanto antes quanto depois deles. Um bom exemplo de elemento em bloco é a tag `<h1>`, que já utilizamos em nosso projeto. Note que não há nenhum outro elemento à esquerda ou à direita do nosso nome da loja, apesar da expressão "Mirror Fashion" não ocupar toda a largura do documento.

Entre os elementos em bloco, podemos destacar as tags de heading `<h1>` a `<h6>`, os parágrafos `<p>` e divisões `<div>`.

Elementos em linha são aqueles que ocupam somente o espaço necessário para que seu próprio conteúdo seja exibido, permitindo que outros elementos em linha possam ser renderizados logo na sequência, seja antes ou depois, exibindo diversos elementos nessa mesma linha.

Entre os elementos em linha, podemos destacar as tags de âncora `<a>`, as tags de ênfase `<small>`, `` e `` e a tag de marcação de atributos ``.

Saber a distinção entre esses modos de exibição é importante, pois há diferenças na estilização dos elementos dependendo do seu tipo.

Pode ser interessante alterarmos esse padrão de acordo com nossa necessidade, por isso existe a propriedade `display` no CSS, que permite definir qual estratégia de exibição o elemento utilizará.

Por exemplo, o elemento `` de uma `` tem por padrão o valor `block` para a propriedade `display`. Se quisermos os elementos na horizontal, basta alterarmos a propriedade `display` da `` para `inline`:

```
ul li {  
    display: inline;  
}
```

3.10 EXERCÍCIOS: RESET E DISPLAY

- Utilizaremos o CSS reset do Eric Meyer. O arquivo `reset.css` já foi copiado para a pasta `css` do nosso projeto quando importamos o projeto no capítulo inicial.

Precisamos só referenciá-lo no head **antes** do nosso `estilos.css`:

```
<head>  
    <meta charset="utf-8">  
    <title>Mirror Fashion</title>  
    <link rel="stylesheet" href="css/reset.css">  
    <link rel="stylesheet" href="css/estilos.css">  
</head>
```

Abra novamente a página no navegador. Observe a diferença, principalmente na padronização dos espaçamentos.

- Próximo passo: transformar nosso menu em horizontal e ajustar espaçamentos básicos.

Vamos usar a propriedade `display` para mudar os `` para `inline`. Aproveite e já coloque um espaçamento entre os links com `margin`.

Repare também como a `sacola` está desalinhada. O texto está muito pra cima e não alinhado com a base do ícone. Um `padding-top` deve resolver.

```
.menu-opcoes ul li {  
    display: inline;  
    margin-left: 20px;  
}  
  
.sacola {  
    padding-top: 8px;  
}
```

Teste a página no navegador. Está melhorando?

- Nosso header ainda está todo à esquerda da página, sendo que no layout ele tem um tamanho fixo e fica centralizado na página. Aliás, não é só o cabeçalho que fica assim: o conteúdo da página em si e o conteúdo do rodapé também.

Temos três tipos de elementos que precisam ser centralizados no meio da página. Vamos copiar e colar as instruções CSS nos 3 lugares? Não! Criamos uma classe no HTML a ser aplicada em todos esses pontos e um único seletor no CSS.

```
.container {  
    margin: 0 auto;  
    width: 940px;  
}
```

Vamos usar essa classe `container` no HTML também. Altere a tag `header` e passe o `class="container"` para ela:

```
<header class="container">
```

Teste a página e veja o conteúdo centralizado. Agora, falta "somente" o posicionamento dos elementos do header.



3.11 POSITION: STATIC, RELATIVE, ABSOLUTE E FIXED

Existe um conjunto de propriedades que podemos utilizar para posicionar um elemento na página, que são `top`, `left`, `bottom` e `right`. Porém essas propriedades, por padrão, não são obedecidas por nenhum elemento, pois elas dependem de uma outra propriedade, a `position`.

A propriedade `position` determina qual é o modo de posicionamento de um elemento, e ela pode receber como valor: **static**, **relative**, **absolute** ou **fixed**. Veremos o comportamento de cada um deles, junto com as propriedades de coordenadas.

O primeiro valor, padrão para todos os elementos, é o **static**. Um elemento com posição `static` permanece sempre em seu local original no documento, aquele que o navegador entende como sendo sua posição de renderização. Se passarmos valores para as propriedades de coordenadas, eles não serão respeitados.

Um dos valores para a propriedade `position` que aceitam coordenadas é o **relative**. Com ele, as coordenadas que passamos são obedecidas em relação à posição original do elemento. Por exemplo:

```
.logotipo {  
    position: relative;  
    top: 20px;  
    left: 50px;  
}
```

Os elementos em nosso documento que receberem o valor "logotipo" em seu atributo `class` terão 20px adicionados ao seu topo e 50px adicionados à sua esquerda em relação à sua posição original. Note que, ao definirmos coordenadas, estamos adicionando pixels de distância naquela direção, então o elemento será renderizado mais abaixo e à direita em comparação à sua posição original.

O próximo modo de posicionamento que temos é o **absolute**, e ele é um pouco complexo. Existem algumas regras que alteram seu comportamento em determinadas circunstâncias. Por definição, o elemento que tem o modo de posicionamento absolute toma como referência qualquer elemento que seja seu pai na estrutura do HTML cujo modo de posicionamento seja diferente de `static` (que é o padrão), e obedece às coordenadas de acordo com o tamanho total desse elemento pai.

Quando não há nenhum elemento em toda a hierarquia daquele que recebe o posicionamento `absolute` que seja diferente de `static`, o elemento vai aplicar as coordenadas tendo como referência a porção visível da página no navegador. Por exemplo:

Estrutura HTML

```
<div class="quadrado"></div>  
<div class="quadrado absoluto"></div>
```

Estilos CSS

```
.quadrado {  
    background-color: green;  
    height: 200px;  
    width: 200px;  
}  
  
.absoluto {  
    position: absolute;  
    top: 20px;  
    right: 30px;  
}
```

Seguindo o exemplo acima, o segundo elemento `<div>`, que recebe o valor "absoluto" em seu atributo `class`, não tem nenhum elemento como seu "pai" na hierarquia do documento, portanto ele vai alinhar-se ao topo e à direita do limite visível da página no navegador, adicionando respectivamente 20px e 30px nessas direções. Vamos analisar agora o exemplo a seguir:

Estrutura HTML

```
<div class="quadrado relativo">  
    <div class="quadrado absoluto"></div>  
</div>
```

Estilos CSS

```

.quadrado {
    background-color: green;
    height: 200px;
    width: 200px;
}

.absoluto {
    position: absolute;
    top: 20px;
    right: 30px;
}

.relativo {
    position: relative;
}

```

Nesse caso, o elemento que recebe o posicionamento `absolute` é "filho" do elemento que recebe o posicionamento `relative` na estrutura do documento, portanto, o elemento `absolute` vai usar como ponto de referência para suas coordenadas o elemento `relative` e se posicionar 20px ao topo e 30px à direita da **posição original** desse elemento.

O outro modo de posicionamento, **fixed**, sempre vai tomar como referência a porção visível do documento no navegador, e mantém essa posição inclusive quando há rolagem na tela. É uma propriedade útil para avisos importantes que devem ser visíveis com certeza.

Um resumo de position

static

- Sua posição é dada automaticamente pelo fluxo da página: por padrão ele é renderizado logo após seus irmãos
- Não aceita um posicionamento manual (left, right, top, bottom)
- O tamanho do seu elemento pai leva em conta o tamanho do elemento static

relative

- Por padrão, o elemento será renderizado da mesma maneira que o static
- Aceita posicionamento manual
- O tamanho do seu elemento pai leva em conta o tamanho do elemento relative, porém sem levar em conta seu posicionamento. O pai não sofreria alterações mesmo se o elemento fosse static

fixed

- Uma configuração de posicionamento vertical (top ou bottom) e uma horizontal (left ou right) é obrigatória
- O elemento será renderizado na página na posição indicada. Mesmo que ocorra uma rolagem, o elemento permanecerá no mesmo lugar
- Seu tamanho não conta para calcular o tamanho do elemento pai, é como se não fosse elemento

filho

absolute

- Uma configuração de posicionamento vertical (top ou bottom) e uma horizontal (left ou right) é obrigatória
- O elemento será renderizado na posição indicada, porém relativa ao primeiro elemento pai cujo position seja diferente de static ou, se não existir este pai, relativa à página
- Seu tamanho não conta para calcular o tamanho do elemento pai

3.12 EXERCÍCIOS: POSICIONAMENTO

1. Posicione o menu à direita e embaixo no header. Use `position: absolute` para isso. E não esqueça: se queremos posicioná-lo absolutamente *com relação* ao cabeçalho, o cabeçalho precisa estar com `position: relative`.

```
header {  
    position: relative;  
}  
  
.menu-opcoes {  
    position: absolute;  
    bottom: 0;  
    right: 0;  
}
```

2. A sacola também deve estar posicionada a direita e no topo. Use `position`, `top` e `right` para conseguir esse comportamento. Adicione as regras de posicionamento ao seletor `.sacola` que já tínhamos:

```
.sacola {  
    position: absolute;  
    top: 0;  
    right: 0;  
}
```

3. Teste a página no navegador. Como está nosso cabeçalho? De acordo com o design original?



3.13 PARA SABER MAIS: SUPORTE HTML5 NO INTERNET EXPLORER ANTIGO

A partir do IE9, há um bom suporte a HTML5, até para elementos avançados como os de multimídia. Entretanto, até o IE8 (incluindo as versões 6 e 7), as tags do HTML5 não são reconhecidas.

Se você abrir nossa página no IE8, verá o design todo quebrado pois as tags de header, footer, nav, section, etc. são ignoradas. Mas é possível corrigir o suporte a esses novos elementos semânticos.

A solução envolve um hack descoberto no IE e chamado de **html5shiv**. Há um projeto opensource com o código disponível para download:

<https://github.com/afarkas/html5shiv>

Para incluir a correção na nossa página, basta adicionar no header:

```
<!--[if lt IE 9]>
<script src="//html5shiv.googlecode.com/svn/trunk/html5.js"></script>
<![endif]-->
```

Repare que isso carrega um JavaScript que acionará o hack. Mas a tag `script` está dentro de um bloco com uma espécie de if dentro de um comentário!

Esse recurso do IE é chamado de **comentários condicionais** e nos permite adicionar código que somente será lido pelo IE -- uma excelente solução para resolver seus bugs e inconsistências sem estragar a página nos demais navegadores.

Nesse caso, estamos dizendo que o tal script com o hack só deve ser carregado pelas versões anteriores ao IE9; já que, a partir desta versão, há suporte nativo a HTML5 e não precisa de hacks.

IE6 e IE7

Ao testar nesses navegadores muito抗igos, você verá que apenas o HTML5shiv não é suficiente. Na verdade, vários recursos e técnicas que usamos no nosso CSS não eram suportados nas versões抗igas.

Felizmente, o uso de IE6 e IE7 no Brasil é bastante baixo e cai cada vez mais - hoje já é menos de 1% dos usuários. Na Caelum, já não suportamos mais essas versões em nosso curso e nem em nossos sites.

3.14 EXERCÍCIOS OPCIONAIS

1. Aplique nosso novo cabeçalho também na página **sobre.html**. Cuidado que teremos algumas incompatibilidades com o código anterior que precisaremos rever.
2. Adicione suporte ao IE8 na nossa página usando o HTML5shiv.

MAIS HTML E CSS

"O medo é o pai da moralidade" -- Friedrich Wilhelm Nietzsche

4.1 ANALISANDO O MIOLO DA PÁGINA

Elaboramos o cabeçalho, mas ainda resta a área central e o rodapé. Focaremos agora nessa área central.

A área central possui duas áreas distintas: o bloco principal inicial, com o menu de navegação e o banner de destaque, e o bloco com os painéis com destaques de produtos.

Na área de navegação, teremos um formulário de busca, permitindo que o usuário busque por produtos.

4.2 FORMULÁRIOS

Em HTML, temos um elemento `<form>` criado para capturar os dados do usuário e submetê-lo a algum serviço da Internet.

Como filho da tag `<form>`, podemos usar as tags `<input>` e `<button>`, dentre várias outras.

Os dados são passados para o `<form>` por meio da tag `<input>`, que recebe os dados digitados. É por meio do atributo `type` que definimos a finalidade desse **input**. Em nosso caso, utilizaremos o tipo `search` para capturar os dados digitados. Esses dados são enviados pela tag `<button>`, que é um botão clicável que submete as informações do formulário.

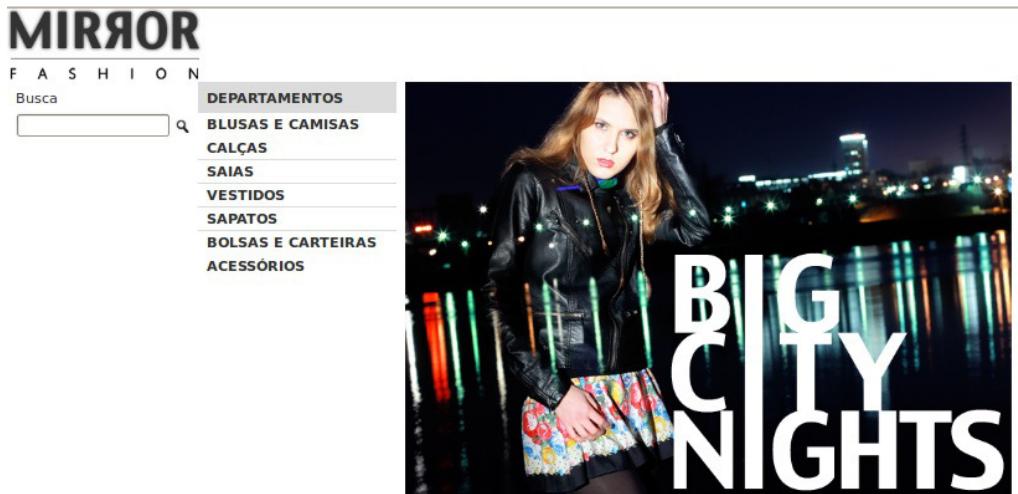
```
<form>
  <input type="search">
  <button>Buscar</button>
</form>
```

4.3 POSICIONAMENTO COM FLOAT E CLEAR

Em nosso layout, precisamos colocar o menu abaixo da busca e alinhado à esquerda com a imagem principal ao lado de ambos. Como conseguir este resultado? Uma solução seria utilizar `position` no menu, mas é algo que quebraria facilmente nosso layout caso a busca aumentasse de tamanho.

Em um dos nossos primeiros exercícios com a página `sobre.html`, colocamos a imagem da família Pelho à direita com a propriedade `float`, fazendo com que o elemento parágrafo a contornasse. Vamos tentar aplicar `float` à busca e ao menu para que ambos fiquem à esquerda, fazendo com que a imagem central os contorne:

```
.busca,  
.menu-departamentos {  
    float: left;  
}
```



O resultado não foi o esperado. Para resolvemos este problema, precisaremos recorrer à propriedade `clear`.

A propriedade clear

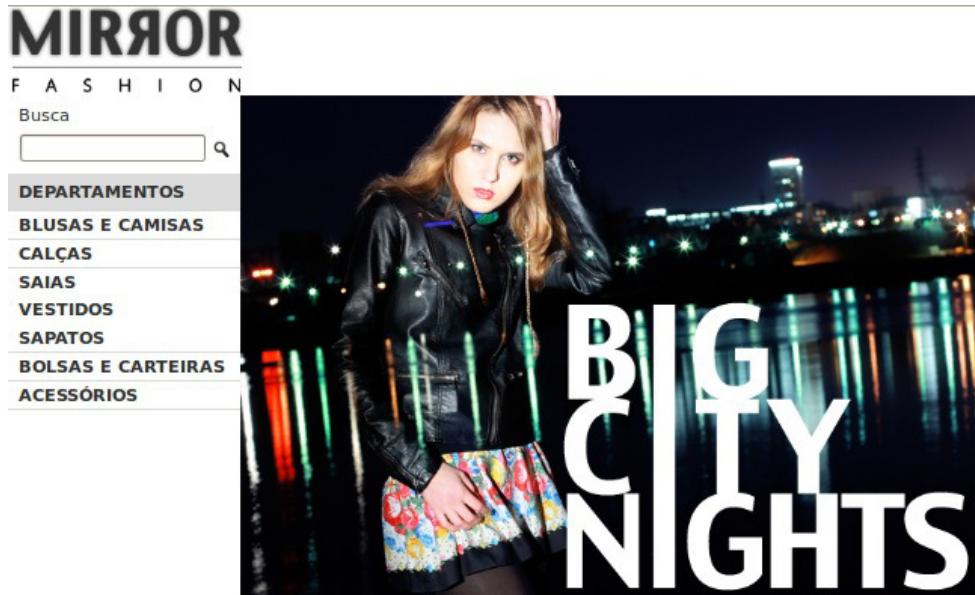
Existe uma propriedade que determina qual vai ser o comportamento de outros elementos que vêm ao redor daquele que a recebe e estão flutuando, e essa propriedade é a `clear`. A propriedade `clear` quer dizer "limpe o fluxo do documento ao meu lado" e pode receber os valores `left`, `right` ou `both`.

O valor `left` impede que elementos flutuantes fiquem à esquerda do elemento que recebe essa propriedade, o valor `right` impede que elementos flutuem à direita desse, e o valor `both` impede que elementos flutuem em ambos os lados do elemento. É importante sabermos que a propriedade `clear` de um elemento só interfere no layout da página caso outros elementos à sua volta estiverem flutuando.

Ao aplicarmos `clear:left` em nosso menu, ele não ficará ao lado da nossa busca com propriedade `float` e será renderizado na linha seguinte:

```
.busca,  
.menu-departamentos {  
    float: left;  
}
```

```
.menu-departamentos {  
    clear: left;  
}
```



4.4 DECORAÇÃO DE TEXTO COM CSS

O CSS permite ainda transformações e decorações de texto.

Transformação de texto

A propriedade `text-transform` permite realizar transformações em textos e seus possíveis valores são:

- **uppercase** - Todas as letras em maiúsculo;
- **lowercase** - Todas as letras em minúsculo;
- **capitalize** - Só as primeiras letras das palavras em maiúsculo.

Se quisermos colocar o texto em caixa alta:

```
.menu-departamentos {  
    text-transform: uppercase;  
}
```

Decoração de texto

Existe uma série de decorações que o navegador adiciona aos textos, dependendo das tags que utilizamos. A decoração mais comum é o sublinhado nos textos de links (tags `<a>` com valor para o atributo "href"). Existem outros tipos de decoração, como por exemplo, o texto contido na tag `` (que serve para indicar um texto que foi removido de uma determinada versão do documento) é exibido

com uma linha bem no meio do texto.

É muito comum que em alguns casos seja desejável ocultar a linha inferior nos links, embora seja recomendado que links dentro de textos sejam decorados para destacarem-se do restante, facilitando a usabilidade e naveabilidade. No caso dos menus, onde temos uma área específica e isolada, normalmente estilizada e decorada o suficiente, normalmente podemos ocultar esse sublinhado, como no exemplo:

```
.item-menu {  
    text-decoration: none;  
}
```

Além do `none` (nenhuma decoração) ainda poderíamos ter configurado `underline` (com uma linha embaixo, o padrão dos links), `overline` (com uma linha em cima do texto) e `line-through` (uma linha no meio do texto).

4.5 CASCATA E HERANÇA NO CSS

Algumas propriedades de elementos pais, quando alteradas, são aplicadas automaticamente para seus elementos filhos em cascata. Por exemplo:

```
<div id="pai">  
    <h1>Sou um título</h1>  
    <h2>Sou um subtítulo</h2>  
</div>  
  
#pai {  
    color: blue;  
}
```

No exemplo acima, todos os elementos filhos **herdaram** o valor da propriedade `color` do elemento pai a qual eles pertencem.

As propriedades que **não** são aplicadas em cascata em elementos filhos geralmente são aquelas que afetam diretamente a caixa (box) do elemento, como `width`, `height`, `margin` e `padding`.

```
h1 {  
    padding-left: 40px;  
}  
  
#pai {  
    color: blue;  
    padding-left: 0;  
}
```

Nesse exemplo o `padding` do elemento `<h1>` não será sobreescrito pelo valor do elemento pai `<div>`, ou seja, o valor `40px` será mantido.

4.6 PARA SABER MAIS: O VALOR INHERIT

Imagine que temos a seguinte divisão com uma imagem:

```
<div>
  
</div>

div {
  border: 2px solid;
  border-color: red;
  width: 30px;
  height: 30px;
}
```

Queremos que a imagem preencha todo o espaço da `<div>`, mas as propriedades `width` e `height` não são aplicadas em cascata, sendo assim, somos obrigados a definir o tamanho da imagem manualmente:

```
img {
  width: 30px;
  height: 30px;
}
```

Esta não é uma solução elegante, porque, se alterarmos o tamanho da `<div>`, teremos que lembrar de alterar também o tamanho da imagem. Uma forma de resolver este problema é utilizando o valor **inherit** para as propriedades `width` e `height` da imagem:

```
img {
  width: inherit;
  height: inherit;
}
```

O valor `inherit` indica para o elemento filho que ele deve utilizar o mesmo valor presente no elemento pai, sendo assim, toda vez que o tamanho do elemento pai for alterado, automaticamente o elemento filho herdará o novo valor, facilitando assim, a manutenção do código.

Lembre-se de que o `inherit` também afeta propriedades que não são aplicadas em cascata.

4.7 EXERCÍCIOS: MENU E DESTAQUE

1. Vamos criar um elemento destaque com a tag `<main>` e, dentro dele, uma `section` para busca, outra para o menu-departamentos e outra para o banner-destaque:

```
<main class="container destaque">

  <section class="busca">
    <h2>Busca</h2>

    <form>
      <input type="search">
      <button>Buscar</button>
    </form>
  </section>
  <!-- fim .busca -->
```

```

<section class="menu-departamentos">
  <h2>Departamentos</h2>

  <nav>
    <ul>
      <li><a href="#">Blusas e Camisas</a></li>
      <li><a href="#">Calças</a></li>
      <li><a href="#">Saias</a></li>
      <li><a href="#">Vestidos</a></li>
      <li><a href="#">Sapatos</a></li>
      <li><a href="#">Bolsas e Carteiras</a></li>
      <li><a href="#">Acessórios</a></li>
    </ul>
  </nav>
</section>
<!-- fim .menu-departamentos -->

<section class="banner-destaque">
  
</section>
<!-- fim .banner-destaque -->

</main>
<!-- fim .container .destaque -->

```

Repare como já usamos diversas classes no HTML para depois selecionarmos os elementos via CSS.

- Aplique o estilo visual do design com CSS. Queremos um fundo cinza nas caixas de busca e no menu de departamentos. Além disso, o texto deve estar em negrito e apresentado em maiúsculas. Aplicaremos também algumas regras de tamanhos e margens.

```

.busca,
.menu-departamentos {
  background-color: #dcdcdc;
  font-weight: bold;
  text-transform: uppercase;
  margin-right: 10px;
  width: 230px;
}

.busca h2,
.busca form,
.menu-departamentos h2 {
  margin: 10px;
}

.menu-departamentos li {
  background-color: white;
  margin-bottom: 1px;
  padding: 5px 10px;
}

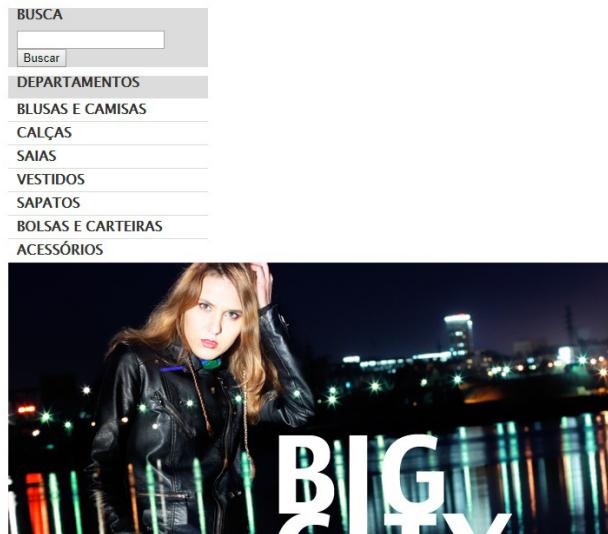
.menu-departamentos a {
  color: #333333;
  text-decoration: none;
}

```

- Na busca, coloque o tamanho do campo de texto para melhor encaixar no design e use seletores de atributo do CSS para isso (veremos mais desses seletores depois no curso).

```
.busca input[type=search] {  
    width: 170px;  
}
```

Teste a página no navegador e veja como o design está quase pronto, apenas o posicionamento dos elementos precisa ser acertado.

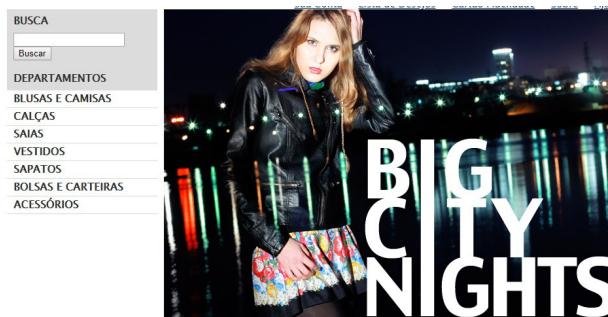


4. Para que o menu de departamentos e a busca estejam à esquerda na página, vamos **flutuar** esses elementos com `float:left`.

Mas só isso fará com que o menu fique a direita da busca (faça o teste). Precisamos indicar ao `menu-departamentos` que ele deve flutuar à esquerda mas não ao lado de outro elemento. Conseguimos isso com a propriedade `clear`.

```
.busca,  
.menu-departamentos {  
    float: left;  
}  
  
.menu-departamentos {  
    clear: left;  
}
```

Observe novamente a página no navegador e veja como o posicionamento está correto.



Repare que o CSS usado foi bastante curto e simples. Mas o conceito do `float` e do `clear` é difícil e complexo. Antes de prosseguir o curso, esteja certo de ter compreendido o porquê do uso dessas propriedades no exercício!

5. Mais acertos de design. Acerte as margens e posicionamentos no menu lateral e no topo:

```
.destaque {  
    margin-top: 10px;  
}  
  
.menu-departamentos {  
    margin-top: 10px;  
    padding-bottom: 10px;  
}
```

Teste o resultado no navegador.

4.8 DISPLAY INLINE-BLOCK

Precisamos criar um painel com uma lista de novidades, onde cada produto será representado por uma ``. Já sabemos que por padrão uma `` possui a propriedade `display:block`, mas queremos os produtos lado a lado. Podemos trocar este comportamento mudando a propriedade `display` dos elementos para `inline`.

Também será necessário alterar as propriedades `width`, `margin` e `padding` das ``, mas agora os elementos `` são `inline` e este modo de exibição ignora alterações que afetam as propriedades da `box`. Como resolver este problema?

Os navegadores mais modernos introduzem um modelo de exibição que é a mistura dos dois, o `inline-block`. Os elementos que recebem o valor `inline-block` para a propriedade `display` obedecem às especificações de dimensão das propriedades `height` (altura) e `width` (largura) e ainda permitem que outros elementos sejam exibidos ao seu lado como elementos `inline`.

```
.painel li {  
    display: inline-block;  
    vertical-align: top;  
    width: 140px;  
    margin: 2px;  
    padding-bottom: 10px;  
}
```

Como o `inline-block` faz os elementos se alinharem como numa linha de texto, podemos controlar o alinhamento vertical dessa linha da mesma forma que fizemos antes com linhas de texto e imagens simples.

Para isso, usamos a propriedade `vertical-align` que, nesse caso, recebeu valor `top` e isso indica que se tivermos vários produtos de tamanhos diferentes, eles vão se alinhar pelo topo. Há outros valores possíveis para essa propriedade, como por exemplo: `bottom`, `middle`, `text-bottom` e `text-top`.

4.9 EXERCÍCIOS: PAINÉIS FLUTUANTES

- Vamos criar agora nossos painéis de novidades e mais vendidos. Crie um elemento `<article>` para conter os **dois painéis de produtos**. Ele deve receber a classe **container**, para se alinhar ao meio da tela, e a classe **paineis** que usaremos depois no CSS.

```
<article class="container painéis">
    <!-- os painéis de novidades e mais vendidos entrarão aqui dentro -->
</article>
<!-- fim .container .paineis -->
```

- Dentro do **article criado acima**, criaremos uma nova `<section>` para cada painel. A primeira, receberá as classes **painel** e **novidades**. Ela conterá o título em um `<h2>` e uma `<nav>` com uma lista ordenada (``) de produtos.

Cada produto deve ser representado como um item na lista (``) com um link para o produto e sua imagem (representado por `figure`, `figcaption` e `img`).

Veja o exemplo com um produto. **Ele deve ser incluído dentro da section que você acabou de criar:**

```
<section class="painel novidades">
<h2>Novidades</h2>
<nav>
    <ol>
        <!-- primeiro produto -->
        <li>
            <a href="produto.html">
                <figure>
                    
                    <figcaption>Fuzz Cardigan por R$ 129,90</figcaption>
                </figure>
            </a>
        </li>
        <!-- fim do primeiro produto -->
        <!-- coloque mais produtos aqui! -->
    </ol>
</nav>
</section>
```

Crie o HTML desse painel e o preencha com vários produtos (6 é um bom número). Lembre-se de que cada produto está na sua própria `li` com link e imagem próprios. Na pasta **img/produtos** do seu projeto, você encontra várias imagens **miniaturaX.png** que podem ser usadas para criar produtos diferentes.

- Crie um segundo painel, para representar os produtos mais vendidos. Esse painel deve ficar **após** o fechamento do painel anterior, mas ainda **dentro** da tag `<article class="container painéis">`.

O novo painel deve receber as classes **painel** e **mais-vendidos**. Sua estrutura é idêntica ao do

exercício anterior (dica: copie o código para evitar refazer tudo de novo).

```
<section class="painel mais-vendidos">
  <h2>Mais Vendidos</h2>
  <nav>
    <ol>
      <!-- coloque vários produtos aqui -->
    </ol>
  </nav>
</section>
```

Nosso *HTML* já está ficando grande e complexo, como é uma página real cheia de conteúdo. Cuidado para não se confundir na posição das tags. Recapitulando essa parte dos painéis, a estrutura deve estar assim:

- **article**: container painéis
- **section**: painel novidades
- **h2**: título Novidades
- **ol**: lista de produtos
- vários **li**s com produtos (links e imagens dentro de cada um)
- **section**: painel mais-vendidos
- **h2**: título Mais Vendidos
- **ol**: lista de produtos
- vários **li**s com produtos (links e imagens dentro de cada um)

4. Vamos posicionar nossos painéis para ficarem de acordo com o design.

O painel de novidades deve flutuar à esquerda e o `mais-vendidos`, à direita. Cada um deve ocupar 445px (pouco menos da metade dos 940px), assim um ficará ao lado do outro:

```
.painel {
  margin: 10px 0;
  padding: 10px;
  width: 445px;
}

.novidades {
  float: left;
}

.mais-vendidos {
  float: right;
}
```

Próximo passo: os itens dos produtos dentro da lista de cada painel. Queremos que sejam dispostos lado a lado mas com certo tamanho e espaçamento para alinhamento. Conseguimos isso colocando `display:inline-block` nos elementos da lista e, para alinhar os produtos pelo topo, com `vertical-align:top`.

```
.painel li {
    display: inline-block;
    vertical-align: top;
    width: 140px;
    margin: 2px;
    padding-bottom: 10px;
}
```

O posicionamento em si deve estar certo. Mas falta umas regras para estilo, como tamanho dos títulos e cores de texto e fundo.

```
.painel h2 {
    font-size: 24px;
    font-weight: bold;
    text-transform: uppercase;
    margin-bottom: 10px;
}

.painel a {
    color: #333;
    font-size: 14px;
    text-align: center;
    text-decoration: none;
}

.novidades {
    background-color: #f5dcfc;
}

.mais-vendidos {
    background-color: #dcdcf5;
}
```

Teste a página no navegador e veja o resultado final!



4.10 SELETORES DE ATRIBUTO DO CSS3

Além dos seletores de tag, classe e id que aprendemos anteriormente, existe mais uma série de seletores avançados do CSS.

Um dos seletores CSS mais versáteis é o seletor de atributo, com ele podemos verificar a presença ou valor de um atributo para selecioná-lo. Por exemplo:

```
input[value] {  
    color: #cc0000;  
}
```

O seletor acima age em todos os elementos da tag `<input>` que têm o atributo "value". Também é possível verificar se o atributo tem um valor específico:

```
input[type="text"] {  
    border-radius: 4px;  
}
```

Além de verificar um valor integralmente, é possível utilizar alguns operadores para selecionar valores em determinadas condições, como por exemplo o seletor de atributo com prefixo:

```
div[class|=“menu”] {  
    border-radius: 4px;  
}
```

O seletor acima vai agir em todas as tags `<div>` cujo atributo "class" comece com a palavra **menu** seguida de um hífen e qualquer outro valor na sequência, como por exemplo **menu-principal**, **menu-departamentos** e **menu-teste**.

Também é possível buscar por uma palavra específica no valor, não importando o valor completo do atributo. Por exemplo:

```
input[value~="problema"] {  
    color: #cc0000;  
}
```

O seletor acima agirá sobre todos os elementos da tag `<input>` que contiverem a palavra "problema" em seu conteúdo.

Com o CSS3 é possível utilizar novos operadores com sinais que se assemelham aos das expressões regulares:

```
/* busca por inputs com valor de "name" iniciando em "usuario" */  
input[name^="usuario"] {  
    color: #99ffcc;  
}  
  
/* busca por inputs com valor de "name" terminando em "teste" */  
input[name$="teste"] {  
    background-color: #ccff00;  
}  
  
/* busca por inputs com valor do atributo "name"  
    contendo "tela" em qualquer posição */  
input[name*="tela"] {  
    color: #666666;  
}
```

Os seletores de atributo têm o mesmo valor de especificidade dos seletores de classe.

4.11 RODAPÉ

Para finalizarmos a página, precisamos desenvolver o rodapé. Visualmente, ele é bastante simples. Mas há algumas questões importantes a serem salientadas.

Semântica

No HTML5, a tag apropriada para rodapés é a `<footer>`, que vamos usar no exercício. Além disso, nosso rodapé ainda tem mais 2 conteúdos: o logo em negativo do lado esquerdo e ícones de acesso a redes sociais do lado direito. Quais elementos podemos usar?

O logo no lado esquerdo é uma simples imagem:

```

```

Já a lista de ícones das redes sociais, na verdade, é uma lista de links. Os ícones são meramente ilustrativos. Em um leitor de tela, vamos querer que um link seja lido para o usuário, independentemente do seu ícone gráfico.

Podemos usar então uma simples lista com `<a>`:

```
<ul class="social">
  <li><a href="http://facebook.com/mirrorfashion">Facebook</a></li>
  <li><a href="http://twitter.com/mirrorfashion">Twitter</a></li>
  <li><a href="http://plus.google.com/mirrorfashion">Google+</a></li>
</ul>
```

Esse é um ponto importante para entendermos a diferença entre marcação semântica e apresentação visual. Repare que criamos uma estrutura no HTML com conteúdo completamente diferente do resultado final visual. Vamos cuidar do visual depois no CSS.

4.12 SUBSTITUIÇÃO POR IMAGEM

Um truque muito usado em CSS é o chamado **Image Replacement** -- ou *substituição por imagem*. Serve para, usando técnicas de CSS, exibir uma imagem em algum elemento que originalmente foi feito com texto. Perfeito para nosso caso dos ícones das redes sociais.

A ideia básica é:

- Acertar o tamanho do elemento para ficar igual ao da imagem;
- Colocar a imagem como `background` do elemento;
- Esconder o texto.

Para esconder o texto, é comum usar a tática de colocar um `text-indent` negativo bastante alto.

Isso, na prática, faz o texto ser renderizado "fora da tela".

```
.facebook {  
    /* tamanho do elemento = imagem */  
    height: 55px;  
    width: 85px;  
  
    /* imagem como fundo */  
    background-image: url(..../img/facebook.png);  
  
    /* retirando o texto da frente */  
    text-indent: -9999px;  
}
```

4.13 ESTILIZAÇÃO E POSICIONAMENTO DO RODAPÉ

Container interno

Repare que o rodapé, diferentemente de todos os elementos do layout, ocupa 100% da largura da página. Ele não é restrito ao tamanho de 940px do miolo do nosso site. Isso porque o rodapé tem um `background` que se repete até os cantos.

Mas repare que o conteúdo dele é limitado aos 940px e centralizado junto com o resto da página -- onde estávamos usando a `class container`.

O que precisamos fazer então é ter o `<footer>` com 100% e uma tag interna para o conteúdo do rodapé em si, com a classe `container`:

```
<footer>  
    <div class="container">  
        ...  
    </div>  
</footer>
```

Posicionamento

Ao colocar o rodapé, você perceberá que ele subirá na página ao invés de ficar embaixo. Isso porque os elementos anteriores a ele, os painéis de destaque, estão flutuando na página e, portanto, saíram do fluxo de renderização. Para corrigir isso, basta usar a propriedade `clear: both` no `footer`.

Dentro do rodapé em si, queremos que a lista de ícones seja colocada à direita. Podemos acertar isso com **posicionamento absoluto**, desde que o container do rodapé esteja *posicionado* (basta dar um `position: relative` a ele).

Já os itens dentro da lista (os 3 links), devem ser flutuados lado a lado (e não um em cima do outro). É fácil fazer com `float: left` no `li`.

Estilização

O rodapé em si terá um `background-image` com o fundo preto estampado repetido infinitamente.

Os elementos internos são todos itens a serem substituídos por imagens via CSS com *image replacement*.

E, para saber qual ícone atribuir a qual link da lista de mídias sociais, podemos usar seletores de atributo do CSS3:

```
.social a[href*="facebook.com"] {  
    background-image: url(..../img/facebook.png);  
}
```

4.14 EXERCÍCIOS: RODAPÉ

1. Vamos finalizar nossa página com o rodapé do layout. Crie uma estrutura semântica no HTML usando a tag `<footer>` e tags ``, ``, `` e `<a>` para o conteúdo.

Atenção especial para a necessidade de um elemento `container` **dentro** do rodapé, para alinhar seu conteúdo com o restante da página.

```
<footer>  
    <div class="container">  
          
  
        <ul class="social">  
            <li><a href="http://facebook.com/mirrorfashion">Facebook</a></li>  
            <li><a href="http://twitter.com/mirrorfashion">Twitter</a></li>  
            <li><a href="http://plus.google.com/mirrorfashion">Google+</a></li>  
        </ul>  
  
    </div>  
</footer>
```

Teste no seu navegador e veja o resultado sem estilo, mas utilizável.

2. Vamos estilizar o conteúdo visual. Coloque o `background` preto no rodapé e faça as substituições das imagens. Use *seletores de atributo* do CSS3 para identificar os ícones de cada rede social.

```
footer {  
    background-image: url(..../img/fundo-rodape.png);  
}  
  
.social a {  
    /* tamanho da imagem */  
    height: 32px;  
    width: 32px;  
  
    /* image replacement */  
    display: block;  
    text-indent: -9999px;  
}  
  
.social a[href*="facebook.com"] {  
    background-image: url(..../img/facebook.png);  
}
```

```
.social a[href*="twitter.com"] {  
    background-image: url(../img/twitter.png);  
}  
  
.social a[href*="plus.google.com"] {  
    background-image: url(../img/googleplus.png);  
}
```

Teste no navegador. O que aconteceu?

O rodapé subiu na página porque os elementos anteriores (os painéis de destaque) estão flutuando. É fácil arrumar, basta adicionar a regra de `clear` no `footer`:

```
footer {  
    clear: both;  
}
```

Teste novamente no navegador. O rodapé voltou para o lugar certo?

3. Último passo para finalizar o rodapé: posicionar os elementos internos do rodapé apropriadamente.

Vamos posicionar os ícones sociais absolutamente à direita com `position: absolute`. Para isso, o container do nosso rodapé precisa estar posicionado. Aproveite também e coloque um espaçamento interno:

```
footer {  
    padding: 20px 0;  
}  
  
footer .container {  
    position: relative;  
}  
  
.social {  
    position: absolute;  
    top: 12px;  
    right: 0;  
}
```

Por fim, precisamos fazer os ícones das redes sociais fluturarem lado a lado horizontalmente:

```
.social li {  
    float: left;  
    margin-left: 25px;  
}
```

Teste no navegador. Como está o resultado final? De acordo com o que o designer queria?



4. Aproveitando que já aprendemos como adicionar imagens, podemos trocar o escrito do botão da

busca pela imagem da lupa, seguindo o nosso layout:

```
.busca button {  
    /* adicionando imagem */  
    background-image: url(..../img/busca.png);  
    background-repeat: no-repeat;  
    border: none;  
  
    /* tamanho da imagem */  
    width: 20px;  
    height: 20px;  
  
    /* imagem replacement */  
    text-indent: -9999px;  
}
```

4.15 EXERCÍCIOS OPCIONAIS

1. Coloque nosso rodapé para a página **sobre.html** do capítulo anterior.
2. Nossa página **sobre.html** foi construída sem muita preocupação semântica. No HTML5, há novas tags com objetivo específico de lidar com conteúdos textuais divididos em partes, com subtítulos etc.

Podem ser artigos de um jornal, um livro online ou mesmo um texto descrevendo nossa empresa - como nossa página **sobre.html** faz.

Podemos representar o texto todo com `<article>` e suas seções com `<section>`. Use essas novas tags na **sobre.html** para termos uma marcação mais semântica.

3. (desafio) O posicionamento dos elementos no rodapé possui um deselegante `top:12px` . Um número mágico arbitrário para centralizar verticalmente os ícones sociais e o logotipo. Repense o posicionamento para chegar em um código mais elegante, que evite o uso de *magic numbers*.

CSS AVANÇADO

"Com o conhecimento nossas dúvidas aumentam" -- Johann Goethe

Desde o surgimento do CSS, os desenvolvedores front-end utilizam diversas técnicas para alterar a exibição dos elementos no navegador. Mesmo assim algumas coisas eram impossíveis de se conseguir utilizando somente CSS. Conhecendo o comportamento dos navegadores ao exibir um elemento (ou um conjunto de elementos) e como as propriedades do CSS agem ao modificar um elemento é possível obter resultados impressionantes.

O CSS3 agora permite coisas antes impossíveis como elementos com cor ou fundo gradiente, sombras e cantos arredondados. Antes só era possível atingir esses resultados com o uso de imagens e às vezes até com um pouco de JavaScript.

A redução do uso de imagens traz grandes vantagens quanto à performance e quantidade de tráfego de dados necessária para a exibição de uma página.

5.1 SELETORES AVANÇADOS

Os seletores CSS mais comuns e tradicionais são os que já vimos: por ID, por classes e por descendência.

No entanto, há muitos outros seletores novos que vão entrando na especificação e que são bastante úteis. Já vimos alguns, como os seletores de atributo que usamos anteriormente. Vejamos outros.

Seletor de irmãos

Veja o seguinte HTML, que simula um texto com vários parágrafos, títulos e subtítulos no meio do documento:

```
<article>
  <h1>Título</h1>
  <p>Início</p>

  <h2>Subtítulo</h2>
  <p>Texto</p>
  <p>Mais texto</p>
</article>
```

Como faremos se quisermos estilizar de uma certa maneira todos os parágrafos após o subtítulo?

O seletor de **irmãos** (*siblings*) (~) serve pra isso! Ele vem do CSS3 e funciona em todos os navegadores modernos (e no IE7 pra frente).

```
h2 ~ p {  
    font-style: italic;  
}
```

Isso indica que queremos selecionar *todos* os `p` que foram precedidos por algum `h2` e são irmãos do subtítulo (ou seja, estão sob a mesma tag pai). No HTML anterior, serão selecionados os dois últimos parágrafos (*Texto* e *Mais texto*).

Seletor de irmão adjacente

Ainda com o HTML anterior, o que fazer se quisermos selecionar apenas o parágrafo imediatamente seguinte ao subtítulo? Ou seja, é um `p` irmão do `h2` mas que aparece logo na sequência.

Fazemos isso com o seletor de irmão adjacente - *adjacent sibling*:

```
h2 + p {  
    font-variant: small-caps;  
}
```

Nesse caso, apenas o parágrafo *Texto* será selecionado. É um irmão de `h2` e aparece logo depois do mesmo.

Esse seletor faz parte da especificação CSS2.1 e tem bom suporte nos navegadores modernos, incluindo o IE7.

Seletor de filho direto

Se tivermos o seguinte HTML com títulos e seções de um artigo:

```
<article>  
    <h1>Título principal</h1>  
  
    <section>  
        <h1>Título da seção</h1>  
    </section>  
</article>
```

Queremos deixar o título principal de outra cor. Como fazer? O seletor de nome de tag simples não vai resolver:

```
/* vai pegar todos os h1 da página */  
h1 {  
    color: blue;  
}
```

Tentar o seletor de hierarquia também não vai ajudar:

```
/* vai pegar todos os h1 do article, incluindo o de dentro da section */  
article h1 {  
    color: blue;
```

```
}
```

Entra aí o **seletor de filho direto** (`>`) do CSS2.1 e suportado desde o IE7 também.

```
/* vai pegar só o h1 principal, filho direto de article e não os netos */
article > h1 {
  color: blue;
}
```

Negação

Imagine o seguinte HTML com vários parágrafos simples:

```
<p>Texto</p>
<p>Outro texto</p>
<p>Texto especial</p>
<p>Mais texto</p>
```

Queremos fazer todos os parágrafos de cor cinza, exceto o que tem o texto especial. Precisamos destacá-lo de alguma forma no HTML para depois selecioná-lo no CSS. Uma classe ou ID resolve:

```
<p>Texto</p>
<p>Outro texto</p>
<p class="especial">Texto especial</p>
<p>Mais texto</p>
```

Mas como escrever o CSS? Queremos mudar a cor dos parágrafos que *não têm a classe especial*. Um jeito seria mudar a cor de todos e sobrescrever o especial depois:

```
p {
  color: gray;
}

p.especial {
  color: black; /* restaura cor do especial */
}
```

No CSS3, há uma outra forma, usando o **seletor de negação**. Ele nos permite escrever um seletor que pega elementos que não batem naquela regra.

```
p:not(.especial) {
  color: gray;
}
```

Isso diz que queremos todos os parágrafos que não têm a classe especial. A sintaxe do `:not()` recebe como argumento algum outro seletor simples (como classes, IDs ou tags).

Essa propriedade do CSS3 possui suporte mais limitado no IE, somente a partir da versão 9 (nos outros navegadores não há problemas).

5.2 PSEUDO-CLASSES

Pegue o seguinte HTML de uma lista de elementos:

```
<ul>
  <li>Primeiro item</li>
  <li>Segundo item</li>
  <li>Terceiro item</li>
  <li>Quarto item</li>
</ul>
```

E se quisermos estilizar elementos específicos dessa lista? Por exemplo, o primeiro elemento deve ter cor vermelha e o último, azul. Com esse HTML simples, usando apenas os seletores que vimos até agora, será bem difícil.

A solução mais comum seria adicionar classes ou IDs no HTML para selecioná-los depois no CSS:

```
<ul>
  <li class="primeiro">Primeiro item</li>
  <li>Segundo item</li>
  <li>Terceiro item</li>
  <li class="ultimo">Quarto item</li>
</ul>
```

Agora é fácil usar cores diferentes para o primeiro e último itens da lista.

Mas essa técnica exigiu alteração no HTML e exige que lembremos de colocar a classe correta, no ponto correto, toda vez que fizermos mudanças nos itens da lista.

O CSS tem um recurso chamado de **pseudo-classes** que são como classes CSS já pré-definidas para nós. É como se o navegador já colocasse certas classes por padrão em certos elementos, cobrindo situações comuns como essa de selecionar o primeiro ou o último elemento.

Há duas pseudo-classes do CSS3 que representam exatamente o primeiro elemento filho de outro (**first-child**) e o último elemento filho (**last-child**). Essas classes já estão definidas, não precisamos aplicá-las em nosso HTML e podemos voltar para o HTML inicial:

```
<ul>
  <li>Primeiro item</li>
  <li>Segundo item</li>
  <li>Terceiro item</li>
  <li>Quarto item</li>
</ul>
```

No CSS, podemos usar pseudo-classes quase da mesma forma que usariamoss classes normais. Repare que para diferenciar um tipo do outro, mudou-se o operador de ponto para dois pontos:

```
li:first-child {
  color: red;
}

li:last-child {
  color: blue;
}
```

O suporte a esses seletores é completo nos navegadores modernos. O `first-child` vem desde o IE7, Firefox 3 e Chrome 4. E (estranhamente), o `last-child` só a partir do IE9 mas desde o Firefox 1

e Chrome 1.

NTH-CHILD

Um seletor ainda mais genérico do CSS3 é o `:nth-child()` que nos permite passar o índice do elemento. Por exemplo, podemos pegar o terceiro item com:

```
li:nth-child(3) { color: yellow; }
```

Porém, o mais interessante é que o `nth-child` pode receber uma expressão aritmética para indicar quais índices selecionar. É fácil fazer uma lista zebraada, com itens ímpares de uma cor e pares de outra:

```
/* elementos pares */
li:nth-child(2n) { color: green; }

/* elementos ímpares */
li:nth-child(2n+1) { color: blue; }
```

O suporte existe a partir do IE9, Firefox 3.5 e Chrome 1.

Pseudo classes de estado

Queremos mudar a cor de um link quando o usuário passa o mouse por cima. Ou seja, queremos mudar seu visual a partir de um evento do usuário (no caso, passar o mouse em cima).

Uma solução ingênua seria criar um código JavaScript que adiciona uma classe nos links quando o evento de mouseover acontece (e remove a classe no mouseout).

Entretanto, o CSS possui excelentes **pseudo-classes que representam estados dos elementos** e, em especial, uma que representa o momento que o usuário está com o mouse em cima do elemento, a `:hover`.

É como se o navegador aplicasse uma classe chamada **hover** automaticamente quando o usuário passa o mouse em cima do elemento e depois retirasse a classe quando ele sai. Tudo sem precisarmos controlar isso com JavaScript.

```
/* seleciona o link no exato momento em que passamos o mouse por cima dele */
a:hover {
    background-color:#ff00ff;
}
```

Podemos usar hover em todo tipo de elemento, não apenas links. Mas os links ainda têm outras pseudo-classes que podem ser úteis:

```
/* seleciona todas as âncoras que têm o atributo "href", ou seja, links */
a:link {
    background-color:#ff0000;
```

```

}

/* seleciona todos os links cujo valor de "href" é um endereço já visitado */
a:visited {
    background-color:#00ff00;
}

/* seleciona o link no exato momento em que clicamos nele */
a:active {
    background-color:#0000ff;
}

```

5.3 PSEUDO ELEMENTOS

Pseudo-classes nos ajudam a selecionar elementos com certas características sem termos que colocar uma classe manualmente neles. Porém, o que fazer quando precisamos selecionar certo tipo de conteúdo que nem elemento tem?

Exemplo: temos um texto num parágrafo:

```
<p>A Caelum tem os melhores cursos!</p>
```

Queremos dar um estilo de revista ao nosso texto e estilizar apenas a *primeira letra* da frase com uma fonte maior. Como fazer para selecionar essa letra? A solução ingênuo seria colocar um elemento ao redor da letra para podermos selecioná-la no CSS:

```
<p><span>A</span> Caelum tem os melhores cursos!</p>
```

HTML confuso e difícil de manter.

O CSS apresenta então a ideia de **pseudo-elementos**. São elementos que não existem no documento mas podem ser selecionados pelo CSS. É como se houvesse um elemento lá!

Podemos voltar o HTML inicial:

```
<p>A Caelum tem os melhores cursos!</p>
```

E no CSS:

```
p::first-letter {
    font-size: 200%;
}
```

Temos ainda outro pseudo-elemento para selecionar a primeira linha apenas em um texto grande:

```
p::first-line {
    font-style: italic;
}
```

Novos conteúdos

Há ainda um outro tipo de pseudo-elemento mais poderoso que nos permite gerar conteúdo novo via CSS. Imagine uma lista de links que queremos, visualmente, colocar entre colchetes:

```
[ Link 1 ]  
[ Link 2 ]  
[ Link 3 ]
```

Podemos, claro, apenas escrever os tais colchetes no HTML. Mas será que o conteúdo é semântico? Queremos que esses colchetes sejam indexados pelo Google? Queremos que sejam lidos como parte do texto pelos leitores de tela?

Talvez não. Pode ser um conteúdo apenas visual. Podemos gerá-lo com CSS usando os pseudo-elementos **after** e **before**.

O HTML seria simples:

```
<a href="#">Link1</a>  
<a href="#">Link2</a>  
<a href="#">Link3</a>
```

E no CSS:

```
a::before {  
    content: '[ ';  
}  
  
a::after {  
    content: ' ]';  
}
```

Ou ainda, imagine que queremos colocar a mensagem **(externo)** ao lado de todos os links externos da nossa página. Usando pseudo-elementos e seletores de atributo, conseguimos:

```
a[href^="http://"]::after {  
    content: ' (externo)';  
}
```

Isso pega todos os elementos `<a>` que começam com `http://` e coloca a palavra **(externo)** depois.

5.4 EXERCÍCIOS: SELETORES, PSEUDO-CLASSES E PSEUDO-ELEMENTOS

1. Vamos alterar nossa página de *Sobre*, a **sobre.html**. Queremos que as primeiras letras dos parágrafos fiquem em negrito.

Adicione no arquivo **sobre.css** o seletor com pseudo-elemento `::first-letter` para isso.

```
p::first-letter {  
    font-weight: bold;  
}
```

Teste a página no navegador!

2. Repare que os parágrafos nessa página *Sobre* têm uma indentação no início. Agora queremos **remover apenas a indentação do primeiro parágrafo** da página.

Poderíamos colocar uma classe no HTML. Ou, melhor ainda, sabendo que esse é o primeiro parágrafo (`<p>`) depois do título (`<h1>`), usar o **seletor de irmão adjacente**.

Acrescente ao **sobre.css**:

```
h1 + p {  
    text-indent: 0;  
}
```

Teste novamente no navegador.

3. Podemos ainda usar o pseudo-elemento `::first-line` para alterar o visual da primeira linha de todos os parágrafos. Por exemplo, transformando-a em **small-caps** usando a propriedade `font-variant` :

```
p::first-line {  
    font-variant: small-caps;  
}
```

Teste de novo no navegador.

A MIRROR FASHION É A MAIOR EMPRESA COMÉRCIO ELETRÔNICO NO SEGMENTO DE MODA EM TODO O MUNDO. FUNDADA EM 1932, POSSUI FILIAIS EM 124 PAÍSES, SENDO LÍDER DE MERCADO COM MAIS DE 90% DE PARTICIPAÇÃO EM 118 DELES.

NOSO CENTRO DE DISTRIBUIÇÃO FICA EM [JACAREZINHO, NO PARANÁ](#). DE LÁ, SAEM 48 AVIÕES QUE DISTRIBUEM NOSSOS PRODUTOS ÀS CASAS DO MUNDO TODO. NOSO CENTRO DE DISTRIBUIÇÃO:

4. Vamos voltar a mexer na página **index.html** do nosso site.

Temos o menu superior (**.menu-opcoes**) que é uma lista de links. Podemos melhorar sua usabilidade alterando seus estados quando o usuário passar o mouse (**:hover**) e quando clicar no item (**:active**).

Adicione ao arquivo **estilos.css**:

```
.menu-opcoes a:hover {  
    color: #007dc6;  
}  
  
.menu-opcoes a:active {  
    color: #867dc6;  
}
```

Teste o menu passando o mouse e clicando nas opções (segure um pouco o clique para ver melhor o efeito).

5. O **hover** é útil em vários cenários. Um interessante é fazer um menu que abre e fecha em puro CSS.

Temos já o nosso **.menu-departamentos** na esquerda da página com várias categorias de produtos. Queremos adicionar *subcategorias* que aparecem apenas quando o usuário passar o mouse.



Hoje, o menu é um simples `` com vários itens (``) com links dentro:

```
<li><a href="#">Blusas e Camisas</a></li>
```

Vamos adicionar no **index.html** uma **sublista** de opções **dentro** do `` de *Blusas e Camisas*, dessa forma:

```
<li>
  <a href="#">Blusas e Camisas</a>
  <ul>
    <li><a href="#">Manga curta</a></li>
    <li><a href="#">Manga comprida</a></li>
    <li><a href="#">Camisa social</a></li>
    <li><a href="#">Camisa casual</a></li>
  </ul>
</li>
```

Por padrão, queremos que essa sublista esteja escondida (`display: none`). Quando o usuário passar o mouse (`:hover`), queremos exibi-la (`display: block`).

Altere o arquivo **estilos.css** com essa lógica.

```
.menu-departamentos li ul {
  display: none;
}

.menu-departamentos li:hover ul {
  display: block;
}

.menu-departamentos ul ul li {
  background-color: #dcdcdc;
}
```

Teste a página e a funcionalidade do menu.

6. Para ajudar a diferenciar os links dos submenus, queremos colocar um **traço na frente**. Podemos alterar o HTML colocando os traços - algo visual e não semântico -, ou podemos **gerar esse traço** via CSS com pseudo-elementos.

Use o `::before` para injetar um conteúdo novo via propriedade `content` no CSS:

```
.menu-departamentos li li a::before {  
    content: '- ';  
}
```

Teste a página.

Veja os exercícios opcionais a seguir com outras possibilidades.

5.5 EXERCÍCIOS OPCIONAIS

1. A propriedade `content` tem muitas variações. Uma variação simples, mas útil, é usar caracteres *unicode* para injetar símbolos mais interessantes.

Altere o seletor que fizemos no exercício anterior:

```
.menu-departamentos li li a::before {  
    content: '\272A';  
    padding-right: 5px;  
}
```

Para os títulos dos painéis podemos adicionar um *unicode* diferente:

```
.painel h2::before {  
    content: '\2756';  
    padding-right: 5px;  
    opacity: 0.4;  
}
```

Repare que usamos também a propriedade `opacity` para deixar esse elemento mais transparente e sutil.

MAIS OPÇÕES DO UNICODE

Consulte em uma tabela Unicode outros caracteres e seu código **hex** correspondente.

<http://www.alanwood.net/unicode/dingbats.html>

<http://www.alanwood.net/unicode/#links>

2. (avançado) Volte à página **sobre.html**, abra-a no navegador.

Em um exercício anterior, colocamos as primeiras linhas em *small-caps* usando o seletor **p::first-line**. Repare que todos os parágrafos foram afetados.

E se quiséssemos que apenas *parágrafos de início de seção* fossem afetados? Podemos pensar assim: queremos alterar todos os parágrafos que não estão no meio do texto, ou seja, não são precedidos por outro parágrafo (mas sim precedidos por títulos, figuras etc).

Com o seletor `:not()` do CSS3, conseguimos (**alterar o que fizemos no exercício anterior**):

```
:not(p) + p::first-line {  
    font-variant: small-caps;  
}
```

Isso significa: **selecione as primeiras linhas dos parágrafos que não são precedidos por outros parágrafos.**

5.6 CSS3: BORDER-RADIUS

Uma das novidades mais celebradas do CSS3 foi a adição de bordas arredondadas via CSS. Até então, a única forma de obter esse efeito era usando imagens, o que deixava a página mais carregada e dificultava a manutenção.

Com o CSS3 há o suporte a propriedade `border-radius` que recebe o tamanho do raio de arredondamento das bordas. Por exemplo:

```
div {  
    border-radius: 5px;  
}
```



Podemos também passar valores diferentes para cantos diferentes do elemento:

```
/* todas as bordas arredondadas com um raio de 15px */  
.a {  
    border-radius: 15px;  
}  
  
/* borda superior esquerda e inferior direita com 5px  
borda superior direita e inferior esquerda com 20px */  
.b {  
    border-radius: 5px 20px;  
}  
  
/* borda superior esquerda com 5px  
borda superior direita e inferior esquerda com 20px  
borda inferior direita com 50px */  
.c {  
    border-radius: 5px 20px 50px;  
}  
  
/* borda superior esquerda com 5px  
borda superior direita com 20px
```

```
borda inferior direita com 50px  
borda inferior esquerda com 100px */  
.d {  
    border-radius: 5px 20px 50px 100px;  
}
```



5.7 CSS3: TEXT-SHADOW

Outro efeito do CSS3 é o suporte a sombras em textos com `text-shadow`. Sua sintaxe recebe o deslocamento da sombra e sua cor:

```
p {  
    text-shadow: 10px 10px red;  
}
```



Ou ainda pode receber um grau de espalhamento (blur):

```
p {  
    text-shadow: 10px 10px 5px red;  
}
```



É possível até passar mais de uma sombra ao mesmo tempo para o mesmo elemento:

```
p {  
    text-shadow: 10px 10px 5px red, -5px -5px 4px red;  
}
```



5.8 CSS3: BOX-SHADOW

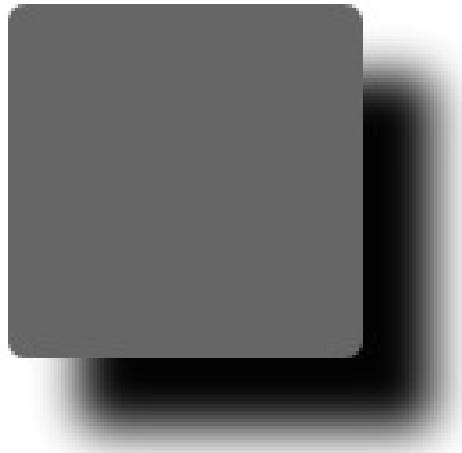
Além de sombras em texto, podemos colocar sombras em qualquer elemento com `box-shadow`. A sintaxe é parecida com a do `text-shadow`:

```
box-shadow: 20px 20px black;
```



Podemos ainda passar um terceiro valor com o blur:

```
box-shadow: 20px 20px 20px black;
```



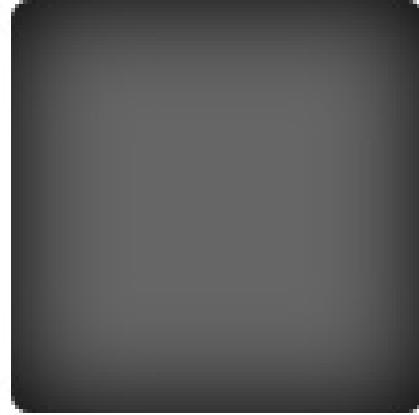
Diferentemente do `text-shadow`, o `box-shadow` suporta ainda mais um valor que faz a sombra aumentar ou diminuir:

```
box-shadow: 20px 20px 20px 30px black;
```



Por fim, é possível usar a keyword **inset** para uma borda interna ao elemento:

```
box-shadow: inset 0 0 40px black;
```



5.9 OPACIDADE E RGBA

Desde o CSS2 é possível mudar a opacidade de um elemento para que ele seja mais transparente com o uso da propriedade `opacity`.

Veja esse exemplo com um parágrafo por cima de uma imagem:



Colocamos um fundo preto e cor branca no texto. E se quisermos um efeito legal de transparência para deixar a imagem mostrar mais?



#

É simples com a `opacity` que recebe um valor decimal entre 0 e 1:

```
p {  
    opacity: 0.3;  
}
```

Repare como todo o elemento fica transparente, o fundo e o texto. E se quiséssemos o texto em branco opaco e o fundo com transparéncia? No CSS3, podemos usar outra técnica, a de definir uma cor de fundo com valor de opacidade específica.

No CSS3, além das cores hex normais (#**ffffff** pro branco), podemos criar cores a partir de seu valor RGB, passando o valor de cada canal (Red, Green, Blue) separadamente (valor entre 0 e 255):

```
/* todos equivalentes */  
color: white;  
color: #ffffff;  
color: rgb(255, 255, 255);
```

Porém, existe uma função chamada `RGBA` que recebe um quarto argumento, o chamado canal Alpha. Na prática, seria como a opacidade daquela cor (um valor entre 0 e 1):

```
/* branco com 80% de opacidade */  
color: rgba(255,255,255, 0.8);
```

No exemplo da foto, queríamos apenas o fundo do texto em preto translúcido (o texto não). Em vez de opacity, podemos fazer então:

```
p {  
  background-color: rgba(0,0,0,0.3);  
  color: white;  
}
```



****Lorem ipsum****

5.10 PREFIXOS

Muitos recursos do HTML5 e do CSS3 ainda são experimentais. Isso quer dizer que foram incluídos no rascunho da especificação mas ainda não são 100% oficiais. As especificações ainda estão em aberto e vai demorar algum tempo até elas serem fechadas.

Existem recursos mais estáveis e menos estáveis. Alguns já estão bastante maduros e há bastante tempo na spec, e não são esperadas mais mudanças. Por outro lado, alguns são bem recentes e talvez ainda possa haver mudança até a aprovação final da especificação.

Os navegadores desejam implementar os novos recursos antes mesmo da especificação terminar, para que os desenvolvedores possam começar a usar as novas propriedades e experimentá-las na prática. Entretanto, como a sintaxe final depois de aprovada pode ser diferente, os navegadores implementam as novas propriedades instáveis usando nomes provisórios.

A boa prática é pegar o nome da propriedade e colocar um **prefixo** específico do fabricante na frente. Quando a especificação ficar estável, tira-se esse prefixo e suporta-se a sintaxe oficial.

As bordas arredondadas, por exemplo, hoje são suportadas em todos os navegadores modernos com o nome normal da propriedade a `border-radius`. Mas até o Firefox 3.6, por exemplo, o suporte da Mozilla era experimental e a propriedade era chamada de `-moz-border-radius` nesse navegador. No Chrome até a versão 3, precisávamos usar o prefixo deles com `-webkit-border-radius`.

Os prefixos dos fabricantes mais famosos são:

- **-webkit-**: navegadores Webkit (Chrome, Safari, iOS, Android)
- **-moz-**: Firefox (Mozilla)

- **-ms-**: Internet Explorer (Microsoft)
- **-o-**: Opera

É preciso consultar tabelas de compatibilidade para saber qual navegador suporta qual propriedade e se é necessário usar prefixos para certas versões. Se quisermos o máximo de compatibilidade, precisamos colocar vários prefixos ao mesmo tempo:

```
p {  
    /* Chrome até versão 3, Safari até versão 4 */  
    -webkit-border-radius: 5px;  
  
    /* Firefox até versão 3.6 */  
    -moz-border-radius: 5px;  
    /* Todas as versões modernas dos navegadores,  
    incluindo IE e Opera que nunca precisaram de  
    prefixo pra isso */  
    border-radius: 5px;  
}
```

Nos casos de CSS3 que tratamos até agora (border-radius, text-shadow, box-shadow, rgba), todos os navegadores modernos já suportam sem uso de prefixos. Você só precisará deles se quiser suportar versões antigas (nesse caso, consulte as documentações para saber o que é necessário e quando).

5.11 CSS3: GRADIENTES

O CSS3 traz também suporte à declaração de gradientes sem que precisemos trabalhar com imagens. Além de ser mais simples, a página fica mais leve e a renderização fica melhor por se adaptar a todo tipo de resolução.

Existe suporte a gradientes lineares e radiais, inclusive com múltiplas paradas. A sintaxe básica é:

```
.linear {  
    background: linear-gradient(white, blue);  
}  
  
.radial {  
    background: radial-gradient(white, blue);  
}
```

Podemos ainda usar gradientes com angulações diferentes e diversas paradas de cores:

```
.gradiente {  
    background: linear-gradient(45deg, #f0f9ff 0%, #cbebff 47%, #a1dbff 100%);  
}
```

Prefixos

A especificação de gradientes já está em sua versão final e já é implementada sem prefixos em todos os browsers.

Mas, enquanto a especificação ainda estava em rascunho, antes de seu final, uma sintaxe diferente

nos gradientes era usada. Isso quer dizer que versões mais antigas dos navegadores que suportavam gradientes esperam uma sintaxe diferente.

Como as especificações em rascunho são implementadas prefixadas nos navegadores, é fácil colocar essas regras com sintaxe diferente para esses navegadores. O problema é que o código fica grande e difícil de manter.

A versão atual da especificação suporta um primeiro argumento que indica a direção do gradiente:

```
.linear {  
    background: linear-gradient(to bottom, white, blue);  
}
```

O código anterior indica um gradiente do branco para o azul vindo de cima **para baixo**. Mas na versão suportada antes do rascunho dos gradientes, o mesmo código era escrito com **top** ao invés de **to bottom**:

```
.linear {  
    background: -webkit-linear-gradient(top, white, blue);  
    background: -moz-linear-gradient(top, white, blue);  
    background: -o-linear-gradient(top, white, blue);  
}
```

Pra piorar, versões bem mais antigas do WebKit - notadamente o Chrome até versão 9 e o Safari até versão 5 -, usavam uma sintaxe ainda mais antiga e complicada:

```
.linear {  
    background: -webkit-gradient(linear, left top, left bottom,  
        color-stop(0%, white), color-stop(100%, blue));  
}
```

Se quiser o máximo de compatibilidade, você terá que incluir todas essas variantes no código CSS.

Gambiarras pro IE antigo

O IE só suporta gradientes CSS3 a partir da versão 10, mas desde o IE6 era possível fazer gradientes simples usando um CSS proprietário da Microsoft:

```
.linear {  
    filter: progid:DXImageTransform.Microsoft.gradient(  
        startColorstr='#ffffff', endColorstr='#0000ff', GradientType=0);  
}
```

O CSS acima faz um gradiente linear do topo para baixo, como nos outros exemplos, funcionar no IE6 até IE9. O IE10 já não aceita mais essa gambiarra e exige que você use a sintaxe oficial do CSS3 que vimos no início.

Geração de gradientes

É comum também a configuração de um `background` simples e sólido antes do gradiente, a ser usado pelos navegadores mais抗igos. Como eles não entendem gradientes, usarão a cor sólida e terão

um design adequado e usável. Os navegadores mais novos vão ler a regra do gradiente e ignorar a cor sólida (progressive enhancement):

```
.gradiente {  
    background: #cbebff;  
    background: linear-gradient(45deg, #f0f0ff 0%, #cbebff 47%, #a1dbff 100%);  
}
```

Uma ferramenta bastante útil e recomendada é o **Ultimate CSS Gradient Generator** da ColorZilla. Ela nos permite criar gradientes CSS3 visualmente como num editor de imagens. Além disso, já implementa todos os hacks e prefixos para navegadores diferentes. Há até uma opção que permite que enviamos uma imagem com o design de um gradiente e ele identifica as cores e gera o código correto.

<http://www.colorzilla.com/gradient-editor/>

Recomendamos fortemente o uso dessa ferramenta para gerar gradientes CSS3.

5.12 PROGRESSIVE ENHANCEMENT

Nesse ponto, você pode estar pensando nos navegadores antigos. Bordas redondas, por exemplo, só funcionam no IE a partir da versão 9. Até o IE8 não há como fazer isso com CSS (nem com prefixos).

O que fazer então? Muitos usuários no Brasil ainda usam IE8 e até versões mais antigas como IE7 e talvez IE6.

O melhor é não usar esses recursos modernos por causa dos usuários de navegadores antigos? **Não!**

Não vamos sacrificar a experiência da maioria dos usuários de navegadores modernos por causa do cada vez menor número de pessoas usando navegadores antigos, mas também não queremos esquecer os usuários de navegadores antigos e deixá-los sem suporte.

Progressive enhancement e graceful degradation

A ideia é fazer seu site funcionar em qualquer navegador, sem prejudicar os navegadores mais antigos e sem deixar de usar os novos recursos em navegadores novos. Graceful degradation foi o nome da primeira técnica a pregar isso; o objetivo era montar seu site voltado aos navegadores modernos e fazer com que ele degradasse "graciosamente", removendo funcionalidades não suportadas.

A abordagem mais recente, chamada **progressive enhancement** tem uma ideia parecida mas ao contrário: comece desenvolvendo as funcionalidades normalmente e vá acrescentando pequenas melhorias mesmo que só funcionem nos navegadores modernos.

Com CSS3, podemos usar progressive enhancement. Não é possível ainda ter um site que dependa hoje totalmente do CSS3. Mas é possível desenvolver seu site e acrescentar melhorias progressivamente usando CSS3.

Ou seja, faça um layout que fique usável com bordas quadradas mas use bordas redondas para deixá-lo melhor, mais bonito, nos navegadores mais modernos.

Saiba mais no blog da Caelum:

<http://blog.caelum.com.br/css3-e-progressive-enhancement/>

5.13 EXERCÍCIOS: VISUAL CSS3

- Dê um ar mais atual para nossa home colocando alguns efeitos nos painéis.

Use `border-radius` e `box-shadow` no painel em si para destacá-lo mais. E use um `text-shadow` sutil para deixar o título do painel mais destacado.

```
.painel {  
    border-radius: 4px;  
    box-shadow: 1px 1px 4px #999;  
}  
  
.painel h2 {  
    text-shadow: 3px 3px 2px #fff;  
}
```



Veja o resultado no navegador.

- O `box-shadow` também aceita a criação de bordas internas aos elementos além da borda externa. Basta usar a opção `inset`, **altere** no seletor que criamos no exercício anterior:

```
.painel {  
    box-shadow: inset 1px 1px 4px #999;  
}
```

Teste na sombra dos painéis que fizemos antes.

- O `border-radius` pode também ser configurado para bordas específicas apenas e até de tamanhos diferentes se quisermos.

```
.busca {  
    border-top-left-radius: 5px;  
    border-top-right-radius: 5px;  
}
```

4. No CSS3, podemos usar cores com **canal Alpha** para translucência usando a sintaxe do RGBA. Altere a sombra do título do painel para branco com 80% de opacidade.

```
.painel h2 {  
    text-shadow: 3px 3px 2px rgba(255, 255, 255, 0.8);  
}
```

5. Use gradientes nos painéis de produtos na Home. Não retire o `background-color` que já está no seletor, ele serve para os navegadores que não suportam o `gradient`. Adicionar o `gradient` a baixo do `background-color` existente.

O painel novidade, por exemplo, poderia ter:

```
.novidades {  
    background: linear-gradient(#f5dcfc, #bebef4);  
}
```

E o painel de mais vendidos:

```
.mais-vendidos {  
    background: linear-gradient(#dcdf5, #f4bebe);  
}
```

PREFIXOS NO EXERCÍCIO

Usamos no exercício a versão oficial da especificação sem prefixos que funciona nas últimas versões do Firefox, Chrome, Opera, Safari e Internet Explorer.

Note que não estamos suportando versões antigas desses navegadores de propósito. Se quiser, você pode adicionar as outras variantes de prefixos para suportá-los. Ou usar uma ferramenta como o Collorzilla Gradient Generator para automatizar:

<http://www.colorzilla.com/gradient-editor/>

Consulte o suporte nos browsers aqui: <http://caniuse.com/css-gradients>

5.14 CSS3 TRANSITIONS

Com as transitions, conseguimos animar o processo de mudança de algum valor do CSS.

Por exemplo: temos um elemento na posição `top:10px` e, quando passarmos o mouse em cima (`:hover`), queremos que o elemento vá para `top:30px`. O CSS básico é:

```
#teste {  
    position: relative;  
    top: 10px;  
}
```

```
#teste:hover {  
    top: 30px;  
}
```

Isso funciona, mas o elemento é deslocado de uma vez quando passamos o mouse. E se quisermos algo mais sutil? Uma animação desse valor mudando lentamente, mostrando o elemento se deslocando na tela? Usamos CSS3 Transitions.

Sua sintaxe possui vários recursos, mas seu uso é mais simples, para esse nosso caso, seria apenas:

```
#teste:hover {  
    transition: top 2s;  
}
```

Isso indica que queremos animar a propriedade top durante 2 segundos.

Por padrão, a animação é linear, mas temos outros tipos para animações mais suaves:

- `linear` - velocidade constante na animação;
- `ease` - redução gradual na velocidade da animação;
- `ease-in` - aumento gradual na velocidade da animação;
- `ease-in-out` - aumento gradual, depois redução gradual na velocidade da animação;
- `cubic-bezier(x1,y1,x2,y2)` - curva de velocidade para animação customizada (avançado)

```
#teste:hover {  
    transition: top 2s ease;  
}
```

Para explorar o comportamento dos tipos de animações disponíveis, e como criar uma curva de velocidade customizada para sua animação, existe uma ferramenta que auxilia a criação do `cubic-bezier` : <http://www.roblapla.com/examples/bezierBuilder/>

Podemos ainda usar mais de uma propriedade ao mesmo tempo, incluindo cores!

```
#teste {  
    position: relative;  
    top: 10px;  
    color: white;  
}  
  
#teste:hover {  
    top: 30px;  
    color: red;  
    transition: top 2s, color 1s ease;  
}
```

Se quisermos a mesma animação, mesmo tempo, mesmo efeito para todas as propriedades, podemos usar o atalho `all` (que já é o valor padrão, inclusive):

```
#teste:hover {  
    transition: all 2s ease;  
}
```

5.15 CSS3 TRANSFORMS

Com essa nova especificação, agora é possível alterar propriedades visuais dos elementos antes impossíveis. Por exemplo, agora podemos alterar o ângulo de um elemento, mostrá-lo em uma escala maior ou menor que seu tamanho padrão ou alterar a posição do elemento sem sofrer interferência de sua estrutura.

Translate

```
.header {  
    /* Move o elemento no eixo horizontal */  
    transform: translateX(50px);  
}  
  
.main {  
    /* Move o elemento no eixo vertical */  
    transform: translateY(-20px);  
}  
  
.footer {  
    /* Move o elemento nos dois eixos (X, Y) */  
    transform: translate(40px, -20px);  
}
```



Rotate

```
#menu-departamentos {  
    transform: rotate(-10deg);  
}
```



Scale

```
#novidades li {  
    /* Alterar a escala total do elemento */  
    transform: scale(1.2);  
}  
  
#mais-vendidos li {  
    /* Alterar a escala vertical e horizontal do elemento */  
    transform: scale(1, 0.6);  
}
```



Skew

```
footer {  
    /* Distorcer o elemento no eixo horizontal */  
    transform: skewX(10deg);  
}  
  
#social {  
    /* Distorcer o elemento no eixo vertical */  
    transform: skewY(10deg);  
}
```



É possível combinar várias transformações no mesmo elemento, basta declarar uma depois da outra:

```
html {  
    transform: rotate(-30deg) scale(0.4);  
}
```

PREFIXOS NOS EXERCÍCIOS

No exercício, optamos por usar sintaxes que funcionam em todos os browsers mas apenas em suas últimas versões. Isso quer dizer que usamos tudo sem prefixos. Se quiser suportar versões mais antigas, adicione mais prefixos.

Consulte a compatibilidade e prefixos:

<http://caniuse.com/css-transitions>

<http://caniuse.com/transforms2d>

5.16 EXERCÍCIOS: CSS3 TRANSFORM E TRANSITION

1. Quando o usuário passar o mouse em algum produto dos painéis de destaque, mostre uma sombra por trás com `box-shadow`. Use também uma transição com `transition` para que essa sombra apareça suavemente:

```
.painel li:hover {  
  box-shadow: 0 0 5px #333;  
  transition: box-shadow 0.7s;  
}
```

Teste o resultado no navegador.

2. **Altere a regra anterior** para também colocar agora um fundo branco no elemento. Anime esse fundo também, fazendo um efeito tipo *fade*.

Na regra `transition` de antes, podemos indicar que todas as propriedades devem ser animadas; para isso, podemos usar a keyword `all` ou simplesmente omitir esse argumento.

```
.painel li:hover {  
  background-color: rgba(255, 255, 255, 0.8);  
  box-shadow: 0 0 5px #333;  
  transition: all 0.7s;  
}
```

3. Mais coisas de CSS3! Ainda quando passar o mouse em cima do item do painel, queremos aumentar o elemento em 20%, dando uma espécie de zoom.

Use `CSS transform` pra isso, com `scale`. **Adicione** na regra anterior (sem remover o que já tínhamos):

```
.painel li:hover {  
  transform: scale(1.2);  
}
```

Teste e repare como o `scale` também é animado suavemente. Isso porque nossa transição estava com `all`.

4. **Altere** a regra anterior do `transform` pra também fazer o elemento rotacionar suavemente em 5 graus no sentido anti-horário:

```
.painel li:hover {  
    transform: scale(1.2) rotate(-5deg);  
}
```



5. (opcional) Faça os elementos ímpares girarem em sentido anti-horário e os pares no sentido horário!

No exercício anterior, fizemos todos girarem anti-horário. Vamos sobrescrever essa regra para os elementos pares usando o seletor `:nth-child`:

```
.painel li:nth-child(2n):hover {  
    transform: scale(1.2) rotate(5deg);  
}
```

6. (opcional) Repare como a animação ocorre apenas quando passamos o mouse em cima, mas quando tiramos, a volta do efeito não é animada.

Podemos habilitar a animação na volta do elemento para o estado normal movendo as regras de transição para o `li` em si (e não só no `:hover`).

```
.painel li {  
    transition: 0.7s;  
}
```

7. (opcional) Um terceiro argumento para a função de transição é a função de animação, que controla como o efeito executa de acordo com o tempo.

Por padrão, os efeitos são **lineares**, mas podemos obter resultados mais interessantes com outras opções como `ease`, `ease-in`, `ease-out` (e até o avançado `cubic-bezier()`).

Por exemplo, podemos **alterar** o que fizemos nos exercícios anteriores:

```
.painel li:hover {  
    transition: 0.7s ease-in;  
}  
  
.painel li {  
    transition: 0.7s ease-out;  
}
```

5.17 PARA SABER MAIS: ESPECIFICIDADE DE SELETORES CSS

Quando declaramos uma série de seletores CSS, é bem possível que nos deparamos com situações em que mais de um seletor esteja aplicando propriedades visuais ao mesmo elemento do HTML. Nesse caso é necessário saber qual seletor tem precedência sobre os outros, a fim de resolver conflitos e garantir que as propriedades desejadas estejam sendo aplicadas aos elementos corretos.

O comportamento padrão dos seletores CSS, quando não há conflitos entre propriedades, é que as propriedades de mais de um seletor para um mesmo elemento sejam acumuladas. Veja no exemplo a seguir:

Estrutura HTML

```
<p>Texto do parágrafo em destaque</p>  
<p>Texto de um parágrafo comum</p>
```

Seletores CSS

```
p {  
    color: navy;  
}  
  
p {  
    font-size: 16px;  
}
```

No exemplo anterior, utilizamos o mesmo seletor duas vezes no CSS. Isso faz com que as propriedades sejam acumuladas em todos os elementos selecionados. No caso, todos os elementos da tag `p` em nosso documento serão exibidos da cor "navy" (azul marinho) e com a fonte no tamanho "16px".

Quando há conflito entre propriedades de seletores equivalentes, ou até mesmo em um mesmo seletor, é aplicada a propriedade declarada depois, como no exemplo a seguir:

```
p {  
    color: navy;  
    font-size: 12px;  
}  
  
p {  
    font-size: 16px;  
}
```

Nesse caso, há conflito entre as propriedades `font-size` declaradas nos seletores. Como os

seletores são equivalentes, os parágrafos serão exibidos com a fonte no tamanho "16px". A declaração anterior, com valor de "12px" é sobreescrita pela nova propriedade declarada mais abaixo no nosso CSS. A cor "navy" continua aplicada a todos os parágrafos do documento.

Especificidade de Seletores CSS

Seletores equivalentes têm suas propriedades sobreescritas conforme são declaradas. Mas o que acontece quando temos diferentes tipos de seletores?

Cada tipo de seletor tem um *peso* diferente quando o navegador os interpreta e aplica suas propriedades declaradas aos elementos de um documento. Existe uma maneira bem simples de saber como funcionam esses pesos porque eles fazem parte da especificação do CSS. Para ficar um pouco mais fácil é só pensarmos em uma regra simples: **quanto mais específico for o seletor, maior seu valor**. Por isso esse peso, ou valor, que cada seletor tem, recebe o nome de **especificidade**.

O seletor de tag, por exemplo `div {}`, é bem genérico. As propriedades declaradas nesse seletor serão aplicadas a todos os elementos `div` do nosso documento, não levando em consideração qualquer atributo que eles possam ter. Por esse motivo, o seletor de tag tem valor baixo de especificidade.

O seletor de classe, por exemplo `.destaque {}`, é um pouco mais específico, nós decidimos quais elementos têm determinado valor para esse atributo nos elementos do HTML, portanto o valor de especificidade do seletor de classe é maior do que o valor de especificidade do seletor de tag.

O seletor de id, por exemplo `#cabecalho {}`, é bem específico, pois só podemos ter um único elemento com determinado valor para o atributo `id`, então seu valor de especificidade é o maior entre os seletores que vimos até agora.

E quando temos seletores compostos ou combinados? Como calcular essa especificidade?

Podemos adicionar um ponto em cada posição do valor de um seletor para chegarmos ao seu valor de especificidade. Para isso vamos utilizar uma tabela para nos ajudar a conhecer esses valores, e a seguir vamos aplicar o cálculo a alguns seletores propostos.

Valor de Especificidade dos Seletores CSS		
Seletor de Id ex: <code>#rodape { font-size: 11pt; }</code>	Seletor de Classe ex: <code>.conteudo { width: 960px; }</code>	Seletor de Tag ex: <code>div { color: green; }</code>
1	1	1

Seguindo os valores descritos na tabela acima, podemos calcular o valor de especificidade para qualquer seletor CSS, por exemplo:

```
p {  
    /* valor de especificidade: 001 */  
    color: blue;  
}  
  
.destaque {  
    /* valor de especificidade: 010 */  
    color: red;  
}  
  
#cabecalho {  
    /* valor de especificidade: 100 */  
    color: green;  
}
```

Nos seletores combinados e compostos, basta somar os valores em suas determinadas posições como nos exemplos a seguir:

```
#rodape p {  
    /* valor de especificidade: 101 */  
    font-size: 11px;  
}  
  
.cabecalho .conteudo h1 {  
    /* valor de especificidade 111 */  
    color: green;  
}  
  
.conteudo div p span {  
    /* valor de especificidade: 013 */  
    font-size: 13px;  
}
```

Quanto maior o valor da especificidade do seletor, maior a prioridade de seu valor, dessa maneira um seletor com valor de especificidade **013** sobrescreve as propriedades conflitantes para o mesmo elemento que um seletor com valor de especificidade **001**.

Essa é uma maneira simples de descobrir qual seletor tem suas propriedades aplicadas com maior prioridade. Por enquanto vimos somente esses três tipos de seletores CSS (tag, classe e id). No decorrer do curso vamos ver outros tipos mais avançados de seletores, e vamos complementando essa tabela para termos uma referência completa para esse cálculo.

WEB PARA DISPOSITIVOS MÓVEIS

"A iniciativa da Internet móvel é importante, informações devem ser igualmente disponíveis em qualquer dispositivo" -- Tim Berners-Lee

6.1 SITE MOBILE OU MESMO SITE?

O volume de usuários que acessam a Internet por meio de dispositivos móveis cresceu exponencialmente nos últimos anos. Usuários de iPhones, iPads e outros smartphones e tablets têm demandas diferentes dos usuários Desktop. Redes lentas e acessibilidade em dispositivos limitados e multitoque são as principais diferenças.

Como atender a esses usuários?

Para que suportemos usuários móveis, antes de tudo, precisamos tomar uma decisão: fazer um site exclusivo - e diferente - focado em dispositivos móveis ou adaptar nosso site para funcionar em qualquer dispositivo?

Vários grandes sites da Internet - como Google, Amazon, UOL, Globo.com etc - adotam a estratégia de ter um site diferente voltado para dispositivos móveis. É comum usar um subdomínio diferente como "m." ou "mobile.", como <http://m.uol.com.br>.

Essa abordagem é a que traz maior facilidade na hora de pensar nas capacidades de cada plataforma, Desktop e mobile, permitindo que entreguemos uma experiência customizada e otimizada para cada situação. Porém, há diversos problemas envolvidos:

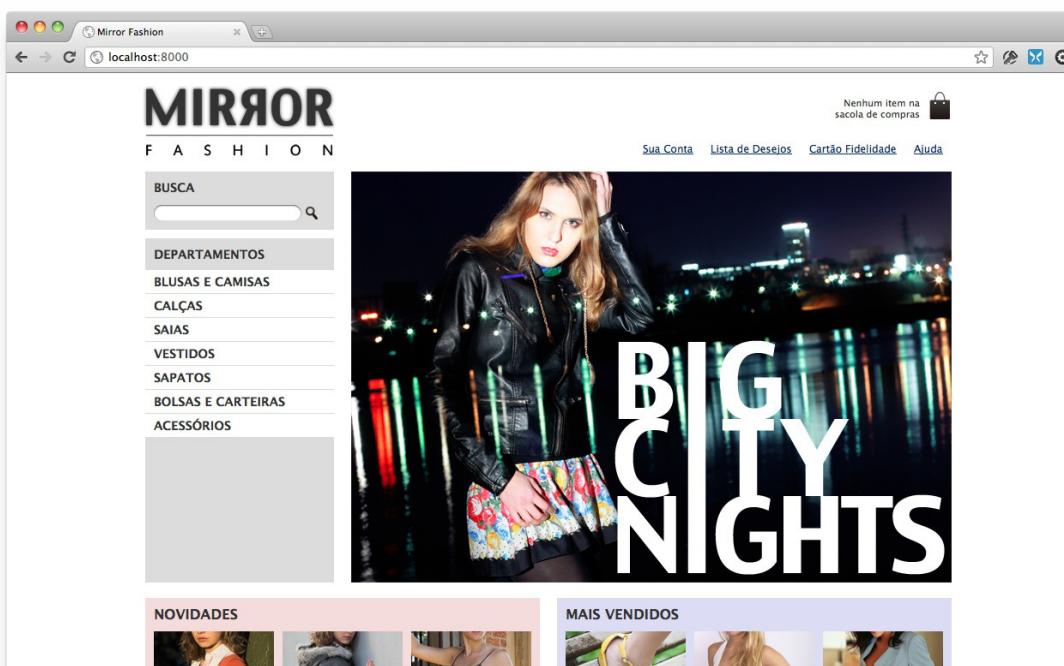
- Como atender adequadamente diversos dispositivos tão diferentes quanto um smartphone com tela pequena e um tablet com tela mediana? E se ainda considerarmos os novos televisores como Google TV? Teríamos que montar um site específico para cada tipo de plataforma?
- Muitas vezes esses sites mobile são versões limitadas dos sites de verdade e não apenas ajustes de usabilidade para o dispositivo diferente. Isso frustra o usuário que, cada vez mais, usa dispositivos móveis para completar as mesmas tarefas que antes fazia no Desktop.
- Dar manutenção em um site já é bastante trabalhoso, imagine dar manutenção em dois - um mobile e um normal.

- Você terá conteúdos duplicados entre sites "diferentes", podendo prejudicar seu SEO se não for feito com cuidado.
- Terá que lidar com redirects entre URLs móveis e normais, dependendo do dispositivo. O usuário pode receber de um amigo um link para uma página vista no site normal; mas se ele abrir esse email no celular, terá que ver a versão mobile desse link, sendo redirecionado automaticamente. E a mesma coisa no sentido contrário, ao mandar um link de volta para o Desktop.

Uma abordagem que vem ganhando bastante destaque é a de ter um único site, acessível em todos os dispositivos móveis. Adeptos da ideia da Web única (**One Web**) consideram que o melhor para o usuário é ter o mesmo site do Desktop normal também acessível no mundo móvel. É o melhor para o desenvolvedor também, que não precisará manter vários sites diferentes. E é o que garante a compatibilidade com a maior gama de aparelhos diferentes.

Certamente, a ideia não é fazer o usuário móvel acessar a página exatamente da mesma maneira que o usuário Desktop. Usando truques de CSS3 bem suportados no mercado, podemos usar a mesma base de layout e marcação porém ajustando o design para cada tipo de dispositivo.

Nossa página no Desktop agora é mostrada assim:



Queremos que, quando vista em um celular, tenha um layout mais otimizado:



COMO DESENVOLVER UM SITE EXCLUSIVO PARA MOBILE?

A abordagem que trataremos no curso é a de fazer adaptações na mesma página para ser compatível com CSS3. Como faremos caso queiramos fazer um site exclusivo para mobile?

Do ponto de vista de código, é a abordagem mais simples: basta fazer sua página com design mais enxuto e levando em conta que a tela será pequena (em geral, usa-se width de 100% para que se adapte à pequenas variações de tamanhos de telas entre smartphones diferentes).

Uma dificuldade estará no servidor para detectar se o usuário está vindo de um dispositivo móvel ou não, e redirecioná-lo para o lugar certo. Isso costuma envolver código no servidor que detecte o navegador do usuário usando o User-Agent do navegador.

É uma boa prática também incluir um link para a versão normal do site caso o usuário não queira a versão móvel.

6.2 CSS MEDIA TYPES

Desde a época do CSS2, há uma preocupação com o suporte de regras de layout diferentes para cada situação possível. Isso é feito usando os chamados *media types*, que podem ser declarados ao se invocar

um arquivo CSS:

```
<link rel="stylesheet" href="site.css" media="screen">
<link rel="stylesheet" href="print.css" media="print">
<link rel="stylesheet" href="handheld.css" media="handheld">
```

Outra forma de declarar os *media types* é separar as regras dentro do próprio arquivo CSS:

```
@media screen {
  body {
    background-color: blue;
    color: white;
  }
}

@media print {
  body {
    background-color: white;
    color: black;
  }
}
```

O *media type screen* determina a visualização normal, na tela do Desktop. É muito comum também termos um CSS com *media type print* com regras de impressão (por exemplo, retirar navegação, formatar cores para serem mais adequadas para leitura em papel etc).

E havia também o *media type handheld*, voltado para dispositivos móveis. Com ele, conseguíamos adaptar o site para os pequenos dispositivos como celulares WAP e palmtops.

O problema é que esse tipo *handheld* nasceu em uma época em que os celulares eram bem mais simples do que hoje, e, portanto, costumavam ser usados para fazer páginas bem simples. Quando os novos smartphones *touchscreen* começaram a surgir - em especial, o iPhone -, eles tinham capacidade para abrir páginas completas e tão complexas quanto as do Desktop. Por isso, o iPhone e outros celulares modernos ignoram as regras de *handheld* e se consideram, na verdade, *screen*.

Além disso, mesmo que *handheld* funcionasse nos smartphones, como trataríamos os diferentes dispositivos de hoje em dia como tablets, televisões etc?

A solução veio com o CSS3 e seus *media queries*.

6.3 CSS3 MEDIA QUERIES

Todos os smartphones e navegadores modernos suportam uma nova forma de adaptar o CSS baseado nas propriedades dos dispositivos, as **media queries** do CSS3.

Em vez de simplesmente falar que determinado CSS é para *handheld* em geral, nós podemos agora indicar que determinadas regras do CSS devem ser vinculadas a propriedades do dispositivo como tamanho da tela, orientação (landscape ou portrait) e até resolução em dpi.

```
<link rel="stylesheet" href="base.css" media="screen">
<link rel="stylesheet" href="mobile.css" media="(max-width: 480px)">
```

Outra forma de declarar os *media types* é separar as regras dentro do mesmo arquivo CSS:

```
@media screen {  
  body {  
    font-size: 16px;  
  }  
  
@media (max-width: 480px) {  
  body {  
    font-size: 12px;  
  }  
}
```

Repare como o atributo *media* agora pode receber expressões complexas. No caso, estamos indicando que queremos que as telas com largura máxima de 480px tenham uma fonte de 12px.

Você pode testar isso apenas redimensionando seu próprio navegador Desktop para um tamanho menor que 480px.

6.4 VIEWPORT

Mas, se você tentar rodar nosso exemplo anterior em um iPhone ou Android de verdade, verá que ainda estamos vendo a versão Desktop da página. A regra do *max-width* parece ser ignorada!

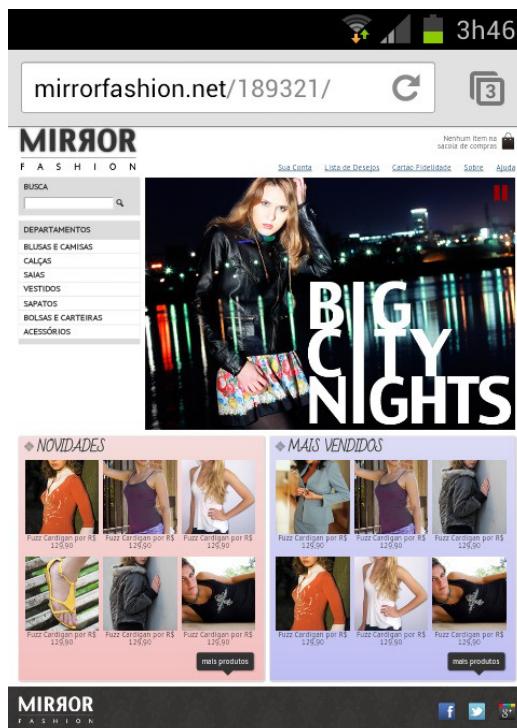


Figura 6.3: Site atual rodando num celular Android

Na verdade, a questão é que os smartphones modernos têm telas grandes e resoluções altas, justamente para nos permitir ver sites complexos feitos para Desktop. A tela de um iPhone 4S por exemplo é 960px por 640px. Celulares Android já chegam a 1280px, o mesmo de um Desktop.

Ainda assim, a experiência desses celulares é bem diferente dos Desktops. 1280px em uma tela de 4 polegadas é bem diferente de 1280px em um notebook de 13 polegadas. A resolução muda. Celulares costumam ter uma resolução em dpi bem maior que Desktops.

Como arrumar nossa página?

Os smartphones sabem que considerar a tela como 1280px não ajudará o usuário a visualizar a página otimizada para telas menores. Há então o conceito de *device-width* que, resumidamente, representa um número em pixels que o fabricante do aparelho considera como mais próximo da sensação que o usuário tem ao visualizar a tela.

Nos iPhones, por exemplo, o *device-width* é considerado como 320px, mesmo com a tela tendo uma resolução bem mais alta.

Por padrão, iPhones, Androids e afins costumam considerar o tamanho da tela visível, chamada de *viewport* como grande o suficiente para comportar os sites Desktop normais. Por isso a nossa página foi mostrada sem zoom como se estivéssemos no Desktop.

A Apple criou então uma solução que depois foi copiada pelos outros smartphones, que é configurar o valor que julgarmos mais adequado para o *viewport*:

```
<meta name="viewport" content="width=320">
```

Isso faz com que a tela seja considerada com largura de 320px, fazendo com que nosso layout mobile finalmente funcione e nossas *media queries* também.

Melhor ainda, podemos colocar o *viewport* com o valor *device-width* definido pelo fabricante, dando mais flexibilidade com dispositivos diferentes com tamanhos diferentes:

```
<meta name="viewport" content="width=device-width">
```

6.5 EXERCÍCIOS: ADAPTAÇÕES PARA MOBILE

1. Vamos adaptar nossa home page (**index.html**) para mobile.



Comece declarando a meta tag com o *viewport* dentro do `<head>` da **index.html**:

```
<meta name="viewport" content="width=device-width">
```

Vamos escrever nosso CSS de adaptação no arquivo **estilos.css**, no final do documento, depois de todos os estilos que já temos. Para que os estilos só se apliquem em resoluções de no **máximo 320px** (celulares comuns), vamos usar *media queries* e colocar dentro dela todos os estilos que queremos alterar.

```
@media (max-width: 320px) {  
    /* adicionar aqui os novos estilos para mobile */  
}
```

2. Nossa página hoje tem o tamanho fixo em 940px e é centralizada com o uso do seletor **.container**. Para deixarmos a página mais flexível nos celulares, precisamos remover esse tamanho absoluto e colocar algum que faça mais sentido em telas menores, onde queremos ocupar quase que a tela toda, deixando apenas uma pequena margem. Para isso, **adicione o código abaixo dentro da @media (max-width: 320px)** do arquivo **estilos.css**:

```
.container {  
    width: 96%;  
}
```

Já é possível redimensionar a tela para 320px e ver que o site começa a se adaptar. Mas ainda há bastante trabalho pela frente.

3. Próximo passo: vamos ajustar os elementos do topo da página. Vamos centralizar o logotipo na página, esconder as informações secundárias sobre a sacola e ajustar o menu opções para ficar abaixo do logo e não mais posicionado à direita.

Adicionar dentro do `@media (max-width: 320px)` :

```
header h1 {  
    text-align: center;  
}  
  
header h1 img {  
    max-width: 50%;  
}  
  
.sacola {  
    display: none;  
}  
  
.menu-opcoes {  
    position: static;  
    text-align: center;  
}
```

Lembre-se que, anteriormente, nosso menu estava com `position: absolute` para ficar a direita do logo. Agora, queremos deixá-lo fluir abaixo do logo; bastou restaurar o `position: static`.

Teste novamente com o navegador redimensionado. Está melhorando?

4. Ajustamos a posição do menu do topo e, automaticamente, os links se posicionaram formando duas linhas. Mas repare como estão próximos uns dos outros. Será que o nosso usuário consegue clicar neles usando seu celular? Vamos aumentar o espaço entre eles.

Adicionar dentro do `@media (max-width: 320px)` :

```
.menu-opcoes ul li {  
    display: inline-block;  
    margin: 5px;  
}
```

5. Ajuste a seção de **busca**, o **menu da esquerda** e a **imagem de destaque**. Como eles são muito grandes, em mobile, é melhor renderizarmos um em cima do outro sem quebrar em colunas.

Vamos ocupar 100% da tela com o menu e a busca. A imagem de destaque (banner) deverá ser redimensionada para ocupar 100% da tela e não estourar o tamanho.

Adicionar dentro do `@media (max-width: 320px)` :

```
.busca,  
.menu-departamentos,  
.banner-destaque img {
```

```
margin-right: 0;  
width: 100%;  
}
```

Teste esse passo no navegador redimensionado.

6. Nossa página está ficando boa em mobile. Falta apenas ajustarmos os painéis de destaque de produtos.

Por ora, eles estão com tamanhos absolutos ocupando metade da tela e mostrando 6 elementos, com 3 por linha. Vamos manter o painel com 3 elementos por linha, mas vamos fazer os dois painéis encaixarem um em cima do outro. Para isso, basta tirarmos a restrição de largura do painel para ele ocupar a tela toda.

Adicionar dentro do `@media (max-width: 320px)` :

```
.painel {  
    width: auto;  
}
```

Com relação aos produtos nos painéis. Vamos precisar redimensioná-los para encaixar 3 em cada linha. Uma boa maneira é colocar cada elemento com 30% do painel, totalizando 90%, e deixando espaço para as margens.

Já a imagem interna de cada produto deverá ocupar 100% do seu quadrado (o `` que ajustamos), senão as imagens vão estourar o layout em certos tamanhos.

Adicionar dentro do `@media (max-width: 320px)` :

```
.painel li {  
    width: 30%;  
}  
  
.painel img {  
    width: 100%;  
}
```

Teste esse passo no navegador redimensionado.

7. O que acontece em resoluções maiores de 320px? Nosso design volta ao padrão de 940px e ficamos com scroll horizontal. A maioria dos smartphones tem 320px de largura, mas nem todos, e nosso layout não se ajusta bem a esses outros. Até mesmo aqueles com 320px de largura, ao girar o aparelho em modo paisagem, a resolução é maior (480px num iPhone e mais de 500px em muitos Androids).

O melhor era que nosso layout adaptável fosse usado não só em 320px mas em diversas resoluções intermediárias antes dos 940px que estabelecemos para o site Desktop.

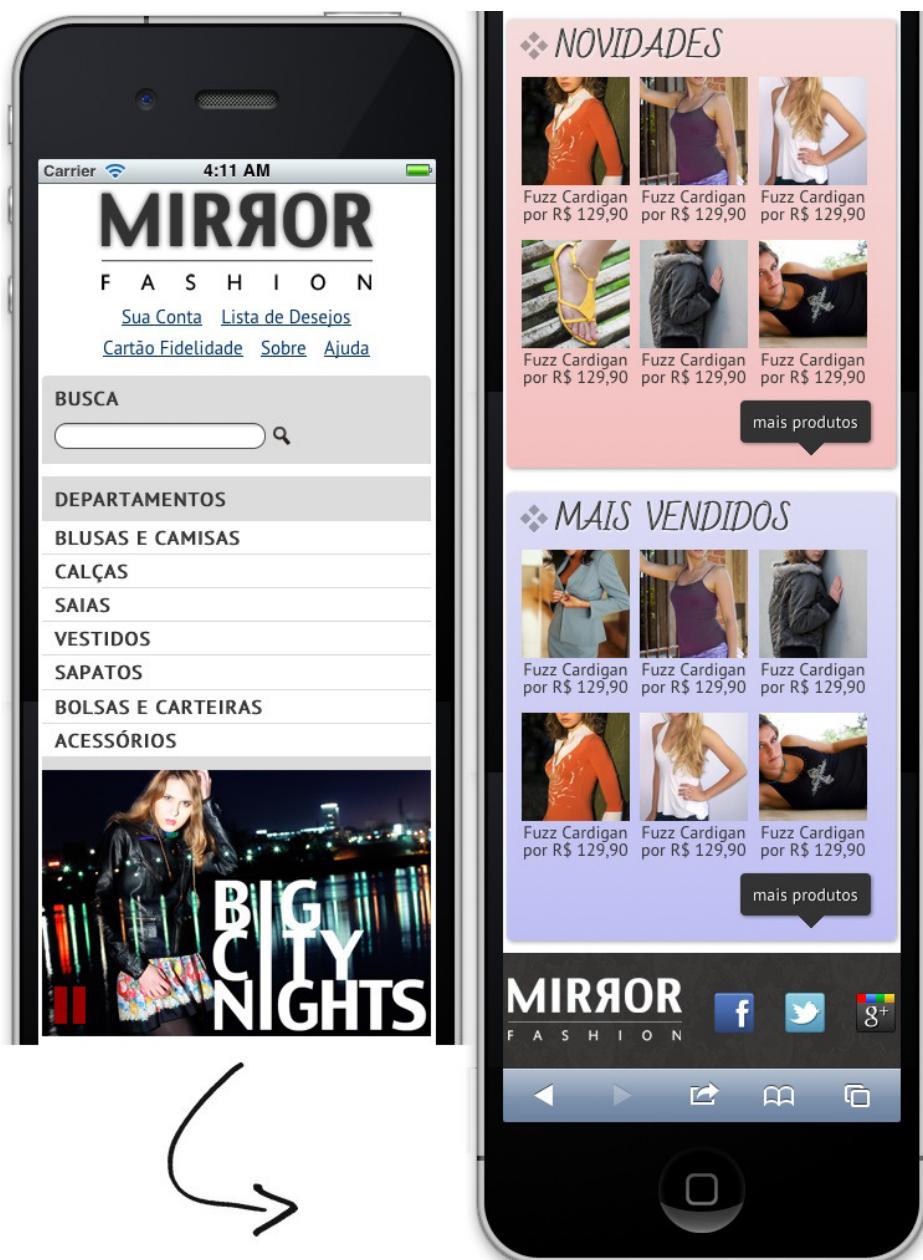
Vamos **alterar** nossa *media query* que estamos usando no arquivo **estilos.css** para aplicar o CSS de adaptação a qualquer tamanho de tela menor que os 940px do Desktop (ou seja, no **máximo 939px**):

Alterando a media query que fizemos nos exercícios anteriores:

```
@media (max-width: 939px) {  
    /* estilos para mobile que já fizemos */  
}
```

Teste, novamente, redimensionando o navegador para várias resoluções diferentes.

Repare que, como fizemos um **layout fluido**, baseado em porcentagens, os elementos se ajustam a diferentes resoluções sem esforço. É uma boa prática usar porcentagens e, sempre que possível, evitar o uso de valores absolutos em pixels.



6.6 RESPONSIVE WEB DESIGN

Repare o que fizemos nesse capítulo. Nossa página, com o mesmo HTML e pequenos ajustes de CSS, suporta diversas resoluções diferentes, desde a pequena de um celular até um Desktop.

Essa prática é o que o mercado chama de **Web Design Responsivo**. O termo surgiu num famoso artigo de Ethan Marcotte e diz exatamente o que acabamos de praticar. São 3 os elementos de um design responsivo:

- layout fluído usando medidas flexíveis, como porcentagens;
- *media queries* para ajustes de design;
- uso de imagens flexíveis.

Nós aplicamos os 3 princípios na nossa adaptação da Home pra mobile. A ideia do responsivo é que a página se **adapte a diferentes condições**, em especial a diferentes resoluções. E, embora o uso de porcentagens exista há décadas na Web, foi a popularização das *media queries* que permitiram layouts verdadeiramente adaptativos.

6.7 MOBILE-FIRST

Nosso exercício seguiu o processo que chamamos de *desktop-first*. Isso significa que tínhamos nossa página codificada para o layout Desktop e, num segundo momento, adicionamos as regras para adaptação a mobile.

Na prática, isso não é muito interessante. Repare como tivemos que **desfazer** algumas coisas do que tínhamos feito no nosso layout para desktop: tiramos alguns posicionamentos e desfizemos diversos ajustes na largura de elementos.

É muito mais comum e recomendado o uso da prática inversa: o **Mobile-first**. Isto é, começar o desenvolvimento pelo mobile e, depois, adicionar suporte a layouts desktop. No código, não há nenhum segredo: vamos apenas usar **mais *media queries min-width*** ao invés de **max-width**, mais comum em códigos desktop-first.

A grande mudança do mobile-first é que ela permite uma abordagem muito mais simples e evolutiva. Inicia-se o desenvolvimento pela área mais simples e limitada, com mais restrições, o mobile. O uso da tela pequena vai nos forçar a criar páginas mais simples, focadas e objetivas. Depois, a adaptação pra Desktop com *media queries*, é apenas uma questão de readaptar o layout.

A abordagem desktop-first começa pelo ambiente mais livre e vai tentando cortar coisas quando chega no mobile. Esse tipo de adaptação é, na prática, muito mais trabalhosa.

Nós recomendamos o uso do mobile-first. E usaremos essa prática no curso a partir das próximas páginas, assim você poderá comparar os dois estilos.

6.8 EXERCÍCIOS OPCIONAIS: VERSÃO TABLET

- Nosso layout anterior tem dois comportamentos: um layout fixo em 940px otimizado para Desktops e outro construído para telas pequenas, mas que é aplicado para qualquer resolução abaixo de 939px.

Repare que, em resoluções altas (menores que 940px), nosso design mobile não fica tão bonito, embora continue funcional! Podemos usar mais *media queries* para ajustar outros detalhes do layout conforme o tamanho da tela varia entre 320px e 939px.

Vamos **adicionar** essas *media queries* no arquivo **estilos.css**, **dentro** da *media query* que fizemos no exercício anterior (**@media (max-width: 939px)**).

Por exemplo, podemos usar 2 colunas no nosso menu quando chegarmos em 480px (um iPhone em paisagem).

Adicionar dentro da **@media (max-width: 939px)** :

```
@media (min-width: 480px) {  
  header h1 {  
    margin: 5px 0;  
  }  
  
  .menu-departamentos {  
    padding-bottom: 10px;  
    margin-bottom: 10px;  
  }  
  
  .menu-departamentos nav > ul {  
    column-count: 2;  
  }  
}
```

Em telas um pouco maiores, como tablets (um iPad tem 768px por exemplo), podemos querer fazer outros ajustes com outra *media query*.

Adicionar dentro da **@media (max-width: 939px)** :

```
@media (min-width: 720px) {  
  header h1 {  
    text-align: left;  
  }  
  
  .menu-opcoes {  
    position: absolute;  
  }  
  
  .sacola {  
    display: block;  
  }  
  
  .painel li {  
    width: 15%;  
  }  
  
  .busca,
```

```
.menu-departamentos {  
    margin-right: 1%;  
    width: 30%;  
}  
  
.menu-departamentos nav > ul {  
    column-count: 1;  
}  
  
.banner-destaque img {  
    width: 69%;  
}  
}
```

Teste agora no navegador. Redimensione em diversos tamanhos desde o pequeno 320px até o Desktop grande. Veja a responsividade do nosso design, se ajustando a diversos tamanhos de tela.

2. (desafio) Adapte o layout da página **sobre.html** para mobile, também. Faça uma solução mobile completa!

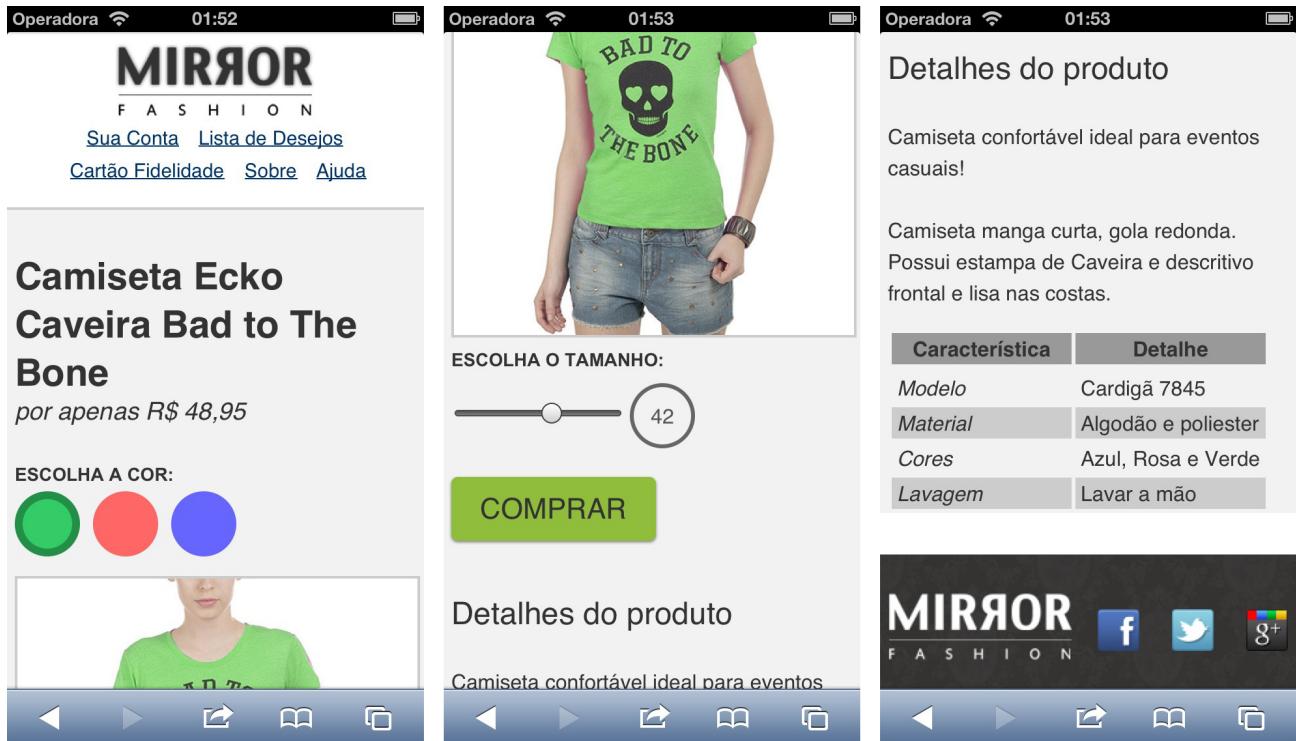
PROGRESSIVE ENHANCEMENT E MOBILE-FIRST

"Qualquer tolo consegue escrever código que um computador consiga entender. Bons programadores escrevem código que humanos conseguem entender." -- Martin Fowler

Vamos criar a próxima página da nossa loja, que será uma página para mostrar os detalhes de um produto após o usuário clicar em um produto na lista da home. Essa página vai mostrar uma foto grande, mostrar opções de cores e preço, exibir a descrição do produto e permitir que o usuário faça a compra.

Nosso designer criou um design para essa página, com estilo mais minimalista e focado no conteúdo a ser exibido.

Eis nosso design aplicado na tela do iPhone para visualizarmos:



7.1 FORMULÁRIO DE COMPRA

Um dos aspectos fundamentais dessa página é permitir que o usuário escolha a variação correta do produto para ele. Repare que ele pode escolher a cor e o tamanho e depois comprar o produto específico que escolheu.

Quando clicar no botão de comprar, as escolhas do usuário precisam ser enviadas para o servidor processar a compra em si. São, claro, parâmetros que o usuário pode escolher.

Esqueça por um instante o design que vimos antes e pense na **funcionalidade** que estamos tentando implementar. Queremos uma maneira do usuário **escolher entre diversas opções** e **enviar sua escolha** para o servidor. Falando assim, é quase óbvio que estamos descrevendo um `<form>`.

Queremos escolher a cor da roupa dentre 3 opções possíveis, temos componentes específicos de formulário para isso. Podemos fazer um combobox com `<select>` ou implementar **3 radio buttons**. Vamos fazer esse último.

```
<form>
  <input type="radio" name="cor">Verde
  <input type="radio" name="cor">Rosa
  <input type="radio" name="cor">Azul
</form>
```

Muito simples e funciona. Mas tem várias falhas de acessibilidade e HTML semântico. O texto que descreve cada opção, por exemplo, não deve ficar solto na página. Devemos usar o elemento `<label>` para representar isso. E associar com o respectivo input.

```
<form>
  <input type="radio" name="cor" id="verde">
  <label for="verde">Verde</label>

  <input type="radio" name="cor" id="rosa">
  <label for="rosa">Rosa</label>

  <input type="radio" name="cor" id="azul">
  <label for="azul">Azul</label>
</form>
```

Ainda temos que explicar para o usuário o que ele está escolhendo com esses radio buttons. É a frase "*Escolha a cor*" que vemos no design. Como escrevê-la no HTML? Um simples `<p>`? Não.

Semanticamente, esses 3 inputs representam a mesma coisa e devem ser agrupados em um `<fieldset>` que, por sua vez, recebe um `<legend>` com a legenda em texto apropriada.

```
<form>
  <fieldset>
    <legend>Escolha a cor</legend>

    <input type="radio" name="cor" id="verde">
    <label for="verde">Verde</label>

    <input type="radio" name="cor" id="rosa">
    <label for="rosa">Rosa</label>

    <input type="radio" name="cor" id="azul">
```

```

        <label for="azul">Azul</label>
    </fieldset>

    <button class="comprar">Comprar</button>
</form>

```

Aproveitamos e colocamos o botão de submit para enviar a escolha da compra.

Podemos ainda melhorar mais. Em vez de mostrar o nome da cor ("Verde") no `label`, podemos colocar a foto de verdade da roupa naquela cor. Uma imagem vale mais que mil palavras. Mas, claro, isso para quem enxerga. Para leitores de tela e outros browsers não visuais, vamos usar o atributo `alt` na imagem para manter sua acessibilidade.

```

<form>
    <fieldset>
        <legend>Escolha a cor</legend>

        <input type="radio" name="cor" id="verde">
        <label for="verde">
            
        </label>

        <input type="radio" name="cor" id="rosa">
        <label for="rosa">
            
        </label>

        <input type="radio" name="cor" id="azul">
        <label for="azul">
            
        </label>
    </fieldset>

    <button class="comprar">Comprar</button>
</form>

```

Se implementarmos esse código, sem visual nenhum, e testarmos no browser, teremos uma funcionalidade completa, funcional e acessível. Isso é fantástico. Resolveremos o visual depois.

7.2 SUBMISSÃO DO FORMULÁRIO

Todo formulário criado no HTML tem seus dados enviados para o servidor quando submetido. Cada campo do formulário é enviado como **parâmetro na requisição** feita ao servidor.

No formulário, podemos indicar que página (URL) vai receber os dados preenchidos. É só especificar o atributo `action`. No nosso exemplo, temos um formulário na página **produto.html** e vamos criar uma próxima página, **checkout.html**, que vai receber o produto escolhido e deixar o usuário proceder com a compra.

No formulário de produto então fazemos:

```
<form action="checkout.html">
```

Há ainda um outro atributo da tag form que indica a *maneira como os dados são enviados*. É o atributo `method` que pode receber dois valores: GET ou POST. Ambos os métodos enviam os dados do formulário ao servidor, mas o GET faz isso via parâmetros na URL enquanto o POST envia os dados no corpo da requisição (e, portanto, não é visível na barra de endereços).

```
<form action="checkout.html" method="POST">
```

7.3 EXERCÍCIOS: FORMULÁRIO DA PÁGINA DE PRODUTO

1. Vamos implementar nossa página de produto, o primeiro passo é a construção de um HTML semântico. Vamos começar criando a estrutura básica do arquivo. Isto inclui as tags doctype, html, head, body, title.

Crie o arquivo **produto.html** com uma base parecida com essa:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width">
    <title>Produto da Mirror Fashion</title>
    <link rel="stylesheet" href="css/reset.css">
    <link rel="stylesheet" href="css/estilos.css">
  </head>
  <body>
    ...
  </body>
</html>
```

Repare que já incluímos algumas tags que vimos antes no curso. Em especial, o charset com UTF-8, a tag viewport para nossa página funcionar bem em mobile e os arquivos CSS de reset.css e estilos.css.

2. Como todas as páginas fazem parte da Mirror Fashion, é muito comum que tenham o mesmo cabeçalho e o mesmo rodapé. Criamos um cabeçalho e um rodapé bacana na **index.html** e, para termos o mesmo cabeçalho e o mesmo rodapé no **produto.html**, teremos que **copiar e colar** o código do `<header>` e do `<footer>` dentro do `<body>`.

```
<body>
  <header class="container">
    <h1>
      
    </h1>

    <p class="sacola">
      Nenhum item na sacola de compras
    </p>

    <nav class="menu-opcoes">
      <ul>
        <li><a href="#">Sua Conta</a></li>
        <li><a href="#">Lista de Desejos</a></li>
        <li><a href="#">Cartão Fidelidade</a></li>
      </ul>
    </nav>
  </header>
```

```

        <li><a href="sobre.html">Sobre</a></li>
        <li><a href="#">Ajuda</a></li>
    </ul>
</nav>
</header>

<!-- adicionar o conteúdo da página -->

<footer>
    <div class="container">
        

        <ul class="social">
            <li><a href="http://facebook.com/mirrorfashion">Facebook</a></li>
            <li><a href="http://twitter.com/mirrorfashion">Twitter</a></li>
            <li><a href="http://plus.google.com/mirrorfashion">Google+</a></li>
        </ul>
    </div>
</footer>
</body>

```

O HTML sozinho não tem recursos muito interessantes para se reaproveitar pedaços de código entre páginas. Para isso é preciso o uso de servidores, como por exemplo o PHP.

3. Edite a página **produto.html** e, entre as tags `<header>` e `<footer>`, adicione um formulário com radios e labels para a escolha da cor. Também usaremos o atributo `alt` nas imagens para melhorar a acessibilidade:

```

<section class="produto">
    <h2>Camiseta Ecko Caveira Bad to The Bone</h2>
    <p>por apenas R$ 48,95</p>

    <form action="checkout.html" method="POST">
        <fieldset class="cores">
            <legend>Escolha a cor:</legend>

            <input type="radio" name="cor" value="verde" id="verde" checked>
            <label for="verde">
                
            </label>

            <input type="radio" name="cor" value="rosa" id="rosa">
            <label for="rosa">
                
            </label>

            <input type="radio" name="cor" value="azul" id="azul">
            <label for="azul">
                
            </label>
        </fieldset>

        <button class="comprar">Comprar</button>
    </form>
</section>

```

4. Teste o HTML feito e veja seu funcionamento sem interferência do CSS que faremos depois.

7.4 DESIGN MOBILE-FIRST

Quando criamos a home page do projeto não sabíamos ainda otimizar nosso site para dispositivos móveis. Vimos o design e codificamos originalmente pensando nos browsers do desktop. Mais tarde, aplicamos os conceitos de media queries e *viewport* para ajustar o projeto para telas menores.

Esse tipo de fluxo de desenvolvimento é muito comum. Desenvolver para desktop primeiro e depois ajustar o design para mobile. Mas isso **não é o melhor**, nem o mais fácil.

Muita gente argumenta a favor de uma técnica inversa, **mobile-first**. Isso significa fazer o design mobile primeiro, implementar o código para mobile primeiro e, depois, ajustar para o desktop. O resultado final deve ser um site que funciona tanto no desktop quanto no mobile, como antes, só mudamos a ordem do *fluxo de desenvolvimento*.

Na prática, o que muita gente descobriu é que criar pensando no ambiente mais restritivo (o mobile) e depois adaptar para o ambiente mais poderoso (desktop) é mais fácil que fazer o contrário.

Um exemplo prático que passamos na nossa home page. Fizemos antes um menu com CSS usando `hover` para abrir subcategorias. Isso é algo super comum e funciona muito bem no desktop. Mas é um desastre no mobile, onde não existe `hover`. Podemos agora repensar nossa home para ser compatível com mobile. Mas se tivéssemos, desde o início, pensando em mobile, talvez nem criássemos o menu `hover`.

Outro exemplo: os links do menu são bastante inacessíveis em mobile. As opções estão muito próximas uma das outras e tocar na opção certa com o dedo (gordo!) é muito difícil. No desktop não há esse problema pois usamos mouse, por isso não pensamos nisso antes.

Se tivéssemos começado pelo mobile, já teríamos feito os links maiores e mais espaçados pensando nos dedos gordos (costuma-se recomendar um valor médio de 50px para cada item clicável).

E, fazendo tudo maior e mais espaçado, assim como evitando o `hover`, o site funciona bem no mobile mas, não só isso, funciona muito bem no desktop. Um site bem feito para mobile funciona perfeitamente no desktop mas o contrário nem sempre é verdade. Por isso o **mobile-first**.

Repare que o designer já mandou a página de produtos para nós pensando em mobile-first: pouca informação, só o essencial, prioritário. Botões grandes e espaçados. Nenhum efeito de hover. Tudo numa coluna só de informações para dar scroll, já que a tela é pequena.

Nem sabemos ainda como será a versão desktop, e não interessa por enquanto.

MAIS SOBRE MOBILE-FIRST

Não vamos nos estender no assunto aqui no curso mas, se interessar, existe um livro só sobre o tema, chamado *Mobile-first* de Luke Wroblewski. Em português, você pode ler mais no livro *A Web Mobile* do instrutor da Caelum Sérgio Lopes.

7.5 PROGRESSIVE ENHANCEMENT

No exercício vamos ver como usar CSS para estilizar o formulário anterior em algo parecido com o design desejado. Mas o importante é perceber como temos uma página funcional e acessível antes de pensarmos em visual.

E este é o papel do HTML: estruturar o conteúdo da página de maneira semântica e acessível, provendo uma base de funcionalidades para a página sem relação imediata com visual.

O CSS, expressará a estratégia de design da página para apresentar a informação do HTML semântico. Depois, veremos que com o JavaScript é possível adicionar dinâmismo à pagina, porém ele é opcional. Apenas com HTML é possível criar uma experiência funcional da página.

Esse tipo de pensamento é o **progressive enhancement**, ou seja, construir uma base sólida, simples, portável e acessível e depois, progressivamente, incrementar a página com recursos mais avançados tanto de estilo com CSS quanto de comportamento com JavaScript.

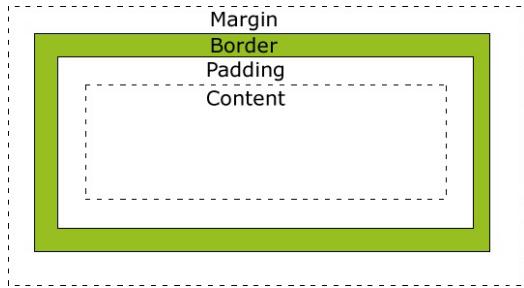
Hoje no mercado, ainda faltam profissionais com experiência e de completo entendimento das implicações de **progressive enhancement**. Por exemplo, quem está começando pode pensar que o mais importante é apenas copiar fielmente o design de um *layout*. Porém, profissionais experientes focam em páginas Web acessíveis, semânticas e portáveis, aliadas à estratégia de design. Já para o usuário final a fusão das duas competências de design e frontend expressam uma completa experiência do usuário.

7.6 BOX MODEL E BOX-SIZING

Box Model padrão do W3C

Quando alteramos as propriedades de elementos dentro de uma página, precisamos estar cientes de como essas alterações se comportarão na presença de outros elementos. Uma forma de entender o impacto causado pela mudança é pensar no **box model** ou **modelo em caixa** em português.

O **box model** é constituído de quatro áreas retangulares: conteúdo (content), espaçamento (padding), bordas (border) e margens (margin) conforme a figura abaixo:



Essas áreas se desenvolvem de dentro para fora, na ordem listada abaixo:

- **conteúdo (content)**: aquilo que será exibido;
- **espaçamento (padding)**: distância entre a borda e o conteúdo;
- **borda (border)**: quatro linhas que envolvem a caixa (box);
- **margem (margin)**: distância que separa um box de outro.

Tendo em mente o `box model`, precisamos ter atenção na alteração de propriedades de um elemento visualizando o impacto em sua apresentação ao lidar com as propriedades listadas acima.

Box-sizing

Os primeiros a perceberem que o `box model` do CSS era esquisito foi a Microsoft. Já no IE6 em *quirks mode* eles trocaram o box model para que o `width` significasse o tamanho total até a borda. A ideia era boa mas o problema foi eles atropelarem a especificação da época, o que acabou criando boa parte dos problemas de incompatibilidade entre navegadores. O IE em *standards mode* usa o `box model` oficial e esse é o padrão a partir do IE8.

Mas como a ideia, no fim, era boa, isso acabou se transformando no `box-sizing` do CSS3, que nos permite trocar o box model que queremos usar.

Por padrão, todos os elementos têm o valor `box-sizing: content-box` o que indica que o tamanho dele é definido pelo seu conteúdo apenas - em outras palavras, é o tal box model padrão que vimos antes. Mas podemos trocar para `box-sizing: border-box` que indica que o tamanho agora levará em conta até a borda – ou seja, o `width` será a soma do conteúdo com a borda e o padding.

7.7 EXERCÍCIOS: PÁGINA DE PRODUTO

Vamos estruturar nosso CSS para implementar a funcionalidade de troca de cores. A cada passo, teste no browser para ir acompanhando o resultado.

1. Crie um novo arquivo `produto.css` na pasta `css/`.
2. Em `produto.html`, importe o `produto.css` após todos os outros CSS's:

```
<link rel="stylesheet" href="css/produto.css">
```

3. Primeiro, vamos desenhar as bolinhas coloridas com pseudo-elementos do CSS3 usando um truque com bordas redondas grandes:

```
.cores label::after {  
    content: "";  
    display: block;  
    border-radius: 50%;  
    width: 50px;  
    height: 50px;  
}  
  
.cores label[for=verde]::after {  
    background-color: #33cc66;  
}  
  
.cores label[for=rosa]::after {  
    background-color: #ff6666;  
}  
  
.cores label[for=azul]::after {  
    background-color: #6666ff;  
}
```

Próximo passo é estilizar a bolinha atualmente selecionada usando pseudo-classe :checked :

```
.cores input:checked + label::after {  
    border: 6px solid rgba(0,0,0,0.3);  
}
```

Repare como a borda da bolinha selecionada aumenta o tamanho total da bolinha por causa do *box model padrão*. Uma solução é trocar o box model com a propriedade box-sizing :

```
.cores label::after {  
    box-sizing: border-box;  
}
```

Agora que temos as bolinhas coloridas visuais configuradas, a bolinha do input radio é desnecessária. Esconda-a:

```
.cores input[type=radio] {  
    display: none;  
}
```

Para fechar a funcionalidade de escolha das cores, falta exibir apenas a imagem atualmente selecionada. Outra forma de falar isso é que *devemos esconder as imagens dos radios não selecionados*. Podemos usar um seletor avançado para isso:

```
.cores input:not(:checked) + label img {  
    display: none;  
}
```

Reflita sobre esse último seletor. Ele é bem complexo. O que ele faz exatamente? Esteja certo de ter entendido cada parte dele antes de prosseguir.

4. Teste a página no navegador. A troca de imagens deve estar funcionando, apesar das coisas não estarem ainda posicionadas corretamente.
5. Com a troca de imagens funcionando, vamos implementar o *posicionamento* correto das bolinhas lado a lado. Para isso, use `position: absolute` já que seus tamanhos são conhecidos:

```
.cores label::after {  
    position: absolute;  
    top: 30px;  
}
```

As bolinhas vão ser posicionadas com relação ao `fieldset` `cores`, então ele precisa estar posicionado. O `padding` superior é para abrir espaço para as bolinhas:

```
.cores {  
    position: relative;  
    padding-top: 90px;  
}
```

As bolinhas ficaram sobrepostas na esquerda. Para corrigir, basta colocar uma coordenada `left` diferente para cada uma:

```
.cores label[for=verde]::after {  
    left: 0;  
}  
  
.cores label[for=rosa]::after {  
    left: 60px;  
}  
  
.cores label[for=azul]::after {  
    left: 120px;  
}
```

Teste o resultado no navegador.

DIFERENÇAS ENTRE CHROME E FIREFOX

Se você testar a página do jeito que ela está agora no Chrome e no Firefox, você verá uma diferença no posicionamento das bolinhas. No Firefox, elas ficarão mais para baixo do que no Chrome.

Isso acontece porque o Firefox dá um tratamento especial para as tags `<fieldset>` e `<legend>`. No caso, o elemento `<legend>` não é considerado pelo Firefox como parte do `<fieldset>` e, assim, o `padding` que colocamos fica muito grande nesse navegador.

Uma forma de resolver esse problema é deixar o `<legend>` posicionado absolutamente:

```
.cores legend {  
    position: absolute;  
    top: 0;  
    left: 0;  
}
```

6. Um ponto importante de uma solução responsiva é que as imagens se adaptem ao tamanho da tela. Às vezes, usamos imagens maiores e, quando a tela é pequena, a imagem fica "vazando" para fora do elemento pai.

Uma forma de corrigir esse problema é garantir que ela nunca passe o tamanho do pai com `max-width`:

```
.cores label img {  
    display: block;  
    max-width: 100%;  
}
```

7. Com toda a parte funcional e de posicionamento pronta, vamos estilizar alguns detalhes visuais da página.

Primeiro, detalhes de tipografia e espaçamento para toda página de produtos:

```
.produto {  
    color: #333;  
    line-height: 1.3;  
    margin-top: 2em;  
}
```

O nome do produto e seu preço também ganham estilos:

```
.produto h2 {  
    font-size: 1.8em;  
    font-weight: bold;  
}  
  
.produto p {
```

```
    font-size: 1.2em;
    font-style: italic;
    margin-bottom: 1em;
}
```

O `<legend>` ganha um destaque:

```
.produto legend {
    display: block;
    font: bold 0.9em/2.5 Arial;
    text-transform: uppercase;
}
```

E por fim, o botão de comprar deve ficar em evidência:

```
.comprar {
    background-color: #91bd3c;
    border: none;
    color: #333;
    font-size: 1.4em;
    text-transform: uppercase;
    box-shadow: 0 1px 3px #777;
    display: block;
    padding: 0.5em 1em;
    margin: 1em 0;
}
```

Teste e observe o estilo simples da página.

8. (opcional) Quando selecionamos a bolinha, uma borda escura aparece. Use `transition` para fazer a borda aparecer suavemente.

```
.cores label::after {
    border: 6px solid rgba(0,0,0,0);
    transition: 1s;
}
```

E podemos colocar também um estilo para quando passar o mouse em cima da bolinha, mostrando uma borda mais leve:

```
.cores label:hover::after {
    border: 6px solid rgba(0,0,0,0.1);
}
```

9. (opcional) Melhore a experiência do nosso formulário com alguns pequenos ajustes. Faça o cursor mudar para mãozinha quando passar o mouse em cima das bolinhas ou em cima do botão:

```
.cores label::after {
    cursor: pointer;
}

.comprar {
    cursor: pointer;
}
```

ALGUNS TIPO DE CURSORES

A propriedade CSS cursor especifica o cursor do mouse mostrado quando o mouse está sobre o elemento, podemos encontrar vários formatos para os cursores na seguinte página:

<https://developer.mozilla.org/pt-BR/docs/Web/CSS/cursor>

7.8 EVOLUINDO O DESIGN PARA DESKTOP

Feito o design mobile-first, é hora de expandir o site para as versões maiores. Do ponto de vista de design, significa ajustar os elementos para melhor aproveitar o espaço maior das telas de tablets e desktops. Do ponto de vista de código, é usar media queries para implementar essas mudanças.

Um exemplo: imagine que, em telas maiores que 600px, queremos flutuar uma imagem a esquerda:

```
@media (min-width: 600px) {  
    img {  
        float: left;  
    }  
}
```

Ao desenvolver mobile-first, usamos muitas media queries do tipo **min-width** para implementar as mudanças para o tablet/desktop.

7.9 MEDIA QUERIES DE CONTEÚDO

Ao escrever medias queries, você precisa escolher algum valor para colocar lá. É o que chamamos de **breakpoints**, os pontos onde seu layout vai ser ajustado por causa de uma resolução diferente. Escrever bons breakpoints é essencial para um design responsivo de qualidade.

E o que mais aparece de pergunta de quem está começando com design responsivo é: quais os valores padrões de se colocar nas media queries? E logo surge uma lista parecida de tamanhos comuns: 320px, 480px, 600px, 768px, 992px, 1200px. O pessoal chama essa prática de device-driven breakpoints, pois são valores gerados a partir de tamanhos de dispositivos.

Mas evite esse tipo de breakpoint. Essa lista pensa em meia dúzia de tipos de dispositivos, mas obviamente não atende todos (e os 360px de um Galaxy S4?). Usar esses valores de media queries não garante que seu design funcionará em todos os dispositivos, mas apenas nos dispositivos "padrões" -- seja lá o que for isso.

Prefira breakpoints com valores baseados no seu conteúdo. Ou seja, achar suas media queries a partir do seu conteúdo e do seu design. Fica bem mais fácil garantir que sua página funciona em todos os dispositivos.

Na prática, faça seu design mobile-first, abra no navegador pequeno, vá redimensionando a janela até achar um ponto que o design quebra ou fica feio; anote o tamanho da janela e crie um breakpoint lá. Não precisa ser um valor bonitinho como 600px. Às vezes sua página vai quebrar justo em 772px. Não tem problema.

7.10 EXERCÍCIOS: RESPONSIVE DESIGN

Nosso layout anterior foi feito com mobile em mente, **mobile-first**. A parte boa é que, quando abrimos no desktop, tudo funciona muito bem. Mas o espaço maior não é bem aproveitado.

Se você redimensionar a janela para um tamanho grande, notará que nosso conteúdo não está centralizado na página como o restante. Lembre-se que criamos uma classe `container` para isso. Podemos usá-la novamente.

1. Na página **produto.html**, crie uma `<div class="container">` ao redor do conteúdo da página.

Isto é, será uma `div` pai da `<section class="produto">`:

```
<!-- envolvendo a section.produto pela div.container -->
<div class="container">
  <section class="produto">
    ...
  </section>
</div>
```

2. Todo o CSS dos exercícios seguintes estará dentro de uma media query que só vai disparar em telas maiores. Edite **produto.css** e adicione a media query no final do arquivo:

```
@media (min-width: 630px) {
  /* adicionar todos os próximos estilos desse exercício */
}
```

Na versão desktop, queremos reposicionar as coisas em duas colunas. Vamos colocar a foto do produtos à esquerda posicionada em relação ao `.produto`. Isso vai afetar o posicionamento das bolinhas então vamos trocar para posicioná-las com `float` simples.

O código é curto mas cheio de detalhes. Acompanhe os comentários para entender o papel de cada item (não precisa copiar os comentários).

Adicione dentro da media query `@media (min-width: 630px)`:

```
.produto {
  /* a foto vai se posicionar absolutamente com relação a esse elemento,
  por isso preciso estar posicionado */
  position: relative;

  /* deixar 40% de espaço em branco na esquerda para foto ocupar */
  padding-left: 40%;
}
```

```

.cores {
    /* estava relative antes; reinicio para static para evitar que a foto se posicione
    com relação a mim */
    position: static;

    /* zerando o padding-top que tinha antes e não preciso mais */
    padding: 0;
}

.cores legend {
    /* estava absolute antes, para ter o mesmo comportamento no Chrome e no Firefox;
    agora podemos voltar ao fluxo da página */
    position: static;
}

.cores label img {
    /* imagem se posiciona absolutamente à esquerda com relação ao .produto */
    position: absolute;
    top: 0;
    left: 0;
}

.cores label::after {
    /* as bolinhas coloridas tinham posição absoluta e não precisamos mais,
    basta flutuar uma do lado da outra */
    position: static;
    float: left;
}

```

Teste a página no navegador e veja que a imagem deixa a desejar, pois ainda não está posicionada corretamente. Apesar disso, o restante já começa a ficar no lugar.

3. Primeiro, para evitar que a imagem vaze para fora do espaço que lhe foi determinado, vamos usar as propriedades `max-width` e `max-height`.

Adicione dentro da media query `@media (min-width: 630px)`. Você pode até escrever apenas as propriedades **dentro dos seletores** existentes na media query:

```

.cores label img {
    max-width: 35%;
    max-height: 100%;
}

```

Podemos aumentar um pouco o tamanho das fontes usadas no produto. Como estamos usando `em` como unidade de medida, basta aumentar a fonte do elemento pai, o `produto` e tudo mais irá escalar proporcionalmente.

Adicione dentro da media query `@media (min-width: 630px)` :

```

.produto {
    font-size: 1.2em;
}

```

Último ajuste é uma pequena margem entre as bolinhas coloridas.

Adicione dentro da media query @media (min-width: 630px) :

```
.cores label::after {  
    margin-right: 10px;  
}
```

Nosso layout final deve ficar como na imagem abaixo:

Nenhum item na sacola de compras

Sua Conta | Lista de Desejos | Cartão Fidelidade | Sobre | Ajuda

Camiseta Ecko Caveira Bad to The Bone
por apenas R\$ 48,95

ESCOLHA A COR:

COMPRAR

7.11 HTML5 INPUT RANGE

Seguindo o design original, precisamos criar uma maneira de selecionar também o tamanho da roupa, além de sua cor. O tamanho é algo simples em nossa loja. Temos como valores possíveis: 36, 38, 40, 42, 44 e 46.

E há muitas formas corretas e semânticas de implementar isso no formulário. Pode ser um `<select>` com esses valores, radio buttons ou, como vamos ver, o **novo input range do HTML5**.

O `<input type="range">` é um componente novo do HTML5, com bom suporte já nos navegadores, que representa um slider numérico. Ele recebe atributos **min** e **max** com o intervalo numérico possível. Opcionalmente, há o atributo **step** que indica de quanto em quanto o número deve pular (algo bem útil para tamanho de roupa, que só tem números pares). Existe também o atributo **value** que indica o valor inicial do `input`.

```
<input type="range" min="36" max="46" value="42" step="2" name="tamanho" id="tamanho">
```

O legal de componentes HTML5 é que eles são nativos dos browsers. Isso significa que não precisamos de trabalho para usá-los ou estilizá-los. Eles já vêm com estilo padrão do navegador em questão o que é bem interessante. A interface padrão é familiar para o usuário.

Veja o range no Safari do iPhone:



E veja o mesmo componente no IE10 do Windows 8:



Visuais totalmente diferentes mas totalmente adaptados à plataforma em questão.

O problema do input range é que não há feedback visual de qual valor está selecionado. Precisamos criar um outro elemento visual na página apenas para mostrar o valor atualmente selecionado no range.

Mas que tag usar pra representar esse elemento cujo valor é resultado do valor escolhido no range?

No HTML5, temos uma tag nova com valor semântico exato pra essa situação: o **<output>**. Essa tag representa a saída de algum cálculo ou valor simples obtido a partir de um ou mais campos de um formulário. Ele tem um atributo **for** que aponta de qual elemento saiu o seu valor.

```
<output for="tamanho" name="valortamanho">42</output>
```

O valor em si está como 42 porque colocamos na mão, dentro da tag. O que precisamos é atualizar esse valor toda vez que o valor do input range mudar, ou seja, toda vez que o usuário arrastar o input range. E para isso acontecer precisamos utilizar o JavaScript, veremos isso mais para frente.

SUPORTE AO INPUT RANGE

Todos os browsers modernos suportam o input range. Você terá problemas porém em versões mais antigas. O IE suporta a partir do 10, o Android a partir do 4.2, e o Firefox no 23.

<http://caniuse.com/input-range>

Lembre que aqui no curso estamos estudando novas ideias. Se você precisar suportar os navegadores抗igos, sempre poderá substituir por um **<select>** simples ou um conjunto de radio buttons. Funcionalmente, terão o mesmo resultado.

7.12 EXERCÍCIOS: SELETOR DE TAMANHO

1. Implemente a funcionalidade de escolher o tamanho da roupa usando um input range e o output do HTML5, colocando o código a seguir logo abaixo do fechamento do nosso primeiro **<fieldset>**

no arquivo **produto.html**:

```
<fieldset class="tamanhos">
    <legend>Escolha o tamanho:</legend>

    <input type="range" min="36" max="46" value="42" step="2" name="tamanho" id="tamanho">
    <output for="tamanho" name="valortamanho">42</output>
</fieldset>
```

Teste seu funcionamento nos browsers modernos. Repare que o elemento `output` não atualiza seu valor sozinho, vamos implementar isso via JavaScript.

2. Temos dois `fieldsets`, um para escolher cor e outro, tamanho. No mobile, eles ficam um em cima do outro. No desktop, podemos posicioná-los lado a lado.

Adicione dentro da media query `@media (min-width: 630px)` :

```
.produto fieldset {
    display: inline-block;
    vertical-align: top;
    margin: 1em 0;
    min-width: 240px;
    width: 45%;
}
```

3. Estilize o `output` para ter um design mais ajustado a nossa página de produto. **Adicione** no arquivo **produto.css**, **fora** das *media queries* existentes.

```
.tamanhos output {
    display: inline-block;
    height: 44px;
    width: 44px;
    line-height: 44px;
    text-align: center;
    border: 3px solid #666;
    border-radius: 50%;
    color: #555;
}
```

7.13 TABELAS

O uso de tabelas era muito comum há alguns anos para a definição de áreas. Seu uso para essa finalidade acabou se tornando prejudicial pela complexidade da marcação, o que dificulta bastante a manutenção das páginas. Além disso havia uma implicação direta na definição de relevância do conteúdo das tabelas para os indexadores de conteúdo por mecanismos de busca.

Ainda assim, hoje, quando queremos exibir uma série de dados tabulares, é indicado o uso da tag de **tabela** `<table>`.

```
<table>
    <tr>
        <th>Título da primeira coluna</th>
        <th>Título da segunda coluna</th>
    </tr>
```

```

<tr>
  <td>Linha 1, coluna 1</td>
  <td>Linha 1, coluna 2</td>
</tr>
<tr>
  <td>Linha 2, coluna 1</td>
  <td>Linha 2, coluna 2</td>
</tr>
</table>

```

Note que na primeira linha `<tr>` da tabela, as células são indicadas com a tag `<th>`, o que é útil para diferenciar seu conteúdo das células de dados.

Existem diversas maneiras de se alterar uma estrutura de uma tabela, como por exemplo indicamos que uma célula `<td>` ou `<th>` ocupa mais de uma linha de altura por meio do atributo `rowspan`, ou então que ela ocupa mais de uma coluna de largura com o uso do atributo `colspan`.

Podemos adicionar um título à nossa tabela com a tag `<caption>`.

Ainda existem as tags `<thead>`, `<tfoot>` e `<tbody>`, que servem para agrupar linhas de nossa tabela. Vale ressaltar que dentro do grupo `<thead>` devemos ter apenas linhas contendo a tag `<th>` como célula.

Outra tag de agrupamento que temos na tabela é a que permite que sejam definidas as colunas, é a tag `<colgroup>`. Dentro dessa tag definimos uma tag `<col>` para cada coluna, e dessa maneira podemos adicionar alguns atributos que influenciarão todas as células daquela coluna.

A seguir um exemplo completo de como utilizar essas tags dentro de uma tabela.

```

<table>
  <caption>Quantidade e preço de camisetas.</caption>

  <colgroup>
    <col width="10%">
    <col width="40%">
    <col width="30%">
    <col width="20%">
  </colgroup>

  <thead>
    <tr>
      <th rowspan="2">
        <th colspan="2" rowspan="2">Quantidade de Camisetas</th>
        <th rowspan="2">Preço</th>
      </th>
    </tr>
    <tr>
      <th>Amarela</th>
      <th>Vermelha</th>
    </tr>
  </thead>

  <tfoot>
    <tr>
      <td>
        <td>Total de camisetas amarelas: 35</td>
      </td>
    </tr>
  </tfoot>

```

```

<td>Total de camisetas vermelhas: 34</td>
<td>Valor total: $45.00</td>
</tr>
</tfoot>

<tbody>
  <tr>
    <td>Polo</td>
    <td>12</td>
    <td>5</td>
    <td>$30.00</td>
  </tr>
  <tr>
    <td>Regata</td>
    <td>23</td>
    <td>29</td>
    <td>$15.00</td>
  </tr>
</tbody>
</table>

```

7.14 EXERCÍCIOS: DETALHES

- Crie a seção de detalhes logo abaixo da section com a classe produto , mas ainda **dentro** da div com a classe container :

```

<div class="container">
  <section class="produto">
    <!-- formulário com a escolha de cor e tamanho -->
  </section>

  <section class="detalhes">
    <h2>Detalhes do produto</h2>
    <p>Camiseta confortável ideal para eventos casuais!</p>
    <p>Camiseta manga curta, gola redonda. Possui estampa de Caveira e descriptivo
       frontal e lisa nas costas.</p>
  </section>
</div>

```

- O estilo é bastante simples, apenas para deixar o texto mais bonito. Adicione **fora** da media query que fizemos antes:

```

.detalhes {
  padding: 2em 0;
}

.detalhes h2 {
  font-size: 1.5em;
  line-height: 2;
}

.detalhes p {
  margin: 1em 0;
  font-size: 1em;
  line-height: 1.5;
}

```

```

@media (min-width: 500px) {
    .detalhes {
        font-size: 1.2em;
    }
}

```

3. Crie uma tabela com características do produto contendo informações técnicas. **Adicione** dentro da `section detalhes`:

```





```

4. Estilize a tabela para deixá-la mais agradável. Use o seletor de filhos múltiplos para um estilo zebra.

Adicione o estilo **fora** das *media query*:

```

.detalhes table {
    border-spacing: 0.2em;
    border-collapse: separate;
}

.detalhes thead {
    background-color: #999;
}

.detalhes thead th {
    font-weight: bold;
    padding: 0.3em 1em;
    text-align: center;
}

.detalhes td {
    padding: 0.3em;
}

```

```
.detalhes tr:nth-child(2n) {  
    background-color: #ccc;  
}  
  
.detalhes td:first-child {  
    font-style: italic;  
}
```

7.15 EXERCÍCIOS OPCIONAIS: FUNDO

1. Para implementarmos o fundo cinza em tela cheia, vamos precisar de um novo elemento pai para conter todos os elementos da página. Crie um `<div class="produto-back">` ao redor da `<div class="container">` que tínhamos antes.

Apenas para referência, nesse momento, seu HTML deve estar mais ou menos assim:

```
<div class="produto-back">  
    <div class="container">  
        <section class="produto">  
  
            <!-- formulário com a escolha de cor e tamanho -->  
  
        </section>  
        <section class="detalhes">  
  
            <!-- tabela com detalhes do produto -->  
  
        </section>  
    </div>  
</div>
```

2. O estilo é bastante simples, apenas usando uma cor e bordas sutis para criar um efeito mais elegante:

Adicione o estilo **fora** das *media query*:

```
.produto-back {  
    background-color: #f2f2f2;  
    margin-top: 1em;  
    border-top: 2px solid #ccc;  
}  
  
.cores label img {  
    border: 2px solid #ccc;  
}
```

BOOTSTRAP E FORMULÁRIOS HTML5

"O trabalho é a melhor das regularidades e a pior das intermitências" -- Victor Marie Hugo

8.1 BOOTSTRAP E FRAMEWORKS DE CSS

Uma tendência em alta no mundo front-end é o uso de frameworks CSS com estilos base para nossa página. Ao invés de começar todo projeto do zero, criando todo estilo na mão, existem frameworks que já trazem toda uma base construída de onde partiremos nossa aplicação.

Existem muitas opções mas o **Twitter Bootstrap** talvez seja o de maior notoriedade. Ele foi criado pelo pessoal do Twitter a partir de códigos que eles já usavam internamente. Foi liberado como opensource e ganhou muitos adeptos. O projeto cresceu bastante em maturidade e importância no mercado a ponto de se desvincular do Twitter e ser apenas o **Bootstrap**.

<http://getbootstrap.com>

O Bootstrap traz uma série de recursos:

- Reset CSS
- Estilo visual base pra maioria das tags
- Ícones
- Grids prontos pra uso
- Componentes CSS
- Plugins JavaScript
- Tudo responsivo e mobile-first

Como o próprio nome diz, é uma forma de começar o projeto logo com um design e recursos base sem perder tempo com design no início.

8.2 ESTILO E COMPONENTES BASE

Para usar o Bootstrap, apenas incluímos seu CSS na página:

```
<link rel="stylesheet" href="css/bootstrap.css">
```

Só isso já nos traz uma série de benefícios. Um reset é aplicado, e nossas tags ganham estilo e

tipografia base. Isso quer dizer que podemos usar tags como um H1 ou um P agora e elas terão um estilo característico do Bootstrap.

Além disso, ganhamos **muitas classes** com componentes adicionais que podemos aplicar na página. São várias opções. Por exemplo, pra criar um título com uma frase de abertura em destaque, usamos o Jumbotron:

```
<div class="jumbotron jumbotron-fluid">
  <div class="container">
    <h1 class="display-4">Ótima escolha!</h1>
    <p class="lead">Obrigado por comprar na Mirror Fashion.</p>
  </div>
</div>
```

No exercício a seguir vamos usar vários outros componentes.

8.3 A PÁGINA DE CHECKOUT DA MIRROR FASHION

Neste capítulo, vamos desenvolver a página de **checkout** da Mirror Fashion. Após escolher o produto desejado, o usuário cai nessa página para efetivar a compra.

Nossa loja foi otimizada pra compra direta, sem carrinho de compras. O cliente escolhe o produto e compra direto, com um clique. Só precisamos coletar os dados dele e do pagamento.

O foco dessa nova página é a coleta de informações para efetivação da compra. Um grande formulário complexo com os campos necessários. Vamos usar o Bootstrap para desenvolver essa página com mais facilidade e rapidez.

The screenshot shows a desktop view of a checkout page. At the top, a large gray header contains the text "Ótima escolha!" and "Obrigado por comprar na Mirror Fashion! Preencha seus dados para efetivar a compra.". Below this, the main content area is divided into sections. On the left, a sidebar titled "Sua compra" displays a thumbnail of a green t-shirt with a skull graphic, along with details: "Camiseta Ecko Caveira", "Cor: Verde", "Tamanho: 42", and "Preço: R\$ 48,95". To the right, there are two main groups of fields. The first group, "Dados pessoais", includes "Nome completo" (with a red border around the input field), "Email" (with a red border around the input field containing "email@example.com"), and "CPF" (with a red border around the input field containing "000.000.000-00"). The second group, "Cartão de crédito", includes "Número - CVV" (input field), "Bandeira" (dropdown menu with "Selecionar uma opção..."), "Validade" (input field), and a "Confirmar Pedido" button with a thumbs-up icon. A checkbox labeled "Quero receber Newsletter da Mirror Fashion" is also present in this section.

Figura 8.1: Site visto no Desktop

E, como aprendemos antes, vamos desenvolver tudo **mobile-first**. Nesse momento, portanto, ainda não teremos o design Desktop mostrado acima, mas uma versão mobile em uma coluna. Veremos como adaptar a versão Desktop com Bootstrap depois.

The screenshot shows a mobile-optimized checkout interface for 'Mirror Fashion'. On the left, a sidebar displays the purchase summary:

- Sua compra**
- Produto**: Camiseta Ecko Caveira
- Cor**: Verde
- Tamanho**: 42
- Preço**: R\$ 48,95

The main form area contains the following fields:

- Email**: email@example.com
- CPF**: 000.000.000-00
- Quero receber Newsletter da Mirror Fashion** (checkbox checked)
- Cartão de crédito** section with fields for **Número - CVV**, **Bandeira** (dropdown menu), and **Validade**.
- Dados pessoais** section with a **Nome completo** field.
- Confirmar Pedido** button.

Figura 8.2: Site visto no Mobile

8.4 EXERCÍCIOS: PÁGINA DE CHECKOUT

- Crie a página **checkout.html** com HTML simples.

```
<!doctype html>
<html>
  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width">
    <title>Checkout Mirror Fashion</title>
  </head>
  <body>

    <h1>Ótima escolha!</h1>
    <p>Obrigado por comprar na Mirror Fashion!
      Preencha seus dados para efetivar a compra.</p>

    <h2>Sua compra</h2>
    <dl>
      <dt>Produto</dt>
      <dd>Camiseta Ecko Caveira</dd>

      <dt>Cor</dt>
      <dd>Verde</dd>

      <dt>Tamanho</dt>
```

```

<dd>42</dd>

<dt>Preço</dt>
<dd>R$ 48,95</dd>
</dl>

</body>
</html>

```

2. Abra a página **checkout.html** no navegador e veja que está com o estilo padrão do navegador.

O primeiro passo é incluirmos o arquivo CSS do bootstrap na nossa página. Você vai ver uma mudança sutil no estilo da página, principalmente em aspectos tipográficos.

Coloque no `<head>` da página de checkout o CSS do bootstrap:

```
<link rel="stylesheet" href="css/bootstrap.css">
```

Teste novamente a página.

3. O primeiro componente pronto do bootstrap que vamos usar é o **jumbotron**. É basicamente a abertura do site, contendo sua chamada principal. Para usá-lo basta criar uma `div` com a classe `jumbotron` e a classe `jumbotron-fluid`.

Envolva as chamadas de abertura que **já tínhamos** com `h1` e `p` em duas `<div>`. A primeiro `div` contém `class="jumbotron jumbotron-fluid"` e a segundo, `class="container"`.

```

<div class="jumbotron jumbotron-fluid">
  <div class="container">

    <!-- para o h1 e p que já tínhamos, somente adicionar as respectivas classes -->
    <h1 class="display-4">Ótima escolha!</h1>
    <p class="lead">Obrigado por comprar na Mirror Fashion!
      Preencha seus dados para efetivar a compra.</p>

  </div><!-- fim .container dentro do jumbotron -->
</div><!-- fim .jumbotron -->

```

Abra a página e note que um estilo diferente aparece. Teste redimensionar o navegador e veja que o tamanho da fonte e espaçamento do componente se ajustam automaticamente. O Bootstrap usa responsive design automaticamente em seus componentes.

Para saber mais do jumbotron: <https://getbootstrap.com/docs/4.1/components/jumbotron/>

4. Use um outro componente do Bootstrap, o **card** para organizar a seção que mostramos as informações da compra do cliente. Cuidado com o exercício, com os nomes das classes, que podem confundir bastante.

Adapte o HTML do `h2 "Sua compra"` e do `dl` que temos para se adequar ao componente `card`, adicionando o seguinte código **após** o fechamento da `<div class="jumbotron">`:

```
<div class="card mb-3">
```

```

<div class="card-header">
  <h2>Sua compra</h2>
</div><!-- fim .card-header -->

<div class="card-body">
  <!-- ... aqui vai o <dl> que já temos ... -->
</div><!-- fim .card-body -->
</div><!-- fim .card -->

```

Repare como os nomes das classes, apesar de serem muitos, fazem sentido para isolar cada parte do cartão.

Teste novamente a página no navegador e veja o resultado. Temos um cartão arredondado com título em destaque no topo.

Para saber mais sobre cartões do Bootstrap: <https://getbootstrap.com/docs/4.1/components/card/>

5. Repare no exercício anterior do jumbotron que o `<div class="container">` é responsável por centralizar e dar espaçamento na tela. Muito parecido aliás com o container que havíamos criado antes em nosso projeto, mas agora é uma classe do Bootstrap.

Crie uma outra div container pra conter o `card` que acabamos de criar e veja como ele fica melhor posicionado no centro da tela.

```

<div class="container">
  <!-- ... card aqui ... -->
</div><!-- fim .container da pagina -->

```



6. Dentro do `card-body`, logo no topo, acima da lista de definições `<dl>`, vamos colocar uma foto do produto na cor escolhida.

Com Bootstrap, podemos ainda acrescentar algumas classes nessa imagem para obter resultados interessantes. A classe `img-thumbnail` faz a imagem ficar flexível e nunca estourar o tamanho do pai, também faz a imagem ficar centralizada e com uma borda de destaque. A classe `mb-3` adiciona uma pequena margem na parte inferior da imagem.

Adicione a imagem do produto logo acima da lista `<dl>` dentro do div `card-body` :

```

```

Teste novamente a página.

8.5 FORMULÁRIOS A FUNDO

Quando solicitamos que o usuário informe seu nome, seu endereço de email, se ele quer receber uma newsletter, qualquer informação, precisamos utilizar os elementos corretos. Para isso, vamos conhecer os formulários HTML: a tag `<form>`.

Já usamos alguns antes. Agora vamos ver a fundo seus desdobramentos.

Atributos do Form

```
<form action="/efetivar.html" method="POST">
</form>
```

O formulário exemplificado anteriormente apresenta o atributo obrigatório **action**. O valor desse atributo é o endereço para onde as informações do formulário serão enviadas, e esse valor depende inteiramente de como é feita a aplicação que receberá essas informações no lado do servidor.

O segundo atributo, `method`, especifica o método do HTTP pelo qual essa informação será transmitida. O valor `post`, de maneira simplista, significa que queremos **inserir** as informações desse formulário, salvá-la de alguma maneira. Outro valor possível para esse atributo, o `get`, é utilizado quando queremos obter alguma coisa a partir das informações que estamos transmitindo, por exemplo, um formulário de busca.

Componentes

Porém, neste exemplo, não temos nenhum elemento para capturar as informações. Na verdade, somente a marcação da tag `<form>` não mostra nenhum elemento visível no navegador. Vamos supor que precisemos de uma informação como o nome do visitante do nosso site para guardar em um banco de dados. Vamos adicionar alguns elementos ao nosso formulário anterior:

```
<form action="/efetivar.html" method="POST">  
  <label for="nome">Nome:</label>  
  <input type="text" name="nome" id="nome">  
  <input type="submit">  
</form>
```

Label

Adicionamos a marcação do elemento `<label>`. Esse elemento é uma tag de conteúdo, e seu texto é exibido de maneira comum dentro do nosso formulário, a única diferença é que essa marcação faz uma ligação com outro elemento qualquer em nosso formulário. Note que nosso `label` tem o atributo `for`, que recebe o valor `nome`.

Quando clicamos com o mouse sobre o texto marcado com a tag `label`, o elemento que tem o atributo `id` com o mesmo valor que o atributo `for` do `label` é selecionado para que possamos interagir com ele. No exemplo, esse elemento vinculado ao `label` é um campo de texto que declaramos com a tag `input`.

Essa marcação `<label>` é de extrema importância para a usabilidade e acessibilidade dos nossos formulários.

Input

A maioria dos elementos que utilizamos nos formulários para capturar informações dos usuários são da tag `<input>`. No exemplo anterior, utilizamos duas variações dessa tag.

Os tipos diferentes de `inputs` são determinados pelo valor do seu atributo `type`. A seguir, vamos detalhar os tipos aceitos para essa tag.

text

```
<input type="text" name="nome_usuario">
```

Provavelmente o tipo mais comum de input, o que tem o atributo `type="text"`, é utilizado quando queremos que o usuário envie uma informação textual simples, pois esse elemento não permite a entrada de quebras de linha.

Ao enviarmos o formulário, a informação digitada pelo usuário é acessível no lado do servidor por meio do atributo `name`, utilizado para identificar cada informação contida nos parâmetros da requisição. Para ter acesso à informação digitada quando tratamos o formulário com algum tipo de script, para validar o conteúdo por exemplo, é necessário obter o conteúdo da propriedade `value` do objeto no DOM.

password

O input que recebe o atributo **type="password"** é similar ao anterior, do tipo **text**, com a diferença de que ele não exibe exatamente o texto digitado pelo usuário, e sim uma série de símbolos * ou outro, dependendo do navegador e sistema operacional.

```
<input type="password" name="senha">
```

checkbox

O elemento **input** do tipo **checkbox** exibe uma caixa para marcação, é muito utilizado quando temos uma opção que pode ser marcada como sim ou não, por exemplo "Aceito os termos de contrato do usuário", ou "Manter a sessão ativa" em formulários de login.

Apesar de muito utilizado com o valor **true**, é possível determinar qualquer valor para o checkbox.

```
<input id="contrato" name="contrato" type="checkbox" value="sim">
<label for="contrato">Aceito os termos do contrato.</label>
```

radio

```
<input type="radio" name="idade" id="idade5" value="5">
<label for="idade5">Menos de 5 anos</label>

<input type="radio" name="idade" id="idade10" value="10">
<label for="idade10">Menos de 10 anos</label>

<input type="radio" name="idade" id="idade15" value="15">
<label for="idade15">Menos de 15 anos</label>

<input type="radio" name="idade" id="idade20" value="20">
<label for="idade20">Menos de 20 anos</label>
```

Quando desejamos que o usuário escolha somente uma entre uma série de opções, podemos utilizar elementos **input** do tipo **radio**. Quando há mais de um elemento desse tipo com o mesmo valor no atributo **name**, somente um pode ser selecionado.

image

```
<input type="image" name="botao" src="images/enviar.png"
       alt="Botão para enviar o formulário" width="20" height="18">
```

É possível substituir o botão de envio do formulário por uma imagem, possibilitando criar um visual mais atrativo para o formulário.

Hoje em dia, com o CSS3, podemos adicionar imagens que não fazem parte do conteúdo, são somente estilo, com **Image Replacement** no CSS, como fizemos na página **index.html**.

file

```
<input type="file" name="anexo">
```

Quando é necessário que o usuário envie um arquivo para a aplicação no lado do servidor é necessário o uso do **input** do tipo **file**. Para o correto envio dos arquivos, muitas vezes também é

necessário adicionar o atributo `enctype="multipart/form-data"` na tag `<form>`.

hidden

```
<input type="hidden" name="codigo" value="abc012xyz789">
```

Muitas vezes precisamos enviar e receber informações que não têm utilidade direta para o usuário e, portanto, não devem ser exibidos no formulário. Para essa finalidade, existe o `input` do tipo **hidden**, que somente carrega um valor.

button

```
<input type="button" name="mostra_dialogo" value="Clique aqui!">
```

O elemento `input` com o atributo `type="button"` renderiza um botão dentro do formulário, mas esse botão não tem nenhuma função direta nele e é comumente utilizado para disparar eventos para a execução de scripts.

O texto do botão é determinado pelo valor do atributo `value`.

submit

```
<input type="submit" name="enviar" value="Enviar">
```

O elemento `input` com o atributo `type="submit"` é similar ao `type="button"`, mas quando acionado esse elemento inicia a chamada que envia as informações do formulário para o endereço indicado no atributo `action` do `<form>`.

reset

```
<input type="reset" name="reset" value="Limpar">
```

O `input` com `type="reset"` elimina os valores digitados anteriormente nos elementos de um formulário, permitindo que o usuário limpe o mesmo.

```
<INPUT> E <BUTTON>
```

A tag `<input>` dos tipos **button**, **submit** e **reset** pode ser substituída pela tag `<button>`. Neste caso, o texto do botão passa a ser indicado como conteúdo da tag. Ainda assim é necessário especificar o valor do atributo `type`, inclusive se ele for **button**:

```
<button type="button" name="mostra_dialogo">Clique aqui!</button>
<button type="submit" name="enviar">Enviar</button>
<button type="reset" name="reset">Limpar</button>
```

Textarea

Quando desejamos que o usuário insira uma quantidade grande de informações textuais, incluindo quebras de linha, é necessário o uso da tag **textarea**

```
<textarea name="texto"></textarea>
```

Select, Optgroup e Option

Quando desejamos que o usuário selecione entre diversas opções, com a possibilidade de flexibilizar a maneira com que ele interage com o componente do formulário, podemos utilizar a tag **<select>**.

```
<select name="cidades">
  <option value="bsb">Brasilia</option>
  <option value="rj">Rio de Janeiro</option>
  <option value="sp">São Paulo</option>
</select>
```

Em sua configuração padrão, o controle **select** exibe o que conhecemos como **menu drop-down**, permitindo que somente uma das opções possa ser selecionada. Caso seja adicionado o atributo **multiple**, é possível selecionar mais de uma opção da mesma maneira que selecionamos diversos arquivos no explorador do sistema operacional.

```
<select name="cidades" multiple>
  <option value="bsb">Brasilia</option>
  <option value="rj">Rio de Janeiro</option>
  <option value="sp">São Paulo</option>
</select>
```

Caso necessário, dependendo do número de opções apresentadas ao usuário, pode ser interessante agrupá-las:

```
<select name="bairro">
  <optgroup label="Brasília">
    <option value="asan_bsb">Asa Norte</option>
    <option value="asas_bsb">Asa Sul</option>
  </optgroup>
  <optgroup label="Rio de Janeiro">
    <option value="botafogo_rj">Botafogo</option>
    <option value="centro_rj">Centro</option>
  </optgroup>
  <optgroup label="São Paulo">
    <option value="vlmariana_sp">Vila Mariana</option>
    <option value="centro_sp">Centro</option>
  </optgroup>
</select>
```

8.6 NOVOS COMPONENTES DO HTML5

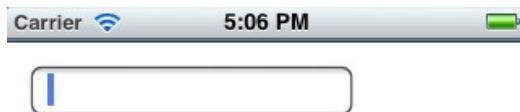
Com a nova especificação do HTML, é possível utilizar uma série de novos componentes que facilitam bastante o desenvolvimento de formulários. Até o momento em que essa apostila foi escrita, muitos componentes são incompatíveis com os navegadores, mas mostram, na maioria dos casos, um campo de texto permitindo a entrada de qualquer tipo de informação.

A maioria dos novos tipos de componentes de formulário foram criados para permitir que o navegador adapte o método de entrada para o mais adequado em cada um dos casos. Alguns desses componentes já são compatíveis com navegadores de dispositivos móveis.

email

```
<input type="email" name="email">
```

O input do tipo **email** permite que os dispositivos móveis, principalmente, exibam um teclado adaptado para facilitar esse tipo de entrada. Por exemplo, o iPhone exibe um teclado com o caractere @ e com as opções de domínio .com .



```
<input type="email">
```



number

```
<input type="number" max="100" step="5">
```

O input do tipo **number**, além de exibir um teclado numérico em dispositivos móveis, nos navegadores modernos exibe um controle que permite incrementar ou decrementar o valor do campo clicando em uma seta para cima ou para baixo.

Além dessa diferença visual, é possível determinar valores mínimos, máximos e se há uma escala de valores válidos. No exemplo anterior, o elemento deve aceitar números múltiplos de 5 com o limite do valor "100".

url

```
<input type="url" name="endereco">
```

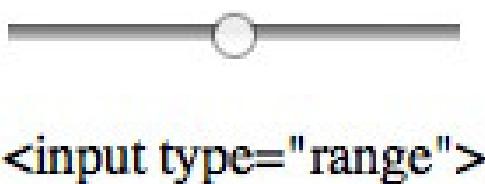
O elemento `input` com tipo **url** permite que os dispositivos exibam um teclado como, no exemplo do iPhone, opções como `www` e `.com`.

range

```
<input type="range" name="volume">
```

O elemento `input` do tipo **range** exibe um controle deslizante nos navegadores modernos, permitindo uma interação mais agradável quando precisamos de um valor numérico em escala. O controle guarda um valor numérico em seu atributo `value`. Assim como o `input` do tipo `number`, é possível especificar um valor mínimo, máximo e uma escala.

A renderização mais comum desse controle, em um Chrome:

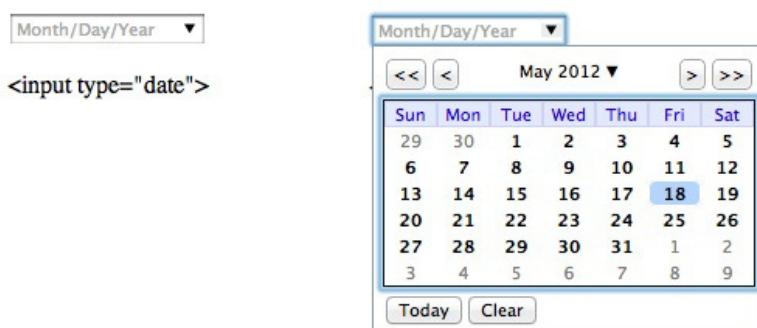


date, month, week, time, datetime e datetime-local

```
<input type="date" name="validade">
```

Os controles de "date picker" são feitos para coletar uma informação de data ou hora. São várias as possibilidades de formato de data ou hora. No navegador Opera, quando utilizado esse tipo de controle, o usuário pode selecionar uma data a partir de um calendário. É possível especificar datas mínima e máxima.

Em geral, os navegadores devem oferecer alguma funcionalidade de escolha de datas para o usuário, como no Chrome:



Ou no iPhone:

Carrier 5:07 PM

May 18, 1973 ▾

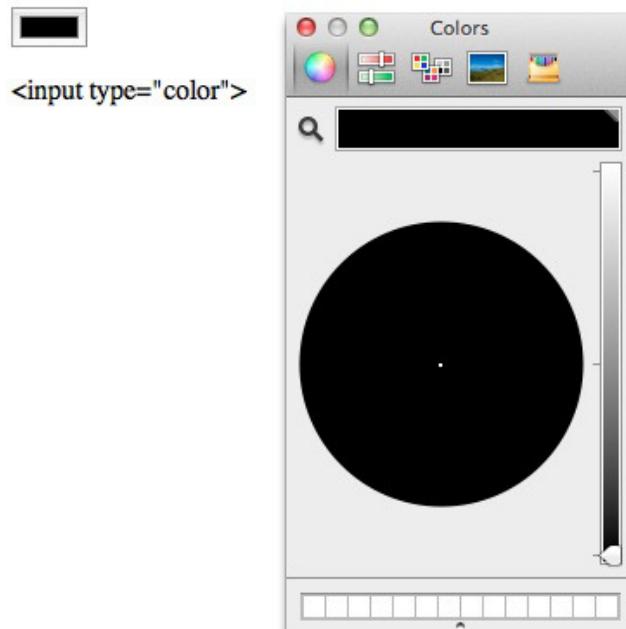
<input type="date">



color

<input type="color" name="cor_olhos">

O elemento `input` do tipo **color** permite que seja exibido um "color picker" para o preenchimento do seu valor. O Chrome no Mac, por exemplo, exibe o color picker padrão do sistema:



search

```
<input type="search" results="10">
```

O input do tipo **search** exibe um campo específico para busca. O atributo "results" determina quantas últimas buscas serão armazenadas e lembradas, além de exibir uma lupa dentro do campo (Safari e Chrome).

tel

```
<input type="tel" name="telefone">
```

O input do tipo **tel** foi especificado para coletar um número de telefone.

Em dispositivos com teclados virtuais como smartphones e tablets, é comum o teclado ser adaptado para exibir apenas opções relevantes à entrada de números telefônicos, como no iPhone:



```
<input type="tel">
```



8.7 NOVOS ATRIBUTOS HTML5 EM ELEMENTOS DE FORMULÁRIO

Na especificação do HTML5 estão definidos novos atributos para os elementos de formulário, visando implementar algumas necessidades comuns que antes não eram possíveis de serem atendidas puramente com a marcação do formulário.

autofocus

Sua presença indica que aquele campo deve iniciar com foco quando a página for carregada. O

usuário já pode começar a digitar algo sem nenhum clique.

```
<input type="text" name="nome" autofocus>  
placeholder  
<input type="text" name="nome" placeholder="Insira seu nome">
```

O atributo **placeholder** exibe o texto contido em seu valor dentro do elemento do formulário caso o seu valor seja vazio.

autocomplete, list e datalist

É possível implementar uma funcionalidade de sugestão de valores com mais facilidade.

```
<input type="text" list="cidades" autocomplete="on">  
<datalist id="cidades">  
  <option value="Brasília">  
  <option value="Rio de Janeiro">  
  <option value="São Paulo">  
</datalist>
```

A implementação de **autocomplete** sem o atributo **list** no campo, ligando-o a um **datalist**, vai utilizar os últimos valores utilizados em outros campos ou em outros formulários, dando prioridade a valores adicionados em **inputs** com o mesmo valor no atributo **name**.

Existem diversas maneiras de utilizar os componentes de formulários, tanto os novos do HTML5 como os já existentes. Mesmo com a oportunidade de inovar e criar uma interação totalmente diferente do usuário com um formulário, é importante manter o mesmo método que adotamos anteriormente. A marcação correta do formulário facilita muito o uso dele em diversos navegadores e em outros tipos de clientes também, como por exemplo navegadores especiais para deficientes visuais.

8.8 ÍCONES

O Bootstrap não inclui uma biblioteca de ícones por padrão, mas eles recomendam algumas bibliotecas para escolher. As recomendadas são:

- **Font Awesome**
- **Iconic**
- **Octicons**

Para saber mais detalhes: <https://getbootstrap.com/docs/4.1/extend/icons/>

Aqui vamos usar o **Iconic**, que é um código aberto com 223 ícones nos formatos SVG e fonte customizada.

Para usarmos essa biblioteca é necessário baixar um arquivo **.css** e adicionar no HTML:

```
<link rel="stylesheet" href="css/open-iconic-bootstrap.css">
```

Depois disso é muito simples utilizar os ícones:

```
<span class="oi oi-thumb-up"></span>
```

Os ícones são disponibilizados através de uma **fonte de texto customizada**. A vantagem de se usar fontes para ícones é que o desenho fica escalável, como uma letra. Ele não perde qualidade em nenhum tamanho ou resolução por ser vetorial. E, assim como uma letra, podemos aplicar efeitos de texto como sombras e cores.

A desvantagem é que cada ícone só pode ter um path no desenho e uma única cor. Não é possível usar ícones complexos com fontes.

8.9 EXERCÍCIOS: FORMULÁRIOS

1. O formulário de compra possui campos para o cliente digitar informações pessoais e informações sobre o pagamento. Para melhor organização, vamos separar os campos em dois **fieldsets**.

Vamos criar o `<form>` logo depois do `card`, e ainda **dentro** do container. Neste `form`, crie o primeiro `<fieldset>` usando `<legend>` para identificar os Dados Pessoais.

Esse primeiro **fieldset**, dos Dados Pessoais, deve conter os campos Nome, Email, CPF e um checkbox para o usuário optar ou não por receber newsletter.

```
<form>
  <fieldset>
    <legend>Dados pessoais</legend>

    <div class="form-group">
      <label for="nome">Nome completo</label>
      <input type="text" class="form-control" id="nome" name="nome">
    </div>

    <div class="form-group">
      <label for="email">Email</label>
      <input type="email" class="form-control" id="email" name="email">
    </div>

    <div class="form-group">
      <label for="cpf">CPF</label>
      <input type="text" class="form-control" id="cpf" name="cpf">
    </div>

    <div class="form-group custom-control custom-checkbox">
      <input type="checkbox" class="custom-control-input" id="newsletter"
             value="sim" checked>
      <label class="custom-control-label" for="newsletter">
        Quero receber Newsletter da Mirror Fashion
      </label>
    </div>
  </fieldset>

  <!-- No próximo exercício, vamos acrescentar outro fieldset -->

</form>
```

Repare que cada campo possui um input e um label. Para agrupá-los, usamos uma div `form-group` do Bootstrap. Cada input deve ter uma classe `form-control`, atenção que para o `input` com `type="checkbox"` é necessário adicionar outras classes do Bootstrap.

Teste a página no navegador e observe o estilo padrão que ganhamos apenas por usar o Bootstrap.

2. Adicione mais um `<fieldset>` ainda dentro do **form** (logo após a tag de fechamento do `<fieldset>` de Dados pessoais) com os Dados do Cartão. Esse **fieldset** tem três campos: um com código do cartão, outro com a bandeira do cartão e outro com data de validade. Neste último, usaremos o *input month* do HTML5.

```
<fieldset>
    <legend>Cartão de crédito</legend>

    <div class="form-group">
        <label for="numero-cartao">Número - CVV</label>
        <input type="text" class="form-control" id="numero-cartao" name="numero-cartao">
    </div>

    <div class="form-group">
        <label for="bandeira-cartao">Bandeira</label>
        <select class="custom-select" id="bandeira-cartao">
            <option disabled selected>Selecione uma opção...</option>
            <option value="master">MasterCard</option>
            <option value="visa">VISA</option>
            <option value="amex">American Express</option>
        </select>
    </div>

    <div class="form-group">
        <label for="validade-cartao">Validade</label>
        <input type="month" class="form-control" id="validade-cartao" name="validade-cartao">
    </div>
</fieldset>
```

Teste novamente no navegador.

3. Para finalizar, é necessário adicionar um botão antes do fechamento da tag **form**, que cuidará do envio dos dados digitados (vamos usar um `btn-primary` do Bootstrap).

```
<button type="submit" class="btn btn-primary mb-3">
    Confirmar Pedido
</button>
```

4. **Adicione o atributo placeholder** do HTML5 nos campos *email* e *CPF* com dicas de preenchimento:

```
<input type="email" class="form-control" id="email" name="email" placeholder="email@ exemplo.com">
...
<input type="text" class="form-control" id="cpf" name="cpf" placeholder="000.000.000-00">
```

Adicione o atributo autofocus do HTML5 no input *nome*:

```
<input type="text" class="form-control" id="nome" name="nome" autofocus>
```

5. Vamos incentivar o clique no botão de confirmação do pedido com um ícone além do texto. Use os **Iconic** pra isso. Primeiro adicione o link para essa biblioteca dentro da tag `<head>`.

```
<link rel="stylesheet" href="css/open-iconic-bootstrap.css">
```

Em seguida, dentro do botão, apenas **adicione a tag span** com a declaração do ícone:

```
<button type="submit" class="btn btn-primary mb-3">
    <span class="oi oi-thumb-up"></span>
    Confirmar Pedido
</button>
```

Para saber mais sobre os ícones: <https://useiconic.com/open/>

6. Use outras classes do Bootstrap para ajustar mais detalhes. No botão, **adicione** a classe `btn-lg` para deixar o botão maior.

Veja mais opções de botões com Bootstrap: <https://getbootstrap.com/docs/4.1/components/buttons/>

7. (opcional) O Bootstrap tem outros recursos para formulários, como os **input groups**. Teste **trocando** o código do campo **email** para isso:

```
<div class="form-group">
    <label for="email">Email</label>

    <div class="input-group mb-3">
        <div class="input-group-prepend">
            <span class="input-group-text">@</span>
        </div>
        <input type="email" class="form-control" id="email" name="email"
               placeholder="email@example.com">
    </div>
</div>
```

Veja outro exemplo do uso do **input groups**. **Troque** o código do **campo Bandeira do cartão de crédito** para isso:

```
<div class="form-group">
    <div class="input-group mb-3">
        <div class="input-group-prepend">
            <label class="input-group-text" for="bandeira-cartao">Bandeira</label>
        </div>
        <select class="custom-select" id="bandeira-cartao">
            <option disabled selected>Selecione uma opção...</option>
            <option value="master">MasterCard</option>
            <option value="visa">VISA</option>
            <option value="amex">American Express</option>
        </select>
    </div>
</div>
```

Implemente também em outros campos, inclusive usando ícones.

Veja mais opções do Bootstrap para formulários:

<https://getbootstrap.com/docs/4.1/components/forms/>

8.10 VALIDAÇÃO HTML5

Entre as muitas novidades de formulários que vimos no HTML5, há ainda toda uma parte de validação de dados com restrições expressas diretamente no código HTML.

required

Podemos indicar na marcação do formulário quando um campo é de preenchimento obrigatório.

```
<input type="text" name="nome" required>
```

Esse atributo permite uma validação *fraca* no lado do cliente.

pattern

Conseguimos também especificar um formato requerido através do atributo **pattern**, adicionando uma expressão regular como valor:

```
<input type="text" pattern="^@\w{2,}" name="usuario_twitter">
```

O atributo **pattern** também permite uma validação *fraca* do campo.

Validação no CSS

A maioria dos novos componentes de formulário e os atributos que funcionam como validadores de campos na verdade somente aplicam uma pseudo-classe específica no campo que não está atendendo ao padrão ou requisito especificado.

Essa pseudo-classe é a **:invalid**, e pode ser utilizada para dar um retorno visual imediato caso o usuário não esteja atendendo aos requisitos dos campos do formulário.

```
:invalid {  
    outline: 1px solid #cc0000;  
}
```

Essa validação é *fraca* pois de maneira direta não é possível impedir que o usuário envie as informações do formulário, mesmo que incompletas ou incorretas. É possível porém alterar o botão de `submit` e deixá-lo desabilitado caso seja possível selecionar algum elemento por essa pseudo-classe no formulário. Essa verificação e alteração do elemento `submit` pode ser feita por JavaScript e jQuery de maneira simples.

SUporte nos Navegadores

A validação HTML5 está implementada no Chrome, Firefox, Safari, Opera e IE10. Dos navegadores móveis, temos suporte em Chrome, Firefox, Opera, IE e Blackberry:

<http://caniuse.com/form-validation>

Se você quiser suportar navegadores mais antigos, recomendamos o uso de um polyfill:

<https://github.com/aFarkas/webshim>

8.11 EXERCÍCIOS: VALIDAÇÃO COM HTML5

1. **Adicione** o atributo **required** nos campos *Nome* e *CPF* para que os campos fiquem com obrigatório o preenchimento.

Teste submeter o formulário sem preencher esses campos.

2. Algumas validações já são implícitas apenas por usarmos o input type correto. Por exemplo, tente submeter o formulário preenchendo o email com um valor inválido (com dois @ por exemplo).
3. Para os campos com inputs que não possuem essa validação implícita, podemos estilizar no CSS quando esse campo estiver inválido. Para isso, vamos criar o arquivo **checkout.css** na pasta **css** e adicionar o estilo abaixo:

```
.form-control:invalid {  
    border: 1px solid #cc0000;  
}
```

Para que esse estilo funcione na página **checkout.html** precisamos adicionar o link dele **depois** de todos os outros links:

```
<link rel="stylesheet" href="css/checkout.css">
```

8.12 GRID RESPONSIVO DO BOOTSTRAP

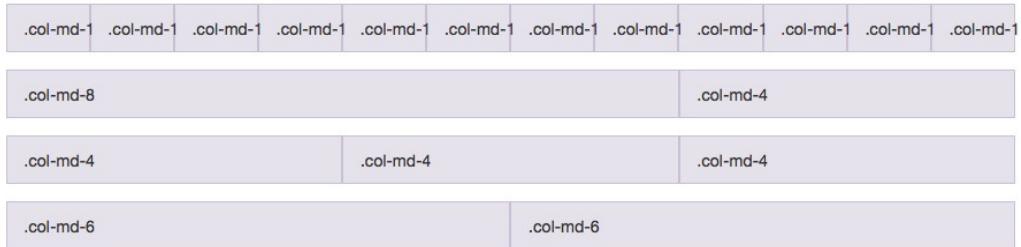
Umas das dificuldades mais comuns de um projeto front-end é o posicionamento de elementos, sobretudo em designs multi coluna. A solução mais comum é uso de grids, uma ideia antiga que veio dos próprios designers.

Divide-se a tela em colunas e vamos encaixando os elementos dentro desse grid.

Todo framework CSS moderno traz um grid pronto para utilização. Todo código CSS necessário para correto posicionamento já foi escrito e só precisamos **usar as classes certas**. O Bootstrap tem um

grid pronto e várias classes para usarmos.

O grid do Bootstrap trabalha com a ideia de **12 colunas** e podemos escrever nosso código escolhendo quantas colunas ocupar. Alguns exemplos:



Essas classes de coluna são as que definem o tamanho de cada elemento na página com base nas 12 partes do grid padrão. Em código:

```
<div class="row">
  <div class="col-md-4">
    ...
  </div>

  <div class="col-md-8">
    ...
  </div>
</div>
```

No código anterior, deixamos a primeira `<div>` ocupando 4/12 da tela e a outra, 8/12. Repare que, para o grid funcionar, ao redor das colunas usamos uma `<div>` com a classe **row**. Ela é necessária para que o layout fique correto.

Podemos ainda criar grids dentro de outro grid, sempre obedecendo a divisão de 12 colunas em cada. Por exemplo:

```
<div class="row">
  <div class="col-md-4">
    ...
  </div>

  <div class="col-md-8">
    <div class="row">
      <div class="col-md-6">
        ...
      </div>
      <div class="col-md-6">
        ...
      </div>
    </div>
  </div>
</div>
```

Esse exemplo criou um segundo grid dentro da coluna da direita do primeiro grid. Nesse segundo grid há duas colunas ocupando metade cada uma (6/12). Mas como um grid está dentro do outro, na

prática, ele vai ocupar metade do tamanho da `<div>` que tem 8/12 de tamanho.

Responsivo

Um dos pontos mais interessantes dos grids é que eles são **responsivos**. Isso quer dizer que podemos aplicar diferentes layouts de colunas no nosso código ao mesmo tempo e cada um deles vai valer só em determinada situação.

Nos códigos anteriores, por exemplo, usamos classes como **col-md-6**. O **md** nessa classe significa que vamos ocupar 6 colunas do grid *apenas em telas maiores que 768px de largura*. Em telas menores, automaticamente nosso grid será de uma coluna só.

E, claro, temos classes pra outros tamanhos de tela também. No Bootstrap temos essas famílias de classes de grids já prontas:

- **col-** : Extra small < 576px
- **col-sm-** : Small >= 576px
- **col-md-** : Medium >= 768px
- **col-lg-** : Large >= 992px
- **col-xl-** : Extra large >= 1200px

Podemos aplicar mais de uma classe ao mesmo tempo no mesmo elemento:

```
<div class="row">
  <div class="col-xs-6 col-sm-4">
    ...
  </div>

  <div class="col-xs-6 col-sm-8">
    ...
  </div>
</div>
```

Nesse exemplo, nosso grid divide no meio (6 pra cada lado) em telas muito pequenas mas depois divide em 4 e 8 pra telas um pouco maiores.

8.13 EXERCÍCIOS: GRIDS

1. Nossa design mobile-first funciona muito bem em telas pequenas. Mas conforme vamos aumentando o browser, notamos que tudo fica meio grande. O `card` e o `form` esticam 100%, o que é um exagero em telas maiores.

Vamos usar grids do Bootstrap para transformar nosso design em 2 colunas em telas maiores. Por padrão, o Bootstrap já traz media queries para adaptação em 768px. A ideia é deixar o `card` ocupar **4/12** e o `form` ocupar **8/12**.

São três alterações necessárias:

- Criar uma `<div>` com classe `row` dentro do container envolvendo o `card` e o `form` ;
- Criar uma `<div>` com classe `col-md-4` ao redor do `card` ;
- Aplicar a classe `col-md-8` no formulário.

Cuidado a `div` do Jumbotron fica fora dessas alterações.

Faça essas alterações e **cuidado** com o resultado final e os milhões de divs misturados. O código deve ficar mais ou menos assim:

```
<div class="container">
  <div class="row">

    <div class="col-md-4">
      <div class="card mb-3">
        <!-- ... cartão todo aqui ... -->
      </div><!-- fim .card .mb-3 -->
    </div><!-- fim .col-md-4 -->

    <form class="col-md-8">
      <!-- ... todos os campos aqui ... -->
    </form><!-- fim .col-md-8 -->

  </div><!-- fim .row -->
</div><!-- fim .container -->
```

Teste a página e redimensione para um tamanho em torno de 768px pra ver o resultado.

2. Repare que o Bootstrap ajusta várias coisas responsivamente pra gente de maneira automática. Além de aplicar as classes do grid, repare como os tamanhos e fontes aumentam de acordo com a resolução, sem precisarmos fazer nada.

Faça os testes.

3. Quando aumentamos bastante a tela, tudo ainda se ajusta na proporção de 4 pra 8 que definimos. Mas o formulário fica grande demais. Em telas maiores, talvez seja legal deixar o formulário em 2 colunas.

Vamos usar outras classes do grid do Bootstrap que se aplicam em layouts maiores que 992px. Vamos dividir o formulário em 2 partes iguais, ou seja 6/12 (lembre que o grid do Bootstrap tem 12 partes como base). Conseguimos isso tudo usando a classe **col-lg-6**.

As mudanças necessárias são:

- Crie uma `<div>` com classe **row** ao redor dos 2 fieldsets e do button;
- Aplique a classe **col-lg-6** no primeirofieldset;
- Em volta do segundofieldset e do button, crie uma `div` com a classe **col-lg-6**.

No final, a estrutura deve estar parecida com essa:

```
<form ...>
  <div class="row">
    <fieldset class="col-lg-6">
      ...
    </fieldset>
    <div class="col-lg-6">
      <fieldset>
        ...
      </fieldset>
      <button ...>
    </div><!-- fim .col-lg-6 -->
  </div><!-- fim .row -->
</form>
```

Teste a página e redimensione para um tamanho em torno de 992px pra ver o resultado.

4. (opcional) É possível usar mais de uma classe de grid ao mesmo tempo no mesmo elemento. Por exemplo: dividimos a tela em 4/12 para o cartão e 8/12 para o formulário. Mas se, em telas maiores, você quiser mudar essa proporção para 3/12 e 9/12, basta adicionar as classes **col-xl-3** e **col-xl-9** em conjunto as que tínhamos antes.

Implemente essa mudança no projeto.

Exemplo:

```
<div class="col-md-4 col-xl-3">
  ...
<form class="col-md-8 col-xl-9">
```

A série **col-xl-** aplica em resoluções acima de 1200px.

Para saber mais sobre os grids do Bootstrap: <https://getbootstrap.com/docs/4.1/layout/grid/>

5. (opcional) Além de alterar o grid nas diferentes resoluções, o Bootstrap também permite esconder/exibir certos elementos apenas em uma resolução específica.

Por exemplo: imagine que, para otimizar o espaço pequeno no design para smartphone, vamos esconder a imagem do produto. Podemos fazer isso **adicionando** as classes **d-none** e **d-sm-block** na ``.

A classe **d-none** esconde o elemento independente da resolução da tela. E a classe **d-sm-block** faz com que o elemento apareça em telas com resoluções maiores que 576px.

Para saber mais sobre as classes auxiliares para responsivo do Bootstrap:
<https://getbootstrap.com/docs/4.1/extend/approach/#responsive>

8.14 PARA SABER MAIS: COMPONENTES JS DO BOOTSTRAP

Além de componentes CSS puro do Bootstrap como card e jumbotron, temos outros componentes

mais avançados que envolvem interatividade e JavaScript.

Há muita coisa disponível por padrão no Bootstrap, pelo menos os componentes mais comuns como janela modal, galeria de imagens, dropdowns, menus de navegação e mais.

<https://getbootstrap.com/docs/4.1/getting-started/javascript/>

No exercício, vamos usar o menu superior (navbar).

8.15 EXERCÍCIOS OPCIONAIS: NAVBAR E JAVASCRIPT

1. Um componente muito famoso do Bootstrap é seu menu superior, chamado de **navbar**. O HTML é um pouco mais complexo pois se trata de um menu completo, mas é relativamente fácil.

Implemente um **navbar** em nossa página acima do jumbotron, logo no topo da página:

```
<nav class="navbar navbar-expand-lg navbar-light bg-light">
  <a class="navbar-brand" href="index.html">Mirror Fashion</a>

  <ul class="navbar-nav">
    <li class="nav-item active">
      <a class="nav-link" href="sobre.html">Sobre</a>
    </li>
    <li class="nav-item">
      <a class="nav-link" href="#">Ajuda</a>
    </li>
    <li class="nav-item">
      <a class="nav-link" href="#">Perguntas frequentes</a>
    </li>
    <li class="nav-item">
      <a class="nav-link" href="#">Entre em contato</a>
    </li>
  </ul>
</nav>
```

Teste o resultado no navegador.

2. Teste o menu em resoluções menores. Note que o Bootstrap ajusta automaticamente o navbar em telas menores. Por padrão, o comportamento é mudar o menu de horizontal para vertical em mobile.

Veja esse comportamento redimensionando o browser.

3. Uma outra solução para menus em telas pequenas é de juntar as opções em uma espécie de dropdown que só abre quando ativado. Isso é, criar um botão para ativar o menu (geralmente com o famoso ícone do sanduíche).

É bem simples fazer isso com Bootstrap, a funcionalidade está toda pronta.

Para fazer o menu colapsar em telas pequenas, basta **adicionar** uma `<div>` nova em volta do `` e 2 classes nessa `<div>`: a **collapse** e a **navbar-collapse**.

```
<div class="collapse navbar-collapse">
```

```

<ul class="navbar-nav">
    ...
</ul>
</div>

```

Se você testar agora, vai notar que o menu some nas telas menores. Para exibi-lo, precisamos fazer o próximo passo: criar o ícone que ativa o menu.

Dentro do `navbar`, logo acima do `<a>`, crie um botão de ativação. Dentro do botão crie um `` com a classe **navbar-toggler-icon**. Esse `` será o nosso ícone hambúrguer.

Para fazer a ligação entre o botão que vamos usar para collapsar a nossa `<div>` e a botão em si, use um id na `<div>`. No caso usamos um id **navbarToggleExternalContent** na `<div>` e um `data-target="#navbarToggleExternalContent"` no button, para indicar que esse button aponta para o id **navbarToggleExternalContent**.

```

<nav class="navbar navbar-expand-lg navbar-light bg-light">
    <button class="navbar-toggler" type="button"
        data-toggle="collapse" data-target="#navbarToggleExternalContent">
        <span class="navbar-toggler-icon"></span>
    </button>
    <a class="navbar-brand" href="index.html">Mirror Fashion</a>

    <div class="collapse navbar-collapse" id="navbarToggleExternalContent">
        <ul class="navbar-nav">
            <li class="nav-item active">
                <a class="nav-link" href="sobre.html">Sobre</a>
            </li>
            <li class="nav-item">
                <a class="nav-link" href="#">Ajuda</a>
            </li>
            <li class="nav-item">
                <a class="nav-link" href="#">Perguntas frequentes</a>
            </li>
            <li class="nav-item">
                <a class="nav-link" href="#">Entre em contato</a>
            </li>
        </ul>
    </div>
</nav>

```

Se testar agora, vai notar que o menu aparece mas não funciona quando clicado. É porque essa funcionalidade no Bootstrap é **implementada com JavaScript**. A boa notícia é que não precisamos escrever uma linha de código JS sequer, mas para tudo funcionar precisamos **adicionar o JavaScript do Bootstrap**.

No fim da página, logo antes de fechar o `</body>`, chame o arquivo do Bootstrap e do jQuery:

```

<script type="text/javascript" src="js/jquery.js"></script>
<script type="text/javascript" src="js/bootstrap.js"></script>

```

Teste novamente e veja o plugin funcionando. Usamos o JavaScript do Bootstrap implicitamente.

ATRIBUTOS CUSTOMIZADOS NO HTML5

Até a versão 4 do HTML, não havia uma forma padronizada de colocar atributos customizados.

A partir do HTML5, atributos começando com `data-` em qualquer tag são considerados atributos customizados e não quebram a validade do nosso código HTML. Esses atributos são bastante úteis para passar informação para um código JavaScript, como fizemos agora, passando informação para o código do Bootstrap.

4. Há muitas opções possíveis para o navbar. Por exemplo, podemos inverter as cores e usar um esquema mais escuro apenas trocando as classes `navbar-light` e `bg-light` pelas classes `navbar-dark` e `bg-dark`.

Para saber mais sobre o navbar: <https://getbootstrap.com/docs/4.1/components/navbar/>

5. Adicione a classe **fixed-top** dentro da tag `<nav>`. Repare que o menu fica fixo no topo mesmo com scroll.

Você talvez queira aplicar um `padding-top` no body pro conteúdo não ficar por baixo do navbar, **adicione** no arquivo **checkout.css**:

```
body {  
    padding-top: 55px;  
}
```

8.16 PARA SABER MAIS: OUTROS FRAMEWORKS CSS

O Bootstrap não é o único framework CSS do mercado. É talvez o mais famoso e com mais usuários, mas há muitas outras opções que às vezes podem ser até melhores para seu caso.

Três opções famosas:

- **Foundation:** Da Zurb, fortemente baseado em mobile e responsivo. <http://foundation.zurb.com/>
- **Semantic UI:** Tem nomes de classes mais simples e semânticos que os outros. <http://semantic-ui.com/>
- **Pure:** Do Yahoo, outra alternativa, mais recente. <http://purecss.io/>

De maneira geral, esses frameworks permitem fazer as mesmas coisas, mas cada um com seu estilo. Um botão principal por exemplo:

```
<!-- Bootstrap -->
```

```
<button class="btn btn-primary btn-lg">Clique aqui</button>

<!-- Foundation -->
<button class="large button">Clique aqui</button>

<!-- Semantic UI -->
<button class="large ui button">Clique aqui</button>

<!-- Pure -->
<button class="pure-button pure-button-primary pure-button-large">
    Clique aqui
</button>
```

JAVASCRIPT E INTERATIVIDADE NA WEB

"Design não é só como uma coisa aparenta, é como ela funciona." -- Steve Jobs

9.1 PORQUE USAMOS JAVASCRIPT?

Na página de produto criamos um input range para selecionar o tamanho da roupa. O problema é que não há feedback visual de qual valor está selecionado. Então criamos um outro elemento visual na página apenas para mostrar o valor atualmente selecionado no range.

No HTML5, temos uma tag nova com valor semântico exato pra essa situação: o **<output>**. Essa tag representa a saída de algum cálculo ou valor simples obtido a partir de um ou mais campos de um formulário. Ele tem um atributo **for** que aponta de qual elemento saiu o seu valor.

```
<input type="range" min="36" max="46" value="42" step="2" name="tamanho" id="tamanho">
<output for="tamanho" name="valortamanho">42</output>
```

O valor em si está como 42 porque colocamos na mão, dentro da tag. O que precisamos é atualizar esse valor toda vez que o valor do input range mudar, ou seja, toda vez que o usuário arrastar o input range.

O HTML vem pronto para o navegador com todo seu conteúdo e tags. Mudar o conteúdo de uma tag baseado numa ação do usuário (dentro do navegador) não é função do HTML. Pra isso, precisamos do **JavaScript**.

9.2 UM POUQUINHO DA HISTÓRIA DO JAVASCRIPT

No início da *Internet* as páginas eram pouco ou nada interativas, eram documentos que apresentavam seu conteúdo exatamente como foram criados para serem exibidos no navegador. Existiam algumas tecnologias para a geração de páginas no lado do servidor, mas havia limitações no que diz respeito a como o usuário consumia aquele conteúdo. Navegar através de *links* e enviar informações através de formulários era basicamente tudo o que se podia fazer.

Nasce o JavaScript

Visando o potencial da *Internet* para o público geral e a necessidade de haver uma interação maior do usuário com as páginas, a Netscape, criadora do navegador mais popular do início dos anos 90, de mesmo nome, criou o Livescript, uma linguagem simples que permitia a execução de *scripts* contidos nas páginas dentro do próprio navegador.

Aproveitando o iminente sucesso do Java, que vinha conquistando cada vez mais espaço no mercado de desenvolvimento de aplicações corporativas, a Netscape logo rebatizou o Livescript como JavaScript num acordo com a Sun para alavancar o uso das duas. A então vice-líder dos navegadores, Microsoft, adicionou ao Internet Explorer o suporte a *scripts* escritos em VBScript e criou sua própria versão de JavaScript, o JScript.

JavaScript é a linguagem de programação mais popular no desenvolvimento Web. Suportada por todos os navegadores, a linguagem é responsável por praticamente qualquer tipo de dinamismo que queiramos em nossas páginas.

Se usarmos todo o poder que ela tem para oferecer, podemos chegar a resultados impressionantes. Excelentes exemplos disso são aplicações Web complexas como Gmail, Google Maps e Google Docs.

9.3 CARACTERÍSTICAS DA LINGUAGEM

O JavaScript, como o próprio nome sugere, é uma linguagem de *scripting*. Uma linguagem de *scripting* é comumente definida como uma linguagem de programação que permite ao programador controlar uma ou mais aplicações de terceiros. No caso do JavaScript, podemos controlar alguns comportamentos dos navegadores através de trechos de código que são enviados na página HTML.

Outra característica comum nas linguagens de *scripting* é que normalmente elas são linguagens **interpretadas**, ou seja, não dependem de compilação para serem executadas. Essa característica é presente no JavaScript: o código é interpretado e executado conforme é lido pelo navegador, linha a linha, assim como o HTML.

O JavaScript também possui **grande tolerância a erros**, uma vez que conversões automáticas são realizadas durante operações. Como será visto no decorrer das explicações, nem sempre essas conversões resultam em algo esperado, o que pode ser fonte de muitos bugs, caso não conheçamos bem esse mecanismo.

O script do programador é enviado com o HTML para o navegador, mas como o navegador saberá diferenciar o script de um código html? Para que essa diferenciação seja possível, é necessário envolver o script dentro da tag `<script>`.

9.4 CONSOLE DO NAVEGADOR

Existem várias formas de executar códigos JavaScript em um página. Uma delas é executar códigos

no que chamamos de **Console**. A maioria dos navegadores desktop já vem com essa ferramenta instalada. No Chrome, é possível chegar ao Console apertando **F12** e em seguida acessar a aba "Console" ou por meio do atalho de teclado **Control + Shift + C**; no Firefox, pelo atalho **Control + Shift + K**.

DEVELOPER TOOLS

O console faz parte de uma série de ferramentas embutidas nos navegadores especificamente para nós que estamos desenvolvendo um site. Essa série de ferramentas é o que chamamos de Developer Tools.

Na próxima seção executaremos alguns códigos no console, para aprender um pouco mais do JavaScript.



9.5 SINTAXE BÁSICA

Operadores

Podemos somar, subtrair, multiplicar e dividir como em qualquer linguagem:

Teste algumas contas digitando diretamente no console:

```
> 12 + 13  
25  
> 14 * 3  
42  
> 10 - 4  
6  
> 25 / 5  
5  
> 23 % 2  
1
```

Variáveis

Para armazenarmos um valor para uso posterior, podemos criar uma **variável**:

```
> var resultado = 102 / 17;  
undefined
```

No exemplo acima, guardamos o resultado de `102 / 17` na variável `resultado`. O resultado de criar uma variável é sempre **`undefined`**. Para obter o valor que guardamos nela ou mudar o seu valor, podemos fazer o seguinte:

```
> resultado  
6  
  
> resultado = resultado + 10  
16  
  
> resultado  
16
```

Também podemos alterar o valor de uma variável usando as operações básicas com uma sintaxe bem compacta:

```
> var idade = 10; // undefined  
> idade += 10; // idade vale 20  
> idade -= 5; // idade vale 15  
> idade /= 3; // idade vale 5  
> idade *= 10; // idade vale 50
```

Tipos de dados

Não são apenas números que podemos salvar numa variável. O JavaScript tem vários tipos de dados.

Number

Com esse tipo de dados é possível executar todas as operações que vimos anteriormente:

```
var pi = 3.14159;  
var raio = 20;  
var perimetro = 2 * pi * raio
```

String

Uma string em JavaScript é utilizada para armazenar trechos de texto:

```
var empresa = "Caelum";
```

Para exibirmos o valor da variável `empresa` fora do console, podemos executar o seguinte comando:

```
alert(empresa);
```

O comando `alert` serve para criação de **popups** com algum **conteúdo de texto** que colocarmos dentro dos parênteses. O que acontece com o seguinte código?

```
var numero = 30;  
alert(numero)
```

O número 30 é exibido sem problemas dentro do **popup**. O que acontece é que qualquer variável pode ser usada no `alert`. O JavaScript não irá diferenciar o tipo de dados que está armazenado numa variável, e se necessário, tentará converter o dado para o tipo desejado.

Automatic semicolon insertion (ASI)

É possível omitir o ponto e vírgula no final de cada declaração. A omissão de ponto e vírgula funciona no JavaScript devido ao mecanismo chamado *automatic semicolon insertion* (ASI).

9.6 A TAG SCRIPT

O console nos permite testar códigos diretamente no navegador. Porém, não podemos pedir aos usuários do site que sempre abram o console, copiem um código e colem para ele ser executado.

Para inserirmos um código JavaScript em uma página, é necessário utilizar a tag `<script>`:

```
<script>
  alert("Olá, Mundo!");
</script>
```

A tag `<script>` pode ser declarada dentro da tag `<head>` assim como na tag `<body>`, mas devemos ficar atentos, porque o código é lido imediatamente dentro do navegador. Veja a consequência disso nos dois exemplos abaixo:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Aula de JS</title>
    <script>
      alert("Olá, Mundo!");
    </script>
  </head>
  <body>
    <h1>JavaScript</h1>
    <h2>Linguagem de programação</h2>
  </body>
</html>
```

Repare que, ao ser executado, o script trava o processamento da página. Imagine um script que demore um pouco mais para ser executado ou que exija alguma interação do usuário como uma confirmação. Não seria interessante carregar a página toda primeiro antes de sua execução por uma questão de performance e experiência para o usuário?

Para fazer isso, basta removermos o script do `<head>`, colocando-o no final do `<body>`:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Aula de JS</title>
  </head>
  <body>
    <h1>JavaScript</h1>
    <h2>Linguagem de Programação</h2>
    <script>
      alert("Olá, Mundo!");
    </script>
```

```
</body>  
</html>
```

Devemos sempre colocar o script antes de fechamos a tag `</body>`? Na maioria esmagadora das vezes sim.

JavaScript em arquivo externo

Se o mesmo script for utilizado em outra página, como fazemos? Imagine ter que reescrever o script toda vez que ele for necessário. Para não acontecer isso, é possível importar scripts dentro da página utilizando também a tag `<script>`:

No arquivo HTML

```
<script type="text/javascript" src="js/hello.js"></script>  
  
Arquivo externo js/hello.js  
  
alert("Olá, Mundo!");
```

Com a separação do script em arquivo externo é possível reaproveitar alguma funcionalidade em mais de uma página.

Mensagens secretas no console

É comum querermos dar uma olhada no valor de alguma variável ou resultado de alguma operação durante a execução do código. Nesses casos, poderíamos usar um `alert`. Porém, se esse conteúdo deveria somente ser mostrado para o desenvolvedor, o console do navegador pode ser utilizado no lugar do alert para imprimir essa mensagem:

```
var mensagem = "Olá mundo";  
console.log(mensagem);
```

IMPRESSÃO DE VARIÁVEIS DIRETAMENTE DO CONSOLE

Quando você estiver com o console aberto, não é necessário chamar `console.log(nomeDaVariavel)`: você pode chamar o nome da variável diretamente que ela será impressa no console.

9.7 DOM: SUA PÁGINA NO MUNDO JAVASCRIPT

Para permitir alterações na página, ao carregar o HTML da página, os navegadores carregam em memória uma estrutura de dados que representa cada uma das nossas tags no JavaScript. Essa estrutura é chamada de **DOM (Document Object Model)**. Essa estrutura pode ser acessada através da variável

```
global document .
```

O termo "documento" é frequentemente utilizado em referências à nossa página. No mundo front-end, documento e página são sinônimos.

querySelector

Antes de sair alterando nossa página, precisamos em primeiro lugar acessar no JavaScript o elemento que queremos alterar. Como exemplo, vamos alterar o conteúdo de um título da página. Para acessar ele:

```
document.querySelector("h1")
```

Esse comando usa os **seletores CSS** para encontrar os elementos na página. Usamos um seletor de nome de tag mas poderíamos ter usado outros:

```
document.querySelector(".class")
document.querySelector("#id")
```

Elemento da página como variável

Se você vai utilizar várias vezes um mesmo elemento da página, é possível salvar o resultado de qualquer `querySelector` numa variável:

```
var titulo = document.querySelector("h1")
```

Executando no console, você vai perceber que o elemento correspondente é selecionado. Podemos então manipular seu conteúdo. Você pode ver o conteúdo textual dele com:

```
titulo.textContent
```

Essa propriedade, inclusive, pode receber valores e ser alterada:

```
titulo.textContent = "Novo título"
```

querySelectorAll

As vezes você precisa selecionar vários elementos na página. Várias tags com a classe `.cartao` por exemplo. Se o retorno esperado é mais de um elemento, usamos `querySelectorAll` que devolve uma lista de elementos (*array*).

```
document.querySelectorAll(".cartao")
```

Podemos então acessar elementos nessa lista através da posição dele (começando em zero) e usando o colchetes:

```
// primeiro cartão
document.querySelectorAll(".cartao")[0]
```

Alterações no DOM

Ao alterarmos os elementos da página, o navegador sincroniza as mudanças e alteram a aplicação em tempo real.

9.8 FUNÇÕES E OS EVENTOS DO DOM

Apesar de ser interessante a possibilidade de alterar o documento todo por meio do JavaScript, é muito comum que as alterações sejam feitas quando o usuário executa alguma ação, como por exemplo, mudar o conteúdo de um botão ao clicar nele e não quando a página carrega. Porém, por padrão, qualquer código colocado no `<script>`, como fizemos anteriormente, é executado assim que o navegador lê ele.

Para guardarmos um código para ser executado em algum outro momento, por exemplo, quando o usuário clicar num botão, é necessário utilizar alguns recursos do JavaScript no navegador. Primeiro vamos criar uma **função**:

```
function mostraAlerta() {  
    alert("Funciona!");  
}
```

Ao criarmos uma função, simplesmente guardamos o que estiver dentro da função, e esse código guardado só será executado quando **chamarmos** a função, como no seguinte exemplo:

```
function mostraAlerta() {  
    alert("Funciona!");  
}  
  
// fazendo uma chamada para a função mostraAlerta, que será executada nesse momento  
mostraAlerta()
```

Para chamar a função **mostraAlerta** só precisamos utilizar o nome da função e logo depois abrir e fechar parênteses.

Agora, para que essa nossa função seja chamada quando o usuário clicar no botão da nossa página, precisamos do seguinte código:

```
function mostraAlerta() {  
    alert("Funciona!");  
}  
  
// obtendo um elemento através de um seletor de ID  
var botao = document.querySelector("#botaoEnviar");  
  
botao.onclick = mostraAlerta;
```

Note que primeiramente foi necessário selecionar o botão e depois definir no `onclick` que o que vai ser executado é a função `mostraAlerta`. Essa receita será sempre a mesma para qualquer código que tenha que ser executado após alguma ação do usuário em algum elemento. O que mudará sempre é qual elemento você está selecionando, a qual evento você está reagindo e qual função será executada.

Quais eventos existem?

Existem diversos eventos que podem ser utilizados em diversos elementos para que a interação do usuário dispare alguma função:

- oninput: quando um elemento input tem seu valor modificado
- onclick: quando ocorre um click com o mouse
- ondblclick: quando ocorre dois clicks com o mouse
- onmousemove: quando mexe o mouse
- onmousedown: quando aperta o botão do mouse
- onmouseup: quando solta o botão do mouse (útil com os dois acima para gerenciar drag'n'drop)
- onkeypress: quando pressionar e soltar uma tecla
- onkeydown: quando pressionar uma tecla
- onkeyup: quando soltar uma tecla
- onblur: quando um elemento perde foco
- onfocus: quando um elemento ganha foco
- onchange: quando um input, select ou textarea tem seu valor alterado
- onload: quando a página é carregada
- onunload: quando a página é fechada
- onsubmit: disparado antes de submeter o formulário (útil para realizar validações)

Existem também uma série de outros eventos mais avançados que permitem a criação de interações para drag-and-drop, e até mesmo a criação de eventos customizados.

9.9 EXERCÍCIOS: MOSTRANDO TAMANHO DO PRODUTO COM JAVASCRIPT

1. Na página **produto.html**, o preenchimento inicial e atualização do valor no output deve ser feita via JavaScript. Quando o input range mudar de valor (evento `oninput`), pegamos seu valor e jogamos no valor do output.

Para escrever o JavaScript, você pode criar um novo arquivo **produto.js**, dentro da pasta **js** e importá-lo na página **produto.html**.

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

O nosso código é:

```
var inputTamanho = document.querySelector('[name=tamanho]')
var outputTamanho = document.querySelector('[name=valortamanho]')

function mostraTamanho(){
    outputTamanho.value = inputTamanho.value
}

inputTamanho.oninput = mostraTamanho
```

Teste o funcionamento no navegador, veja se o output atualiza de valor corretamente.

IE10

Para suportar o IE10, precisamos colocar o evento `onchange`. O correto no HTML5 seria usar o evento `oninput`, que até funciona melhor nos browsers modernos.

```
inputTamanho.oninput = mostraTamanho  
inputTamanho.onchange = mostraTamanho
```

Além disso, como o elemento `output` não é corretamente reconhecido pelo navegador, alterar a propriedade `value` dele não vai ter o resultado esperado. Para o nosso código funcionar nele, precisamos mexer diretamente no texto do elemento:

```
function mostraTamanho() {  
    outputTamanho.value = inputTamanho.value  
    outputTamanho.textContent = event.target.value  
}
```

9.10 FUNÇÕES ANÔNIMAS

No exercício anterior nós indicamos que a função `mostraTamanho` deveria ser executada no momento em que o usuário inserir o tamanho do produto no `<input type="range">`. Note que não estamos executando a função `mostraTamanho`, já que não colocamos os parênteses. Estamos apenas indicando o nome da função que deve ser executada.

```
inputTamanho.oninput = mostraTamanho  
  
function mostraTamanho(){  
    outputTamanho.value = inputTamanho.value  
}
```

Há algum outro lugar do código no qual precisamos chamar essa função? Não! Porém, é pra isso que damos um nome à uma função, para que seja possível usá-la em mais de um ponto do código.

É muito comum que algumas funções tenham uma única referência no código. É o nosso caso com a função `mostraTamanho`. Nesses casos, o JavaScript permite que criemos a função no lugar onde antes estávamos indicando seu nome.

```
inputTamanho.oninput = function() {  
    outputTamanho.value = inputTamanho.value  
}
```

Transformamos a função `mostraTamanho` em uma função sem nome, uma função anônima. Ela continua sendo executada normalmente quando o usuário alterar o valor para o tamanho.

9.11 MANIPULANDO STRINGS

Uma variável que armazena um string faz muito mais que isso! Ela permite, por exemplo, consultar o seu tamanho e realizar transformações em seu valor.

```
var empresa = "Caelum";  
  
empresa.length; // tamanho da string  
  
empresa.replace("lum", "tano"); // retorna Caetano
```

A partir da variável `empresa`, usamos o operador ponto seguido da ação `replace`.

Imutabilidade

String é imutável. Logo, no exemplo abaixo, se a variável `empresa` for impressa após a chamada da função `replace`, o valor continuará sendo "Caelum". Para obter uma string modificada, é necessário receber o retorno de cada função que manipula a string, pois uma nova string modificada é retornada:

```
var empresa = "Caelum";  
  
// substitui a parte "lum" por "tano"  
empresa.replace("lum", "tano");  
console.log(empresa); // imprime Caelum, não mudou!  
  
empresa = empresa.replace("lum", "tano");  
console.log(empresa); // imprime Caetano, mudou!
```

Conversões

O JavaScript possui funções de conversão de string para number:

```
var textoInteiro = "10";  
var inteiro = parseInt(textoInteiro);  
  
var textoFloat = "10.22";  
var float = parseFloat(textoFloat);
```

9.12 MANIPULANDO NÚMEROS

Number, assim como string, também é imutável. O exemplo abaixo altera o número de casas decimais com a função `toFixed`. Esta função retorna uma string, mas, para ela funcionar corretamente, seu retorno precisa ser capturado:

```
var milNumber = 1000;  
var milString = milNumber.toFixed(2); // recebe o retorno da função  
console.log(milString); // imprime a string "1000.00"
```

9.13 CONCATENAÇÕES

É possível concatenar (juntar) tipos diferentes e o JavaScript se encarregará de realizar a conversão

entre os tipos, podendo resultar em algo não esperado.

String com String

```
var s1 = "Caelum";
var s2 = "Inovação";
console.log(s1 + s2); // imprime CaelumInovação
```

String com outro tipo de dados

Como vimos, o JavaScript tentará ajudar realizando conversões quando tipos diferentes forem envolvidos numa operação, mas é necessário estarmos atentos na maneira como ele as realiza:

```
var num1 = 2;
var num2 = 3;
var nome = "Caelum"

// O que ele imprimirá?

// Exemplo 1:
console.log(num1 + nome + num2); // imprime 2Caelum3

// Exemplo 2:
console.log(num1 + num2 + nome); // imprime 5Caelum

// Exemplo 3:
console.log(nome + num1 + num2); // imprime Caelum23

// Exemplo 4:
console.log(nome + (num1 + num2)); // imprime Caelum5

// Exemplo 5:
console.log(nome + num1 * num2); // imprime Caelum6
// A multiplicação tem precedência
```

NaN

Veja o código abaixo:

```
console.log(10 - "curso")
```

O resultado é `NaN` (not a number). Isto significa que todas operações matemáticas, exceto subtração, que serão vistas mais a frente, só podem ser feitas com números. O valor `NaN` ainda possui uma peculiaridade, definida em sua especificação:

```
var resultado = 10 - "curso"; // retorna NaN
resultado == NaN; // false
NaN == NaN; // false
```

Não é possível comparar uma variável com `NaN`, nem mesmo `NaN` com `NaN`! Para saber se uma variável é `NaN`, deve ser usada a função `isNaN`:

```
var resultado = 10 - "curso";
isNaN(resultado); // true
```

9.14 EXERCÍCIOS: CALCULANDO O TOTAL DA COMPRA

1. Para que o usuário possa escolher a quantidade do produto que ele quer comprar, criaremos um campo para a quantidade e outro para exibição do valor total.

Os campos serão inseridos **depois do fechamento** da `div.card.mb-3` e **dentro** da `div.col-md-4` na página **checkout.html**.

Vamos adicionar o código abaixo, criando mais um `card` na nossa página:

```
<div class="card mb-3">
  <div class="card-body">

    <div class="form-group">
      <label for="qtd">Quantidade:</label>
      <input type="number" id="qtd" min="1" max="99" value="1" class="form-control">
    </div>

    <div class="form-group">
      <label for="total">Total:</label>
      <output for="qtd preco" id="total" class="form-control">R$ 48,95</output>
    </div>

  </div>
</div>
```

2. No primeiro `card` que criamos com uma lista de definições, temos um `<dd>` com o preço unitário do nosso produto, precisamos adicionar um `id` nele para que o valor seja acessível de forma mais simples quando implementarmos nosso JavaScript.

Somente **adicone** o **id** no campo que já criamos anteriormente:

```
<dd id="preco">R$ 48,95</dd>
```

3. O usuário já consegue inserir a quantidade que ele deseja, porém, nada acontece. Para que o valor total da compra seja alterado quando o usuário alterar a quantidade, precisaremos do **JavaScript**.

Crie o arquivo **total.js** dentro da pasta **js** e importe ele na sua página **checkout.html**.

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

4. Agora, dentro de **total.js** precisamos acessar os elementos da nossa página. Pegaremos o conteúdo da `<dd>` de preço unitário do produto e multiplicaremos pela quantidade que o usuário digitar no `<input>` de quantidade.

```
var $input_quantidade = document.querySelector("#qtd");
var $output_total = document.querySelector("#total");

$input_quantidade.oninput = function() {
    var preco = document.querySelector("#preco").textContent;
    preco = preco.replace("R$ ", "");
    preco = preco.replace(",", ".");
    preco = parseFloat(preco);

    var quantidade = $input_quantidade.value;
    var total = quantidade * preco;
    total = "R$ " + total.toFixed(2);
    total = total.replace(".", ",");

    $output_total.value = total;
}
```

Agora, teste sua página, o cálculo do total já deve estar funcionando.

ARGUMENTOS EM FUNÇÕES

É possível definir que a função vai ter algum valor variável que vamos definir quando quisermos executá-la:

```
function mostraAlerta(texto) {
    // Dentro da função "texto" conterá o valor passado na execução.
    alert(texto);
}

// Ao chamar a função é necessário definir o valor do "texto"
mostraAlerta("Funciona com argumento!");
```

9.15 ARRAY

O array é útil quando precisamos trabalhar com diversos valores armazenados:

```
var palavras = ["Caelum", "Ensino"];
palavras.push("Inovação"); // adiciona a string "Inovação" por último no array
```

Também é possível guardar valores de tipos diferentes:

```
var variosTipos = ["Caelum", 10, [1,2]];
```

Como obter um valor agora? Lembre-se que o tamanho de um array vai de 0 até o seu tamanho - 1.

```
console.log(variosTipos[1]) // imprime o número 10
```

ADICIONANDO ELEMENTO PELO ÍNDICE

No lugar de usar a função **push**, que adiciona o elemento como último do array é possível fazer:

```
var palavras = ["Caelum", "Ensino"];
palavras[9] = "Inovação";
```

Isso alterará o tamanho do array para dez e adicionará na última posição a string "Inovação", deixando as posições intermediárias com o valor `undefined`.

Outro aspecto interessante é o tamanho do array: podemos adicionar quantos elementos quisermos que seu tamanho aumentará quando necessário.

9.16 BLOCOS DE REPETIÇÃO

Muitas vezes precisamos executar um trecho de código repetidamente até que uma condição seja contemplada, ou enquanto uma condição for verdadeira. Para isso, o JavaScript oferece uma série de blocos de repetição. O mais comum é o **for**.

for

O bloco **for** precisa de algumas informações de controle para evitar que ele execute infinitamente:

```
for /* variável de controle */; /* condição */; /* pós execução */ {
  // código a ser repetido
}
```

Das informações necessárias, somente a condição é obrigatória, mas normalmente utilizamos todas as informações:

```
var palavras = ["Caelum", "Ensino"];

for (var i = 0; i < palavras.length; i++) {
  alert(palavras[i]);
}
```

WHILE

O bloco `while` executa determinado código repetitivamente enquanto uma condição for verdadeira. Diferente do bloco `for`, a variável de controle, bem como sua manipulação, não são responsabilidades do bloco em si:

```
var contador = 1;

while (contador <= 10) {
    alert(contador + " Mississipi...");
    contador++;
}

alert("Valor do contador: " + contador);
```

9.17 FUNÇÕES TEMPORAIS

Em JavaScript, podemos criar um *timer* para executar um trecho de código após um certo tempo, ou ainda executar algo de tempos em tempos.

A função `setTimeout` permite que agendemos alguma função para execução no futuro e recebe o nome da função a ser executada e o número de milissegundos a esperar:

```
// executa a minhaFuncao daqui um segundo
setTimeout(minhaFuncao, 1000);
```

Se for um código recorrente, podemos usar o `setInterval` que recebe os mesmos argumentos mas executa a função indefinidamente de tempos em tempos:

```
// executa a minhaFuncao de um em um segundo
setInterval(minhaFuncao, 1000);
```

É uma função útil para, por exemplo, implementar um banner rotativo, como faremos no exercício a seguir.

CLEARINTERVAL

As funções temporais devolvem um objeto que representa o agendamento que foi feito. É possível usá-lo para cancelar a execução no futuro. É especialmente interessante para o caso do *interval* que pode ser cancelado de sua execução infinita:

```
// agenda uma execução qualquer
var timer = setInterval(minhaFuncao, 1000);

// cancela execução
clearInterval(timer);
```

9.18 EXERCÍCIOS OPCIONAIS: BANNER ROTATIVO

1. Implemente um banner rotativo na **index.html** da Mirror Fashion usando JavaScript.

Temos duas imagens, a **destaque-home.png** e a **destaque-home-2.png** que queremos trocar a cada 4 segundos, para isso, vamos utilizar o `setInterval`.

Há várias formas de implementar essa troca de imagens. Uma sugestão é manter um array com os valores possíveis para a imagem e um inteiro que guarda qual é o banner atual.

Crie o arquivo **banner.js** na pasta **js** e não esqueça de adicionar ele no HTML da **index.html**

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

No arquivo **banner.js**:

```
var banners = ["img/destaque-home.png", "img/destaque-home-2.png"];
var bannerAtual = 0;

function trocaBanner() {
    bannerAtual = (bannerAtual + 1) % 2;
    document.querySelector('.banner-destaque img').src = banners[bannerAtual];
}

setInterval(trocaBanner, 4000);
```

2. (avanhado) Faça um botão de *pause* que pare a troca do banner.

Dica: use o `clearInterval` para interromper a execução.

3. (avanhado) Faça um botão de *play* para reativar a troca dos banners.

Sugestão para o desafio de pause/play

Podemos criar no HTML **index.html** um novo link para controlar a animação. Crie dentro da `section` com classe `banner-destaque`:

```
<a href="#" class="pause"></a>
```

O JavaScript deve chamar `clearInterval` para pausar ou novamente o `setInterval` para continuar a animação.

Precisamos **editar** o código anterior que chamava o `setInterval` para pegar o seu retorno. Será um objeto que controla aquele interval e nos permitirá desligá-lo depois:

```
var timer = setInterval(trocaBanner, 4000);
```

Agora, precisamos **adicionar** nosso código que controla o `pause` e `play`:

```
var controle = document.querySelector('.pause');

controle.onclick = function() {
    if (controle.className == 'pause') {
        clearInterval(timer);
        controle.className = 'play';
    } else {
        timer = setInterval(trocaBanner, 4000);
        controle.className = 'pause';
    }

    return false;
}
```

Por fim, podemos estilizar o botão como pause ou play apenas trabalhando com bordas no CSS, **adicone** os estilos **fora** das *media query* existentes no arquivo **estilos.css**:

```
.destaque {
    position: relative;
}

.pause,
.play {
    display: block;
    position: absolute;
    right: 15px;
    top: 15px;
}

.pause {
    border-left: 10px solid #900;
    border-right: 10px solid #900;
    height: 30px;
    width: 5px;
}

.play {
    border-left: 25px solid #900;
    border-bottom: 15px solid transparent;
    border-top: 15px solid transparent;
}
```



9.19 PARA SABER MAIS: VÁRIOS CALLBACKS NO MESMO ELEMENTO

Nos exercícios que trabalhamos com eventos, usamos o `onclick` e o `onsubmit` diretamente no elemento que estávamos manipulando:

```
document.querySelector('#destaque').onclick = function() {  
    // tratamento do evento  
};
```

É uma forma fácil e portável de se tratar eventos, mas não muito comum na prática. O maior problema do código acima é que só podemos atrelar uma única função ao evento. Se tentarmos, em outra parte do código, colocar uma segunda função para executar no mesmo evento, ela sobrescreverá a anterior.

A maneira mais recomendada de se associar uma função a eventos é com o uso de `addEventListener`:

```
document.querySelector('#destaque').addEventListener('click', function() {  
    // tratamento do evento  
});
```

Dessa maneira, conseguimos adicionar vários listeners ao mesmo evento, deixando o código mais flexível. Só há um porém: embora seja o modo oficial de se trabalhar com eventos, o `addEventListener` não é suportado do IE8 pra baixo.

Para atender os IEs antigos, usamos a função `attachEvent`, semelhante:

```
document.querySelector('#destaque').attachEvent('onclick', function() {  
    // tratamento do evento  
});
```

O problema é ter que fazer sempre as duas coisas para garantir a portabilidade da nossa página. Essa questão é resolvida pelos famosos frameworks JavaScript, como o jQuery, que veremos mais adiante no curso.

9.20 PARA SABER MAIS: CONTROLANDO AS VALIDAÇÕES HTML5

A ideia da nova validação do HTML5 é permitir que os navegadores já possuam uma forma simples de prover validações sem que os desenvolvedores precisem recorrer a complicadas bibliotecas

JavaScript (algo comum em muitas páginas).

No entanto, muitas vezes, as opções padrão do navegador não são exatamente o que precisamos, e queremos mudar o comportamento da validação ou executar validações personalizadas e diferentes.

Podemos, então, usando JavaScript, desabilitar a validação padrão e fazer a nossa própria:

```
<script type="text/javascript">

document.querySelector('form input').oninvalid = function(event) {

    // cancela comportamento padrão do browser
    event.preventDefault();

    // verifica a validade e mostra o alert
    if (!this.validity.valid) {
        alert("Nome obrigatório!");
    }
};

</script>
```

Isso nos permite trocar, por exemplo, todo o visual e forma de apresentação dos erros. E, o melhor, caso o usuário esteja com JavaScript desabilitado, será executada a validação padrão sem problemas. Um ótimo fallback (nas soluções tradicionais de validação com jQuery, por exemplo, tudo se perde quando o usuário desabilita JavaScript).

Outra forma de desabilitar a validação, afetando o formulário inteiro, é colocando o atributo novalidate na tag `<form>`.

Além de desabilitar completamente a validação do navegador, podemos apenas trocar a mensagem de erro, mas ainda usar o mecanismo e design padrão:

```
<script type="text/javascript">

document.querySelector('input[type=email]').oninvalid = function() {

    // remove mensagens de erro antigas
    this.setCustomValidity('');

    // executa novamente a validação
    if (!this.validity.valid) {

        // se inválido, coloca mensagem de erro
        this.setCustomValidity("Email inválido");
    }
};

</script>
```

JQUERY

"O primeiro problema para todos, homens e mulheres, não é aprender, mas desaprender" -- Gloria Steinem

Por conta das dificuldades enfrentadas pelos programadores JavaScript para páginas Web, foi criada uma biblioteca que traz diversas funcionalidades voltadas à solução dos problemas mais difíceis de serem contornados com o uso do JavaScript puro.

A principal vantagem na adoção de uma biblioteca de JavaScript é permitir uma maior compatibilidade de um mesmo código com diversos navegadores. Uma maneira de se atingir esse objetivo é criando funções que verificam quaisquer características necessárias e permitam que o programador escreva um código único para todos os navegadores.

Além dessa vantagem, o jQuery, que é hoje a biblioteca padrão na programação front-end para Web, traz uma sintaxe mais "fluida" nas tarefas mais comuns ao programador que são: selecionar um elemento do documento e alterar suas características.

10.1 JQUERY - A FUNÇÃO \$

O jQuery é uma grande biblioteca que contém diversas funções que facilitam a vida do programador. A mais importante delas, que inicia a maioria dos códigos, é a função \$.

Com ela é possível selecionar elementos com maior facilidade, maior compatibilidade, e com menos código. Por exemplo:

```
// JavaScript "puro"
var cabecalho = document.querySelector("#cabecalho");

if (cabecalho.attachEvent) {
  cabecalho.attachEvent("onclick", function (event) {
    alert("Você clicou no cabeçalho, usuário do IE!");
  });
} else if (cabecalho.addEventListener) {
  cabecalho.addEventListener("click", function (event) {
    alert("Você clicou no cabeçalho!")
  }, false);
}

// jQuery
$("#cabecalho").click(function (event) {
  alert("Você clicou no cabeçalho!");
```

```
});
```

Note como a sintaxe do jQuery é bem menor, e a biblioteca se encarrega de encontrar o modo mais compatível possível para adicionar o evento ao elemento cujo **id** é `cabecalho`.

Existem diversas funções que o jQuery permite que utilizemos para alterar os elementos que selecionamos pela função `$`, e essas funções podem ser encadeadas, por exemplo:

```
$("#cabecalho").css({"margin-top": "20px", "color": "#333333"})
    .addClass("selecionado");
```

No código acima, primeiramente chamamos a função `$` e passamos como argumento uma string idêntica ao seletor CSS que utilizariamos para selecionar o elemento de id `cabecalho`. Na sequência chamamos a função `css` e passamos um objeto como argumento, essa função adicionará ou alterará as informações desse objeto como propriedades de estilo do elemento que selecionamos com a função `$`. Em seguida chamamos mais uma função, a `addClass`, que vai adicionar o valor "selecionado" ao atributo `class` do elemento com o id `cabecalho`.

Dessa maneira, é possível fazer muito mais com muito menos código, e ainda por cima de uma maneira que funciona em diversos navegadores.

10.2 JQUERY SELECTORS

Um dos maiores poderes do jQuery está na sua capacidade de selecionar elementos a partir de seletores CSS.

Como já aprendemos, existem diversas formas de selecionarmos quais elementos ganharão determinado estilo. Infelizmente muitos desses seletores não funcionam em todos os navegadores. Contudo, no jQuery, temos todos à nossa disposição.

Por exemplo, se quisermos esconder todas as tags `<td>` filhas de um `<tbody>`, basta:

```
$('#tbody td').hide();
```

Seletores mais comuns:

```
// pinta o fundo do formulário com id "form" de preto
$('#form').css('background', 'black');

// esconde todos os elementos com o atributo "class" igual a "headline"
$('.headline').hide();

// muda o texto de todos os parágrafos
$('p').text('alô :D');
```

Mais exemplos:

```
$(‘div > p:first’); // o primeiro elemento <p> imediatamente filho de uma <div>
$(‘input:hidden’); // todos os inputs invisíveis
```

```

$(‘input:selected’); // todas os checkboxes selecionados
$(‘input[type=button]’); // todos os inputs com type="button"
$(‘td, th’); // todas as tds e ths

```

Lembre-se de que a função que chamamos após o seletor é aplicada para **todos** os elementos retornados. Veja:

```

// forma ineficiente
alert($('div').text() + $('p').text() + $('ul li').text());

// forma eficiente :D
alert($('div, p, ul li').text());

```

A função `text()` é chamada para todos os `<div>`s, `<p>`s, e ``s filhos de ``s.

10.3 FILTROS CUSTOMIZADOS E POR DOM

Existem diversos seletores herdados do css que servem para selecionar elementos baseados no DOM. Alguns deles são:

```

$('div > p'); // <p>s imediatamente filhos de <div>
$('p + p'); // <p>s imediatamente precedidos por outro <p>
$('div:first-child'); // um elemento <div> que seja o primeiro filho
$('div:last-child'); // um elemento <div> que seja o último filho
$('div > *:first-child'); // um elemento que seja o primeiro filho direto de uma <div>
$('div > *:last-child'); // um elemento que seja o último filho direto de uma <div>
$('div p:nth(0)'); // o primeiro elemento <p> filho de uma <div>
$('div:empty'); // <div>s vazias

```

10.4 UTILITÁRIO DE ITERAÇÃO DO JQUERY

O jQuery traz também entre suas diversas funcionalidades, uma função que facilita a iteração em elementos de um Array com uma sintaxe mais agradável:

```

$("#menu-departamentos li").each(function (index, item) {
    alert(item.text());
});

```

A função `each` chamada logo após um seletor executa a função que passamos como argumento para cada um dos itens encontrados. Essa função precisa de dois argumentos. O primeiro será o "índice" do elemento atual na coleção (0 para o primeiro, 1 para o segundo e assim por diante), e o segundo será o próprio elemento.

Também é possível utilizar a função `each` do jQuery com qualquer Array:

```

var pessoas = ["João", "José", "Maria", "Antônio"];

$.each(pessoas, function(index, item) {
    alert(item);
})

```

Nesse caso, chamamos a função `each` diretamente após o `$`, pois essa implementação é um método do próprio objeto `$`. Passamos dois argumentos, o primeiro é o Array que queremos percorrer e o segundo a função que desejamos executar para cada um dos itens do Array.

10.5 CARACTERÍSTICAS DE EXECUÇÃO

Para utilizarmos o jQuery em nossos projetos com maior segurança, devemos tomar alguns cuidados.

Importação

Antes de mais nada é necessário incluir o jQuery em nossa página. Só assim o navegador executará seu código para que possamos utilizar suas funcionalidades em nosso código.

Por isso é necessário que a tag `<script>` do jQuery seja a primeira de todas na ordem de nosso documento:

```
<script type="text/javascript" src="scripts/jquery.js"></script>
<!-- só podemos utilizar o jQuery após sua importação -->
<script type="text/javascript" src="scripts/meuscript.js"></script>
<script type="text/javascript" src="scripts/meuoutroscript.js"></script>
```

Executar somente após carregar

Como estamos constantemente selecionando elementos do documento e alterando suas características, é importante garantir que os elementos que pretendemos utilizar já tenham sido carregados pelo navegador.

A melhor maneira de garantir isso é somente executar nosso script após o término do carregamento total da página com a função `$` dessa maneira:

```
$(function () {
  $("#cabecalho").css({"background-color": "#000000"});
})
```

Essa função `$` que recebe uma função anônima como argumento garante que o código dentro dela só será executado ao fim do carregamento de todos os elementos da página.

10.6 MAIS PRODUTOS NA HOME

Uma técnica comum de se implementar com JavaScript é a de permitir mais conteúdo ser mostrado na tela a partir de algum clique ou até ao se passar o mouse em cima.

Na nossa página, exibimos 6 produtos em cada painel de destaque. Poderíamos criar um botão para "Mostrar mais" produtos que exiba mais 6 produtos.

Para implementar, a maneira mais simples é inserir esses produtos adicionais no HTML e escondê-los com CSS usando `display: none`. Aí colocamos o botão de *Mostrar Mais* e, via JavaScript, exibimos quando o usuário clicar.

CARREGAMENTO DE CONTEÚDO COM AJAX

No nosso exercício, vamos apenas esconder ou exibir o conteúdo usando CSS e JavaScript. Em alguns casos, pode ser interessante baixar conteúdo novo do servidor no momento do clique.

Esse tipo de página usa Ajax para requisitar novos dados ao servidor e inseri-los dinamicamente na página via JavaScript. Ajax e outras técnicas de JavaScript avançadas são tópicos do curso WD-47 da Formação Web da Caelum:

<http://www.caelum.com.br/curso/wd47>

10.7 EXERCÍCIOS: JQUERY NA HOME

1. Na página **index.html**, crie um `<button>` no final de cada **section painel**, logo após a `</nav>`. Esse será o botão responsável por exibir mais produtos.

```
<button type="button">Mostrar mais</button>
```

Já temos 6 produtos em cada `painel` na página **index.html**, precisamos acrescentar mais **6** produtos em cada `painel` para que esses apareçam quando apertarmos o botão que acabamos de criar. **Adicione** as `` com os produtos dentro das `` de cada `painel`, **novidades** e **mais-vendidos**.

2. Para que possamos usar o jQuery, precisamos importá-lo na **index.html**. Inclua a seguinte linha **imediatamente antes** da importação de qualquer outro JavaScript que tenha sido importado:

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

Agora crie o arquivo **home.js** dentro da pasta **js**, onde vamos colocar nosso código para o botão funcionar, e o importe na **index.html** depois da importação do **jQuery**:

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

3. Implemente a funcionalidade de compactar o painel de produtos para mostrar apenas os 6 primeiros por padrão. Vamos fazer isso no arquivo **estilos.css**, **fora** das *media queries* existentes.

```
.painel li:nth-child(n+7) {  
    display: none;  
}
```

Vamos criar uma funcionalidade com a classe `painel-aberto` que vamos adicionar ao `painel`

quando apertarmos o botão e assim aparecerá os produtos escondidos. Vamos fazer isso no arquivo **estilos.css**, **fora** das *media queries* existentes.

```
.painel-aberto li:nth-child(n+7) {  
    display: inline-block;  
}
```

Essa classe, claro, só vai fazer efeito se adicionarmos ela na página. Para testar, vá na div com classe `novidades` e **adicone** a classe `painel-aberto` do lado.

Agora que você já viu funcionando, o primeiro passo é **remover** a classe `painel-aberto` do HTML que você acabou de colocar.

4. Vamos implementar a funcionalidade em JavaScript. Como a classe `painel-aberto` é uma classe atrelada a funcionalidade JavaScript, vamos adicioná-la com jQuery, apenas se o botão for clicado.

Vamos aplicar o JavaScript para o painel `novidades`. Implemente o evento de clique no botão e quando clicado, adiciona a classe `painel-aberto`. Adicione o código no arquivo **home.js**:

```
$('.novidades button').click(function() {  
    $('.novidades').addClass('painel-aberto');  
})
```

Teste a funcionalidade no navegador.

5. Agora quando clicamos no botão do painel `novidades` os produtos adicionais já aparecem, mas quando clicamos novamente no botão, não acontece nada. Para que isso aconteça, precisamos utilizar uma função do jQuery que consegue verificar se a classe já existe. Essa função remove a classe, se ela já existir e se não existir, ela adiciona. Essa função é a `toggleClass`. Altere o arquivo **home.js**:

```
$('.novidades button').click(function() {  
    $('.novidades').toggleClass('painel-aberto');  
})
```

Teste a funcionalidade no navegador.

6. (opcional) Implemente a mesma funcionalidade para o painel da direita, o `mais-vendidos`.
7. (opcional trabalhoso) Podemos estilizar o botão de mostrar mais produtos com regras CSS3 que aprendemos. Adicionar no arquivo **estilos.css**, **fora** das *media queries* existentes.

```
.painel button {  
    /* posicionamento */  
    float: right;  
    margin-right: 10px;  
    padding: 10px;  
  
    /* estilo */  
    background-color: #333;  
    border: 0;  
    border-radius: 4px;
```

```

    box-shadow: 1px 1px 3px rgba(30, 30, 30, 0.5);
    color: white;
    font-size: 1em;
    text-decoration: none;
    text-shadow: 1px 0 1px black;

    /* animação */
    transition: 0.3s;
}

.painel button:hover {
    background-color: #393939;
    box-shadow: 1px 0 20px rgba(200, 200, 120, 0.9);
}

```

Se tiver disposição, adicione mais detalhes:

```

.painel button {
    /* posicionamento */
    position: relative;
    margin-bottom: 10px;
}

.painel button::after {
    /* elemento vazio */
    content: '';
    display: block;
    height: 0;
    width: 0;

    /* triângulo */
    border-top: 10px solid #333;
    border-left: 10px solid transparent;
    border-right: 10px solid transparent;

    /* posicionamento */
    position: absolute;
    top: 100%;
    left: 50%;
    margin-left: -5px;

    /* animação */
    transition: 0.3s;
}

.painel button:hover::after {
    border-top-color: #393939;
}

```

8. (opcional avançado) Em vez de escrever o código 2x para suportar os 2 painéis, podemos generalizá-lo e resolver as duas coisas de uma vez.

```

$('.painel button').click(function() {
    $(this).parent().toggleClass('painel-aberto');
})

```

10.8 PLUGINS JQUERY

Além de usar os componentes JavaScript que vêm prontos no Bootstrap, podemos baixar outros

plugins feitos para o jQuery ou para o Bootstrap que trazem novas funcionalidades.

A grande riqueza do jQuery é justamente sua vasta gama de plugins disponíveis. Há até mesmo um diretório no site deles:

<http://plugins.jquery.com/>

Cada plugin é um arquivo JavaScript que você inclui na página e adiciona uma funcionalidade específica. Muitos exigem que escrevemos um pouco de código para configurar seu uso; outros são mais plug and play.

Você vai precisar consultar a documentação do plugin específico quando for usar.

10.9 EXERCÍCIOS: PLUGIN

1. Um plugin que podemos usar na nossa página é **máscaras numéricas** para digitar em campos como CPF ou CEP. Isso ajuda bastante o usuário.

Para usar esse plugin, basta invocar seu arquivo JavaScript no final da página do **checkout.html**, logo após a chamada do **jQuery.js** e do **bootstrap.js**:

```
<script type="text/javascript" src="js/inputmask-plugin.js"></script>
```

2. Cada campo que for usar uma máscara numérica precisa definir o atributo **data-mask** com o formato a ser usado.

No **<input>** do CPF, adicione o atributo:

```
data-mask="999.999.999-99"
```

No **<input>** do número do cartão com código de verificação, podemos adicionar:

```
data-mask="9999 9999 9999 9999 - 999"
```

APÊNDICE - INTEGRAÇÕES COM SERVIÇOS WEB

"Pessoas viviam em fazendas, depois foram viver nas cidades. Agora todos nós vamos viver na Internet"
-- Sean Parker

11.1 WEB 2.0 E INTEGRAÇÕES

Boa parte do grande poder da Web, de estarmos conectados o tempo todo, é o de permitir a integração entre as páginas. A Web nasceu com esse conceito de tudo interligado, por meio dos links.

Mas a tal Web 2.0 trouxe ideias ainda mais complexas. Interligar páginas e serviços diferentes, criando novos resultados a partir de outras páginas.

São mapas do Google Maps espalhados em vários sites por aí. Ou os onipresentes botões de curtir do Facebook. E muitos outros exemplos.

11.2 IFRAMES

Uma das formas de se fazer esse tipo de integração é com o uso da tag `<iframe>`. Ela nos permite embutir o conteúdo de uma outra página no meio da nossa muito facilmente:

```
<iframe src="outrapagina.html"></iframe>
```

Podemos incluir páginas internas ou externas. E quando se trata das externas, é fácil usar esse recurso para incluir componentes reaproveitáveis de outros serviços. Como mapas.

11.3 VÍDEO EMBUTIDO COM YOUTUBE

O YouTube provê uma integração muito fácil com nossas páginas. Podemos incluir um vídeo qualquer pra ser tocado diretamente em nossa página.

Basta entrar no YouTube e copiar o endereço de um `<iframe>` que eles disponibilizam para ser embutido na nossa página.

Vamos fazer isso em nosso projeto.

11.4 EXERCÍCIOS: IFRAME

1. Acesse o vídeo institucional da Mirror Fashion: <http://youtu.be/Tb06abHE4hY>

Vamos embutir o vídeo em nossa página **sobre.html**, no meio do texto explicativo. Para obter o código de embutir no YouTube, localize a opção **share/compartilhar** e vá na aba **Embed**. Ele vai te dar o código HTML do iframe.

```
<iframe width="420" height="315" src="http://www.youtube.com/embed/Tb06abHE4hY"
frameborder="0" allowfullscreen></iframe>
```

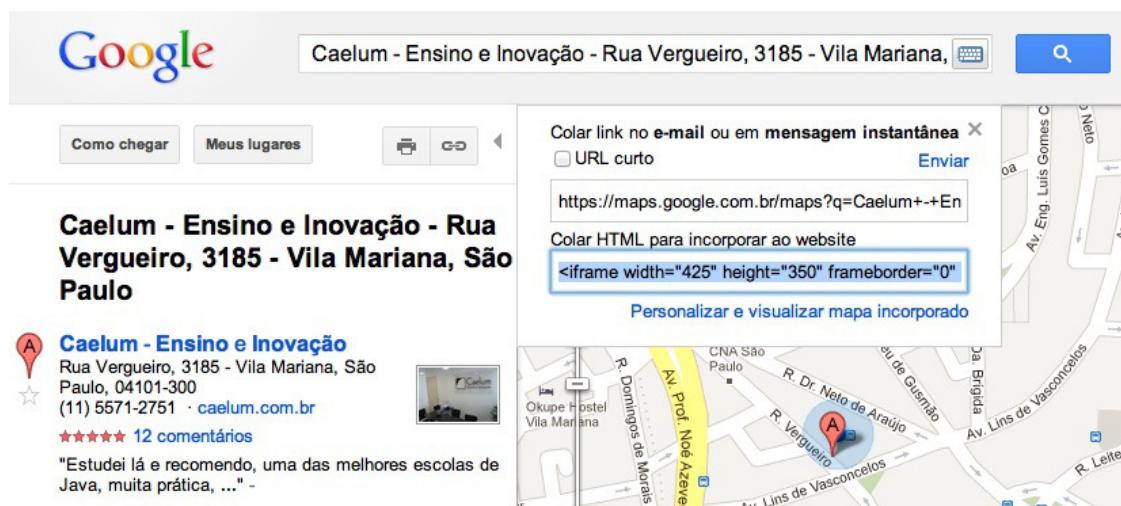
Copie o código do `<iframe>` e cole na página **sobre.html** onde achar melhor. Teste a página.

11.5 EXERCÍCIOS OPCIONAIS: GOOGLE MAPS

1. O Google Maps possui recurso parecido de embutir um mapa completo através de um `<iframe>`.

Abra o Google Maps no seu navegador e acesse um endereço que usaremos para o site da Mirror Fashion.

Clique em "compartilhar", na aba "incorporar um mapa", copie o código que aparece na opção HTML (é um iframe).



2. Na página **sobre.html**, adicione o `<iframe>` copiado no ponto que achar mais conveniente para exibir pro usuário:

```
<iframe width="425" height="350" frameborder="0"
scrolling="no" marginheight="0" marginwidth="0"
src="http://maps.google.com.br/maps?q=Jacarezinho&output=embed">
</iframe>
```

Abra a página e veja o mapa lá.

11.6 FONTES CUSTOMIZADAS COM @FONT-FACE

Fontes na Web sempre foram um problema. Só podemos declarar no `font-family` fontes que estejam disponíveis na máquina quando o usuário visualizar a página, o que restringe bastante as opções.

Porém, desde o CSS2.1 é possível incorporar novas fontes declarando a propriedade `@font-face` indicando a URL onde o navegador pode baixar aquela fonte:

```
@font-face {  
    font-family: "Minha Fonte";  
    src: url("minhafonte.ttf");  
}
```

E, surpreendentemente, essa funcionalidade existe desde o Internet Explorer 4.

Mas nem tudo são flores. O suporte entre navegadores é um imenso problema. Cada um suporta um tipo de arquivo. Fontes true type, por exemplo, padrão entre designers, só é suportado a partir do IE9. O IE usava fontes em formato EOT, outros navegadores TTF, o iPhone só SVG e ainda surgiu um terceiro formato WOFF, que agora faz parte da especificação.

11.7 SERVIÇOS DE WEB FONTS

Lidar com essas diferenças entre navegadores é um problema, e isso sem contar a dificuldade de se obter as fontes legalmente em vários formatos e servi-las corretamente e de maneira otimizada.

Ultimamente surgiram serviços de web fonts de terceiros, que oferecem toda a infraestrutura necessária para usarmos fontes na Web sem problemas e sem precisarmos instalar nada. As fontes são servidas direto dos servidores do serviço em uso da maneira correta e rápida.

Um dos mais famoso é o **Typekit**: <http://typekit.com/>

É um serviço pago mas com preços acessíveis e planos com todo tipo de fonte. Eles têm um catálogo imenso com fontes famosas e de altíssima qualidade.

A alternativa gratuita mais famosa é o **Google Web Fonts**: <https://www.google.com/fonts>

É provido pelo Google apenas com fontes abertas e gratuitas. Seu catálogo é, portanto, mais limitado, mas possui excelentes opções e é muito fácil de ser integrado a uma página Web, bastando importar um CSS deles e usar a fonte.

11.8 EXERCÍCIOS: GOOGLE WEB FONTS

1. Vamos usar duas fontes do **Google Web Fonts** na nossa página inicial. Uma fonte base **PT Sans** e outra para os títulos dos painéis, **Bad Script**.

Importe as fontes via CSS no topo da nossa página **index.html**:

```
<link href='http://fonts.googleapis.com/css?family=PT+Sans|Bad+Script'  
      rel='stylesheet'>
```

Use as fontes no **estilos.css** referenciando-as pelo nome:

```
body {  
    font-family: 'PT Sans', sans-serif;  
}  
  
.painel h2 {  
    font-family: 'Bad Script', sans-serif;  
}
```

Abra a página no navegador e veja as mudanças visuais.



2. (opcional) Navegue no catálogo de fontes do Google em <https://www.google.com/fonts> e escolha uma outra fonte para usarmos no título da página de produtos.

APÊNDICE - OTIMIZAÇÕES DE FRONT-END

"A esperança...: um sonho feito de despertares" -- Aristóteles

Estudos de diversas empresas ao redor do mundo já provaram que sites rápidos, além de mais prazerosos de usar, trazem mais conversões, vendas e usuários felizes.

A Amazon, por exemplo, descobriu que cada 100ms de melhora na velocidade de carregamento da página trazia um crescimento de 1% em seu faturamento.

O Yahoo! provou que cada 400ms de melhora em sua homepage provoca um aumento de 9% no tráfego dos usuários.

A Mozilla tornou suas páginas 2.2s mais rápidas e, com isso, causou um aumento de 15% nos downloads do Firefox. São 60 milhões de downloads a mais por ano.

O Google descobriu que aumentar o tempo de carregamento da página de busca de 0.4s para 0.9s ao subir o número de resultados de 10 para 30 diminuía o tráfego em 20%.

Até a Caelum já fez experimentos em seu Site. No nosso caso, uma página que tinha tempo de carregamento de 6s em comparação com uma de 2s causava uma perda de 18% na taxa de conversões.

Otimização é coisa de programadores

Um dos maiores erros a se cometer com relação à performance é pensar que é um problema exclusivo da programação da parte server-side do projeto. Certamente um código ruim no servidor pode causar imensos gargalos de performance. Uma busca mal feita no banco de dados, um algoritmo pesado de executar etc.

Na esmagadora maioria das situações, a realidade é que o processamento server-side é responsável por menos de 10% do tempo total de carregamento de uma página. A maior parte dos gargalos de performance está em práticas client-side!

Um estudo que fizemos com 100 Sites de participantes de uma conferência de tecnologia de alto nível técnico - a QCon SP de 2011 - trouxe dados interessantes. Nessa amostra, o tempo médio de carregamento da página era de 9s (com casos bem graves, demorando mais de 30s). Mesmo assim, 75%

dos Sites executavam em menos de 400ms no servidor. Os outros 8s são gastos em outros pontos, muito ligados a práticas client-side que veremos.

12.1 HTML E HTTP - COMO FUNCIONA A WORLD WIDE WEB?

Apesar de que conhecer profundamente o funcionamento do protocolo HTTP não seja estritamente necessário para a criação de páginas Web, entender como as coisas funcionam internamente nos ajuda a entender uma série de técnicas e conceitos, resultando em maior qualidade na criação de páginas, além de contribuir para a confiança do programador ao enfrentar um novo desafio.

A primeira coisa que devemos levar em consideração ao conhecer o ciclo de comunicação entre o navegador (cliente) e o servidor, é que o cliente deve conhecer a localização da página (recurso) que ele deseja obter e exibir ao usuário final. O cliente deve ser informado de qual o endereço do recurso necessário em determinado momento, normalmente o usuário final provê essa informação entrando um endereço na barra de endereços do navegador, ou clicando em um link.

Esse endereço é conhecido como URL (Universal Resource Locator), por exemplo:

`http://209.85.227.121:80/index.html`

O endereço exemplificado é composto por 4 partes básicas. A primeira delas é o protocolo de comunicação a ser utilizado para a obtenção do recurso. No exemplo acima o protocolo requerido é o HTTP. A comunicação entre um cliente (geralmente o navegador) e um servidor pode ser feita com o uso de diversos protocolos, por exemplo o FTP (File Transfer Protocol) para a transferência de arquivos, ou o protocolo *file*, de leitura direta de arquivo quando desejamos obter um recurso acessível diretamente pelo computador sem utilizar uma conexão com um servidor Web.

Hoje em dia, os navegadores não precisam que explicitemos o protocolo HTTP em sua barra de endereços, sendo ele o padrão para as requisições de páginas Web.

A segunda e terceira partes, entre `//` e `/`, são o endereço IP do servidor (onde está hospedado o recurso que queremos) e a porta de comunicação com o servidor. Os servidores Web utilizam a porta 80 por padrão, então no exemplo poderíamos ter omitido essa informação que a comunicação seria feita com sucesso.

O endereço IP é um código composto de 4 octetos representados em formato decimal separados por um ponto, é um número um tanto difícil de ser memorizado, (a próxima geração de endereços IP, criada para evitar o fim dos endereços disponíveis é formada por 8 grupos de 4 dígitos hexadecimais separados por ::), por exemplo: `2001:0db8:85a3:08d3:1319:8a2e:0370:7344`) por isso a Web utiliza servidores de nomeação de domínios (DNS), para que o usuário final possa informar um nome em vez de um número e uma porta, por exemplo `www.caelum.com.br`.

A quarta parte é o caminho do recurso que desejamos obter dentro do servidor. No nosso exemplo

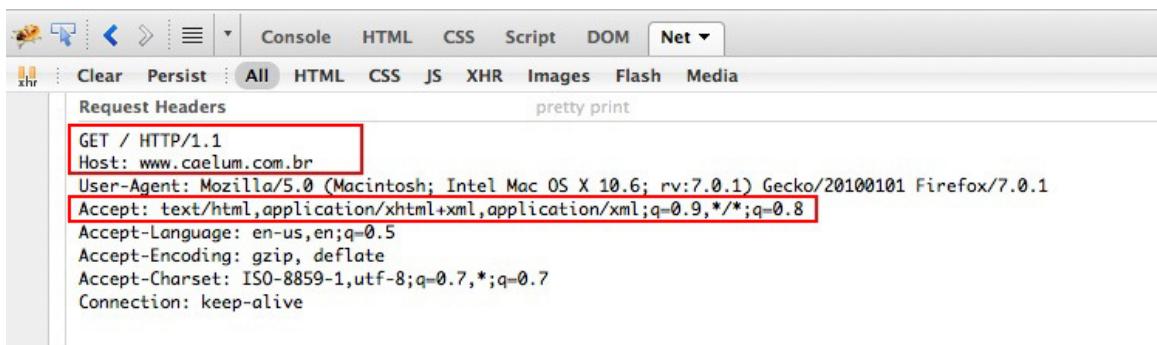
estamos solicitando o arquivo `index.html`, que é o nome padrão de arquivo para a página inicial de um Site, e, nesse caso, também poderia ser omitido. A adoção desses valores padrões permite que para obtermos a página inicial do site da Caelum, por exemplo, os usuários finais digitem somente **www.caelum.com.br** na barra de endereços de seu navegador.

O ciclo HTTP

Conhecemos um pouco sobre URLs, e as utilizaremos quando formos adicionar links e recursos externos em nossos documentos, mas o que acontece quando clicamos em um link ou digitamos um endereço no navegador e clicamos em "ir"? Essas ações disparam uma chamada, dando início ao ciclo HTTP. Essa chamada é o que chamamos de **request** (requisição).

A correta comunicação entre os dois lados do ciclo depende de uma série de informações. Um HTTP request leva consigo todos os dados necessários para que o lado do servidor tome a decisão correta sobre o que fazer. Existem algumas ferramentas que permitem que observemos quais são essas informações.

O protocolo HTTP pode ser utilizado por uma série de aplicações, para uma série de finalidades. Nosso foco é no uso do HTTP para páginas da Web que podemos acessar de um navegador. Alguns navegadores incluem ferramentas de inspeção da página em exibição, e a maioria dessas ferramentas consegue nos mostrar o conteúdo da requisição HTTP. Uma dessas ferramentas é o complemento **Firebug**, disponível para o navegador Firefox.

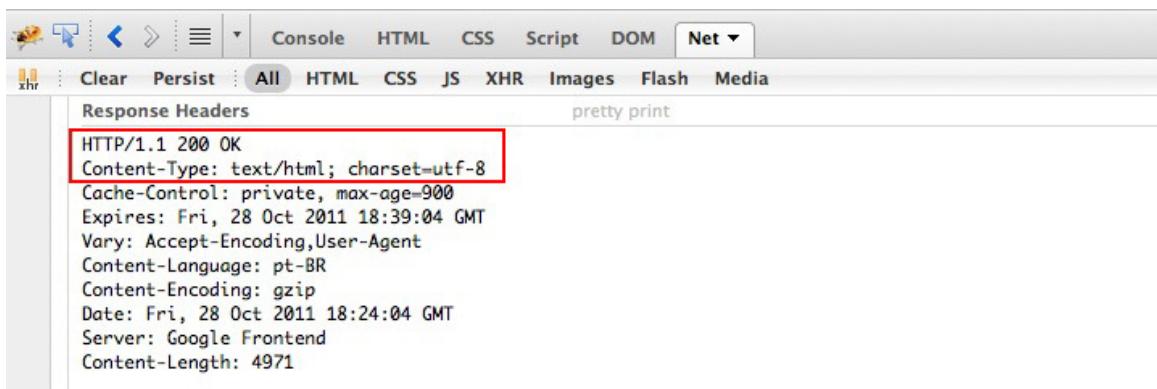


Na imagem estão destacadas as informações mais relevantes da requisição. A primeira delas é a palavra **GET**. GET é um dos métodos suportados pelo protocolo HTTP para realizar a comunicação, e ele deve ser utilizado quando queremos **obter** um recurso que o servidor tem acesso. Caso o servidor encontre o recurso que queremos, ele retorna para o cliente uma **response** (resposta) contendo o recurso que desejamos.

Outras informações importantes são o endereço do recurso que desejamos obter (**Host**) e o tipo de recurso que o cliente espera obter (**Accept**). No exemplo, esperamos encontrar uma página HTML pelo endereço **www.caelum.com.br**.

Com essas informações, o servidor processa o pedido e prepara uma **response** (resposta). Essa

resposta contém uma série de informações importantes para que o cliente possa tomar as decisões necessárias e, no caso de sucesso na comunicação, exibir o recurso para o usuário.



The screenshot shows the Network tab of a browser's developer tools. The 'Response Headers' section is selected. A red box highlights the 'Content-Type' header, which is listed as 'text/html; charset=utf-8'. Other headers visible include 'HTTP/1.1 200 OK', 'Cache-Control: private, max-age=900', 'Expires: Fri, 28 Oct 2011 18:39:04 GMT', 'Vary: Accept-Encoding,User-Agent', 'Content-Language: pt-BR', 'Content-Encoding: gzip', 'Date: Fri, 28 Oct 2011 18:24:04 GMT', 'Server: Google Frontend', and 'Content-Length: 4971'. There is also a 'pretty print' link next to the headers.

As informações mais importantes, no nosso caso, são o código da resposta e o tipo do recurso encontrado (**Content-Type**). No nosso exemplo, o código **200** indica que o recurso foi localizado com sucesso e incluído na resposta.

Todas essas informações que vimos até aqui fazem parte dos **cabeçalhos** da requisição e da resposta. São informações irrelevantes para o usuário, porém essenciais para o correto tratamento das informações solicitadas. As informações que serão exibidas no navegador para o usuário estão contidas no **corpo** da resposta. No nosso exemplo, assim como em toda requisição solicitando uma página Web, o corpo da resposta contém as informações marcadas para exibição correta no navegador.

12.2 PRINCÍPIOS DE PROGRAMAÇÃO DISTRIBUÍDA

Uma página Web é uma aplicação distribuída. Isso significa que há uma comunicação via rede entre dois pontos. No caso, o navegador e o servidor da página.

E, como toda aplicação distribuída, há alguns princípios básicos de performance. Quando há comunicação remota envolvida, em geral, queremos:

Diminuir o volume de dados trafegados entre as partes.

e

Diminuir o número de chamadas remotas.

12.3 FERRAMENTAS DE DIAGNÓSTICO - YSLOW E PAGESPEED

O primeiro passo é saber o que melhorar. Há diversas boas práticas pregadas na literatura de performance Web. Melhor ainda é a existência ferramentas automatizadas para diagnóstico que analisam sua página e dão dicas sobre o quê e como melhorar. Há até uma nota de 0 a 100 para você saber o quanto

bem está nas práticas de otimização.

As mais famosas ferramentas são duas extensões, a **YSlow** feita pelo pessoal do Yahoo! e a **PageSpeed** feita pelo Google. Ambas são extensões para o Firefox e para Chrome - mas rodam melhor no Firefox. Para instalá-las, primeiramente, você vai precisar do Firebug, depois é só baixá-las nos respectivos sites:

<http://developer.yahoo.com/yslow/> <http://code.google.com/speed/page-speed/download.html>

O PageSpeed tem até uma versão online que analisa suas páginas que já estejam publicadas em algum endereço na internet:

<https://developers.google.com/pagespeed/>

YSlow

Abra o Firebug e clique no YSlow. Ele te mostra uma nota para as otimizações da página e sugestões do que melhorar.

Dê uma olhada nas regras. Note que há algumas que envolvem programação no servidor, como configurar compressão GZIP ou acertar os headers HTTP. É uma boa conversar com os programadores do projeto para também fazerem esses acertos no servidor, além do que você já vai fazer no client-side.

Várias regras dizem respeito a otimizações que já podemos implementar como comprimir/minificar nossos CSS e JavaScript, algo que veremos no tópico seguinte.

PageSpeed

Abra o Firebug e vá na aba do PageSpeed. Ele lhe mostra uma nota para a página e diversas práticas de otimização. Aquelas que estão em vermelho são as que você deveria fazer mas não fez. Amarelos são algumas sugestões e verdes são as que você está bem - mas às vezes há mais sugestões até nessas.

Para saber mais: [WebPageTest.org](#)

Outra ferramenta interessante e online é a WebPageTest.org. Ela também nos dá notas e sugestões de melhoria. Um diferencial bem interessante é que ela executa a página em navegadores reais em diversos lugares do mundo e nos dá métricas de tempo de carregamento, e até um vídeo mostrando a performance de acordo com o tempo.

Se você já tem a página publicada em algum endereço, é bem interessante testar essa ferramenta também.

12.4 COMPRESSÃO E MINIFICAÇÃO DE CSS E JAVASCRIPT

Durante todo o curso, aprendemos diversas boas práticas de codificação CSS e JavaScript. E, como toda programação, um ponto importante é manter a legibilidade do código.

Dar bons nomes a variáveis, escrever bons comentários, escrever código identado, com bom espaçamento visual etc.

Entretanto, nada disso é necessário no momento do navegador renderizar a página. Um comentário no código é completamente inútil na hora da execução. Assim como espaços, identação ou nomes de variáveis legíveis.

Mais que isso, todas essas práticas adicionam bytes e mais bytes aos arquivos CSS e JavaScript. Arquivos esses que vão ser baixados pelos navegadores de todos os nossos usuários com o único objetivo de executá-los. Porque então gastar dados trafegando comentários e outras coisas inúteis?

Uma otimização muito importante e extremamente fácil de se implementar com as ferramentas atuais é o que chamamos de **minificação dos arquivos CSS e JavaScript**.

Rodamos um programa compressor nos nossos arquivos para tirar todos esses bytes desnecessários para simples execução. O resultado são arquivos CSS e JavaScript completamente idênticos em funcionalidade mas sem bytes de comentários, espaços etc. Até variáveis longas são reescritas com nomes mais curtos - como apenas 'a', 'b' etc.

Mas repare que não estamos defendendo que você deva *escrever seu código* retirando comentários, identação etc. A boa prática continua sendo escrever código legível e bem documentado. Queremos apenas que, antes de colocar o site no ar, os arquivos sejam minificados. E, com as ferramentas automáticas de hoje em dia, é muito fácil fazer isso.

YUI Compressor

Há diversas ferramentas para compressão de CSS e JavaScript. Uma das mais famosas é o **YUI Compressor** do Yahoo!. Por ser em Java, é multiplataforma e fácil de se usar. Ele comprime tanto código CSS quanto código JavaScript.

Você pode baixá-lo em <http://developer.yahoo.com/yui/compressor/>

Ele é uma ferramenta de linha de comando, o que torna muito fácil automatizar sua execução antes do site ser colocado no ar, por exemplo.

Usá-lo é bem simples:

```
java -jar yuicompressor-x.y.z.jar script.js -o script-min.js
```

Este comando vai ler o arquivo `script.js`, minificar seu conteúdo e gravar o resultado em `script-min.js`. O mesmo poderia ser feito com arquivos CSS.

TESTANDO ONLINE

Diversos sites oferecem uma interface Web para o YUI - e outros compressores. São úteis para você testar e ver logo o impacto sem instalar nada, mas são mais chatos para se automatizar.

<http://refresh-sf.com/yui/>

12.5 COMPRESSÃO DE IMAGENS

Imagens são também fortes candidatas a otimizações. Quão importante será fazer isso?

O **HTTPArchive.org** armazena informações históricas coletadas mensalmente sobre os 17 mil sites mais acessados da Internet mundial. Com base nelas, compila alguns gráficos e dados interessantes.

E, com relação a imagens, os estudos mostram que mais de 60% do peso de uma página está nelas:

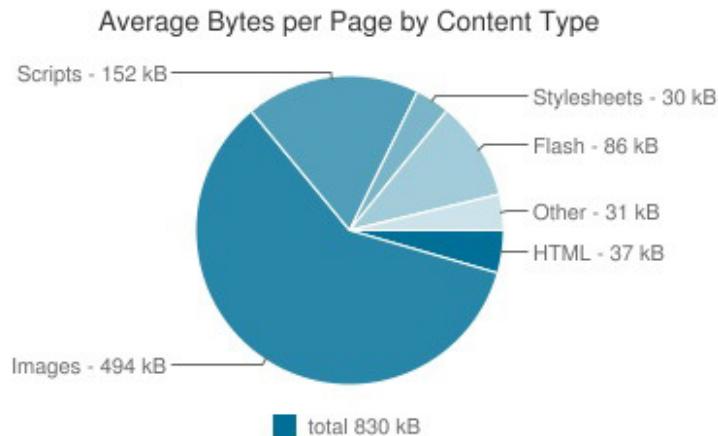


Figura 12.3: Participação de cada tipo de arquivo no tamanho total das páginas

Se conseguir otimizar um pouco as imagens, o resultado final será de grande impacto para a página!

Otimizações lossy

A otimização mais direta relacionada a isso é diminuir a qualidade das imagens. Quando você salva um JPG, pode escolher o grau de compressão, obtendo imagens menores mas sacrificando um pouco a qualidade. Chamamos esse tipo de otimização de *lossy*, pois há perda de qualidade.

É preciso avaliar até que ponto se pode sacrificar a qualidade do design em prol da performance. No entanto, tenha em mente que muitas otimizações podem acabar sendo imperceptíveis para o olho humano - ainda mais o olho ágil e desatento dos usuários Web que varrem as páginas rapidamente e

podem nem perceber que uma imagem está com menos definição.

Você pode otimizar fotos JPG manualmente no seu editor de imagens favorito e chegar a um meio termo entre qualidade e tamanho final. Ou então, pode tentar o excelente serviço online **JPEGMini** que promete diminuir o tamanho das suas imagens diminuindo a qualidade de maneira praticamente imperceptível. Eles usam um algoritmo que promete simular as características da percepção do olho humano, o que lhes permite piorar a qualidade da imagem apenas em pontos que são pouco percebidos pelo nosso olhar.

<http://www.jpegmini.com/>

Design otimizado

Outra estratégia boa é pensar bem na hora de fazer o design - ou convencer o designer a pensar bem. Será que realmente precisamos daquele monte de imagens na página? Será que aquele ícone precisa ser truecolor ou podia ser salvo em grayscale?

Um ponto importante é que o crescimento e adoção de CSS3 tem trazido novas possibilidades de design em CSS puro que antes só eram possíveis com imagens. Bordas redondas, gradientes, sombras, etc. Usando CSS, conseguimos o mesmo efeito evitando colocar mais imagens na página.

Pense bem no seu design e na forma como o codifica. Ele pode ser um fator de peso na performance da sua página.

Otimizações lossless

A otimização mais simples e eficaz com imagens é o que chamamos de compressão *lossless*. É diminuir o tamanho da imagem sem perder absolutamente nada na qualidade.

Isso é possível porque os formatos da Web (JPEG, PNG, GIF) em geral guardam em seus arquivos mais informações do que as necessárias para renderizar a imagem. Uma foto JPEG por exemplo tem diversos metadados embutidos como horário da foto e até coordenadas de GPS, se a câmera suportar essa funcionalidade. Ou ainda PNG exportados pelos editores como Photoshop levam diversos metadados extra e muitas vezes até uma miniatura da imagem embutida no mesmo arquivo.

Tudo isso pode ser interessante para se organizar os arquivos pessoais, montar seus álbuns etc. Mas são completamente irrelevantes para a renderização na página.

Podemos usar ferramentas automáticas para retirar esses bytes extra de imagens sem perda alguma de qualidade. A ferramenta mais famosa é o **Smush.it** do Yahoo:

<http://smush.it>

E o próprio JPEGMini já faz isso também para nossas fotos JPEG.

E OFFLINE?

Usar o Smush.it é uma das formas mais simples e eficientes de se otimizar as imagens. Caso você queira usar algo direto no computador, recomendamos dois programas com interfaces gráficas locais:

MAC: <http://imageoptim.pornel.net/> Windows: <http://luci.criosweb.ro/riot/>

Se o objetivo é automatizar a otimização, você provavelmente vai querer ferramentas de linha de comando. E há várias delas disponíveis (inclusive as usadas pelo Smush.it). Procure por: optipng, pngout, pngcrush, advpng, jpegoptim, gifsicle.

12.6 DIMINUIR O NÚMERO DE REQUESTS

Todas as práticas que vimos até agora tinham como objetivo diminuir o tamanho das requisições, o volume do tráfego de dados. Há ainda outro ponto que levantamos sobre aplicações distribuídas: *diminuir o número total de requests*.

No YSlow, na aba *Components*, você pode ver todos os componentes de página que acabou de fazer. Cada imagem, arquivo JavaScript, CSS e até vídeos e Flash são requisições feitas ao servidor. Isso sem pensar no próprio HTML da página e em possíveis requests extras numa aplicação Ajax.

Cada requisição envolve uma chamada para o servidor o que gera um overhead bastante grande. A maior parte do tempo de um request é gasto em tarefas de rede (DNS, SSL, TCP/IP etc). Se você já otimizou o tamanho dos requests, verá que uma pequena parte apenas é gasta no download dos bytes.

Fora o próprio gargalo de rede, existe uma limitação no número de requisições que um navegador faz simultaneamente a um mesmo servidor. Esse número varia bastante, mas chega a ser bem baixo em navegadores antigos (apenas 2 conexões). Hoje, nos navegadores mais modernos, gira em torno de 6 a 8 conexões. Parece um número alto - e realmente foi uma grande evolução -, mas se você começar a contar todos os arquivos externos que está usando, vai ver que há chances de uma página mediana fazer dezenas de requests.

O HTTPArchive reporta uma média de mais de 80 requests sendo feitos na página:

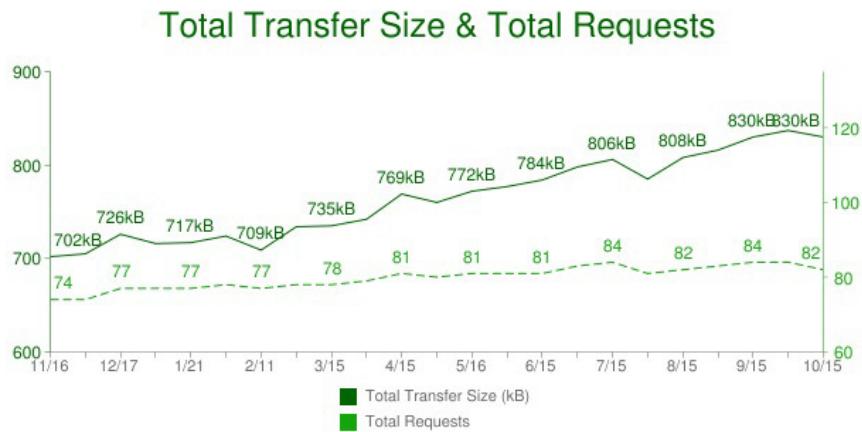


Figura 12.4: Número e volume médio de requests

Ou seja, mesmo com 8 conexões simultâneas, o volume de requests é bem maior o que vai atrasar o carregamento total da página. Em alguns casos (como arquivo JS), o navegador fica travado até que todos os downloads terminem.

Diminuir o número de requisições é essencial.

12.7 JUNTAR ARQUIVOS CSS E JS

No caso de arquivos CSS e JavaScript, a boa prática é juntar diversos arquivos em um número menor para minimizar requisições.

Ao usar jQuery, por exemplo, importamos a biblioteca em si e, em geral, criamos um script a mais da nossa aplicação que vai usá-lo. É muito comum também (como veremos no curso WD-47) usarmos diversos plugins com o jQuery para fazer várias tarefas avançadas. São todos requests a mais que vão sobrecarregar nossa página.

Se vamos usar, por exemplo, jQuery com 3 plugins e mais 2 arquivos da aplicação, poderíamos simplesmente juntar todos em 1 ou 2 arquivos. Novamente, a boa prática não é escrever um código todo misturado de uma vez só em um arquivo de milhares de linhas. Faça seu código com boas práticas e execute alguma rotina que junte automaticamente esses arquivos pra você antes de subir no servidor.

O próprio jQuery faz isso. Usamos um arquivo chamado *jquery.js* mas se você for olhar o projeto oficial, verá que esse é apenas um arquivo gerado a partir de outros 22 arquivos separados, que foram criados de maneira independente para melhor organizar e encapsular as funcionalidades do projeto.

A mesma dica vale para arquivos CSS. Você pode organizar seu código em arquivos diferentes (por exemplo, um com tipografia, outro com estrutura de layout e outro com estilos visuais), porém, na hora de subir a aplicação no ar, a boa prática é diminuir o número de arquivos e juntar quantos puder.

12.8 IMAGE SPRITES

Juntar arquivos texto como JS e CSS é muito fácil, basta copiar os conteúdos seguidamente. E as imagens? Já vimos que elas costumam ser o componente mais pesado das páginas. Em designs complexos, são responsáveis também por um grande número de requisições.

Conseguimos juntar imagens?

É possível, e essa técnica é chamada de **sprites**, que é um processo muito mais complicado. Juntar imagens consiste em criar um arquivo só e posicionar diversas imagens dentro. Depois, usando CSS, "recorta-se" os pedaços da imagem que devem ser mostradas em cada parte da página.

Diversos grandes sites usam essa técnica. O Google por exemplo, tem uma única imagem em sua home:



Criar a imagem é o primeiro passo, e um dos mais chatos. Há algumas ferramentas que tentam automatizar (como o [sprite.me](#)), mas em geral o processo é bastante manual. Abre-se um editor de imagens e se posicionam as imagens para obter o resultado final.

E, principalmente, nesse processo de juntar as imagens no editor, devemos prestar bastante atenção no posicionamento que precisa ser anotado precisamente. A posição (X,Y) de cada imagem dentro da sprite, além do tamanho (width, height) de cada uma. Essas informações serão valiosas para escrever o

CSS.

Utilizamos a propriedade background do CSS do elemento como na técnica de Image Replacement, mas é preciso especificar a posição da imagem em relação ao ponto inicial do elemento.

Para saber mais - Data URIs

Ainda pensando em minimizar o número de requisições para imagens, há uma outra técnica conhecida como **Data URIs**. A ideia é que você pode embutir conteúdos binários (como imagens) dentro de arquivos textos (como páginas HTML ou arquivos CSS). Basta converter os bytes da imagem para uma string comum que segue o formato de codificação chamado *Base 64*.

Esse processo de conversão é feito por algum programa que converte para base64 ou diretamente no servidor por meio de código Java, PHP etc. Há alguns serviços online que ajudam nessa tarefa, também.

Imagine que queremos colocar o seguinte logo da Caelum em nossa página:



Com HTML normal, faríamos:

```

```

Isso vai causar uma requisição para o arquivo `logo-caelum.png`. Usando data URIs, vamos embutir o código base64 da imagem direto na tag HTML:

```


O resultado é assustador e parece até pior com relação a tamanho, mas lembre-se de que estamos economizando os bytes da imagem. E essa string dentro do HTML, após o GZIP, costuma ter um tamanho muito próximo ao que seria a imagem binária original.

Você pode usar essa técnica também em arquivos CSS, dentro de `background-image` por exemplo.

O maior problema dessa técnica é que ela não é suportada em todos os navegadores. Todos os mais modernos suportam, já o IE6 e IE7 e mesmo o IE8 possui algumas limitações. Além disso, o processo de geração dessa string base64 costuma exigir um pouco de conhecimento de programação no servidor.

## 12.9 PARA SABER MAIS

A Caelum tem vários posts no Blog sobre o assunto, sendo o principal:

<http://blog.caelum.com.br/por-uma-web-mais-rapida-26-tecnicas-de-otimizacao-de-sites/>

## 12.10 EXERCÍCIOS: OTIMIZAÇÕES WEB

1. Rode as ferramentas de análise do PageSpeed Online e do WebPageTest:

<https://developers.google.com/pagespeed/>

<http://webpagetest.org/>

Analise o resultado. Veja possíveis pontos de melhoria.

2. Nossos banners principais na home são fotografias imensas, mas estão em PNG. O formato ideal para eles é JPG, que traz uma qualidade satisfatória com muito menos kbytes.

Faça a conversão dos banners de PNG para JPG e compare os resultados.

3. Comprima as imagens do projeto para economizarmos no tamanho.

Os PNGs e GIFs podem ser comprimidos sem perdas no Smush.it do Yahoo: <http://smushit.com>

Um serviço alternativo é o <http://kraken.io>

Os JPEGs podem ser comprimidos diminuindo sua qualidade e o seu tamanho sem muita perda de qualidade. Uma ótima ferramenta pra isso é o JPEGMini: <http://jpegmini.com>

4. Criamos vários arquivos CSS e JavaScript na nossa home. Podemos juntá-los para obter um ganho de performance.

Nos CSS, temos dois arquivos: `estilos.css` e `mobile.css`. Podemos juntá-los num único arquivo. (dica: o `mobile.css` é importado com media query; para juntá-lo no arquivo principal, você precisará escrever a media query corretamente dentro do CSS)

Nos JavaScripts, podemos, por exemplo, juntar o jQuery e o jQuery UI num único arquivo.

5. Comprima os arquivos CSS e JavaScript.
6. Depois dessas otimizações, teste novamente no PageSpeed e WebPageTest. Houve alguma melhora?

# APÊNDICE - LESS

*"Não podemos solucionar problemas usando a mesma forma de raciocínio que usamos quando os criamos." -- Albert Einstein*

LESS é uma linguagem baseada em CSS (mesma ideia, sintaxe familiar) com recursos que fazem falta no CSS em algumas situações. É também chamado de pré-processador pois, na verdade, é usado para gerar um arquivo CSS no final.

Alguns dos recursos apresentados pela linguagem são variáveis, suporte a operações aritméticas, sintaxe mais compacta para representar hierarquias e mixins.

## 13.1 VARIÁVEIS

Você já precisou usar a mesma cor no CSS em vários pontos diferentes? Um título e um botão com mesma cor, por exemplo? O CSS tem uma solução pra evitar copiar e colar, que seria o uso de classes. Mas, muitas vezes, usar essa mesma classe em tantas tags diferentes não é uma boa ideia.

Programadores estão acostumados com variáveis pra isso, mas o CSS não tem nada parecido. Mas o LESS sim!

```
@corprincipal: #BADA55;

#titulo {
 font-size: 2em;
 color: @corprincipal;
}

button {
 background-color: @corprincipal;
 color: white;
}
```

Repare no uso da `@corprincipal` que não é CSS puro, mas tem uma sintaxe bem parecida e familiar. Depois de compilado, o LESS vira esse CSS:

```
#titulo {
 font-size: 2em;
 color: #BADA55;
}

button {
 background-color: #BADA55;
 color: white;
```

---

```
}
```

## 13.2 CONTAS

Sabe quando você tem aquele elemento principal com 960px mas que precisa de um padding de 35px e duas colunas lá dentro de tamanhos iguais mas deixando 20px entre elas? Qual o tamanho de cada coluna mesmo? 435px. Aí você coloca no CSS:

```
.container {
 padding: 35px;
 width: 960px;
}
.coluna {
 width: 435px;
}
```

E quando alguém mudar o tamanho do padding, você torce pra lembrarem de refazer a conta da coluna - que, aliás, seria  $(960\text{px} - 35\text{px} * 2 - 20\text{px}) / 2 = 435\text{px}$ . No LESS, você pode fazer a conta direito na propriedade e o resultado final é calculado:

```
.coluna {
 width: (960px - 35px * 2 - 20px) / 2;
}
```

Melhor ainda, junte com as variáveis que vimos antes e você nem copia e cola valores!

```
@total: 960px;
@respiro: 35px;
@espacamento: 20px;

.container {
 padding: @respiro;
 width: @total;
}
.coluna {
 width: (@total - @respiro * 2 - @espacamento) / 2;
}
```

E dá pra fazer contas de tudo que é tipo, até com cores!

## 13.3 HIERARQUIA

Você tem um `#topo` com um título `h1` dentro e uma lista `ul` de links. E quer estilizar todos esses elementos. Algo assim:

```
#topo {
 width: 100%;
}

#topo h1 {
 font-size: 2em;
}

#topo ul {

```

---

```
}
```

E se você precisar mudar o id do `#topo`? Ou trocá-lo por um `<header>` semântico? Tem que mexer em 3 lugares (e torcer pra ninguém ter usado em outro canto). Fora que o código fica desorganizado já que essas três regras não necessariamente precisam estar agrupadas no arquivo e podiam estar espalhadas por aí, apesar de serem todos sobre nosso cabeçalho.

No LESS, podemos escrever regras de maneira hierárquica, uma dentro da outra, e ele gera os seletores de parent. O mesmo CSS acima podia ser no LESS:

```
#topo {
 width: 100%;

 h1 {
 font-size: 2em;
 }

 ul {
 margin-left: 10px;
 }
}
```

Podemos usar vários níveis de hierarquia (mas não abuse!), deixando nosso código mais estruturado, flexível e legível.

## 13.4 FUNÇÕES DE CORES E PALHETAS AUTOMÁTICAS

Provavelmente você já viu algum design que tem uma cor base principal e algumas cores secundárias combinando. Talvez uma versão mais light dessa cor base é usada como fundo e uma cor mais saturada no botão.

Você então pega o código de cada cor no Photoshop e coloca no CSS. E, se precisar mudar a cor, deve gerar as outras secundárias, certo? No LESS, você pode usar funções pra gerar essas cores:

```
@corbase: #BADA55;

body {
 background: lighten(@corbase, 20%);
}
h1 {
 color: @corbase;
}
button {
 background: saturate(@corbase, 10%);
}
```

Vai gerar cores 20% mais lights e 10% mais saturadas:

```
body {
 background:#dceca9;
}
h1 {
 color:#bada55;
}
```

```
button {
 background:#bfe44b;
}
```

Você ainda tem: `darken` , `desaturate` , `fadein` , `fadeout` , `spin` , `mix` e até funções matemáticas como `round` .

## 13.5 REAPROVEITAMENTO COM MIXINS

Uma das coisas mais legais do LESS é sua capacidade de criar as próprias funções, que ele chama de **mixins**. É útil quando você tem que repetir a mesma coisa várias vezes, como nas propriedades CSS3 que precisam de prefixos, tipo uma borda redonda.

Você pode definir um mixin recebendo argumento o tamanho da borda e cuspindo as versões pros diversos navegadores:

```
.arredonda(@raio: 5px) {
 -webkit-border-radius: @raio;
 -moz-border-radius: @raio;
 border-radius: @raio;
}
```

Parece uma classe CSS mas ele recebe uma variável como parâmetro (que pode ter um valor default também). E você pode usar esse mixin facilmente:

```
.painel {
 .arredonda(10px);
}
.container {
 .arredonda;
 width: 345px;
}
```

Isso gera o CSS:

```
.painel{
 -webkit-border-radius:10px;
 -moz-border-radius:10px;
 border-radius:10px;
}
.container{
 -webkit-border-radius:5px;
 -moz-border-radius:5px;
 border-radius:5px;

 width:345px;
}
```

As possibilidades são infinitas! Pense num mixin pra te ajudar a gerar aqueles gradientes CSS3 chatos ou um mixin próprio `.botaoBonito` que gera botões legais só recebendo uma cor base e um tamanho.

## 13.6 EXECUTANDO O LESS

---

No site do LESS, você pode baixar a versão *standalone* dele. Você pode rodá-lo apenas incluindo um JavaScript na página que faz o parsing dos arquivos `.less` quando ela carrega.

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

Com isso, podemos incluir diretamente nosso arquivo `.less` usando uma tag `<link>`, colocada **antes** da tag que carrega o `less.js`:

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

Para melhor performance, o ideal seria gerar o CSS antes usando o compilador. Há uma versão em linha de comando usando NodeJS, mas como é JavaScript, você pode rodá-lo em qualquer canto - até no Java com Rhino.

Há também programas visuais pra instalar, como o LESS.app e dá pra testar código rapidamente online mesmo no LessTester.com.

No Windows, você pode usar o **WinLESS** que é um compilador com interface gráfica e bem fácil de usar: <http://winless.org/>

## 13.7 PARA SABER MAIS: RECURSOS AVANÇADOS E ALTERNATIVAS

A linguagem LESS tem recursos ainda mais avançados. Dá pra fazer mixins mais complicados com uma espécie de if/else por exemplo e até usar pattern matching. Você encontra todos os detalhes na documentação oficial.

Além do LESS, existem outros pré-processadores CSS no mercado. O Sass (<http://sass-lang.com/>) é muito famoso no mundo Ruby e tem zilhões de funções, tornando-o mais poderoso que o LESS mas mais complexo também. Há também o Stylus (<http://learnboost.github.io/stylus/>), que simplifica ainda mais a sintaxe.

## 13.8 EXERCÍCIOS: LESS

- Atualmente, nosso arquivo `estilos.css` possui várias regras que usam o seletor de hierarquia (espaço). Essas regras podem ser escritas de forma mais compacta com LESS. Então vamos usar LESS para escrevê-lo. Neste exercício, vamos usar a versão Javascript do LESS, que transforma nosso código em CSS dentro do navegador.

Crie uma nova pasta no seu projeto chamada `less` e copie o `estilos.css` para lá. Em seguida, renomeie-o para `estilos.less`. Por fim, altere o `index.html` para usar o `estilos.less`. Troque a linha

```
<link rel="stylesheet" href="css/estilos.css">
```

por

```
<link rel="stylesheet/less" href="less/estilos.less">
```

E inclua a seguinte linha antes de fechar a tag `body` para carregar o pré-processador LESS:

```
<script src="js/less.js"></script>
```

Teste a página no navegador. Ela deve continuar com a mesma aparência de antes.

2. Vamos começar a migrar nosso código CSS para LESS. Um seletor bastante repetido no código é `.painel`. Agrupe todas as regras que usam esse seletor num único seletor no LESS. O código final deve ficar parecido com este:

```
.painel {
 /* regras que estavam em .painel {...} */
 li {
 /* regras que estavam em .painel li {...} */

 &:hover {
 /* regras que estavam em .painel li:hover {...} */

 &:nth-child(2n) {
 /* regras que estavam em .painel li:nth-child(2n):hover {...} */
 }
 }
 }

 h2 {
 /* regras que estavam em .painel h2 {...} */

 &:before {
 /* regras que estavam em .painel h2:before {...} */
 }
 }

 a {
 /* regras que estavam em .painel a {...} */
 }

.mostra-mais {
 /* regras que estavam em .painel .mostra-mais {...} */
}
```

Teste novamente no navegador. A página não deve mudar, mas veja que o código fica mais organizado e curto!

3. A cor `#333333` (cinza escuro) se repete algumas vezes no nosso estilo. Vamos isolá-la numa variável para facilitar a manutenção:

```
@escuro: #333333;
```

Procure os lugares que usam a cor e use a variável no lugar. Por exemplo:

```
body {
 color: @escuro;
 /* outras regras */
}
.mostra-mais {
```

```

background: @escuro;
/* outras regras */
}

```

Experimente trocar o valor da variável e veja o efeito: para mudar a cor de vários elementos da página agora basta mexer num único lugar!

- Vamos deixar o nosso código de transições mais limpo isolando os prefixos num único lugar. Para isso, vamos criar um *mixin*:

```

.transicao(@propriedades, @tempo) {
 -webkit-transition: @propriedades @tempo;
 -moz-transition: @propriedades @tempo;
 -ms-transition: @propriedades @tempo;
 -o-transition: @propriedades @tempo;
 transition: @propriedades @tempo;
}

```

Agora altere o código que escala e rotaciona as fotos dos produtos quando o mouse passa em cima: apague as declarações `transition` e coloque no lugar

```
.transicao(all, 0.7s);
```

Faça o mesmo com os gradientes dos painéis:

```

.gradiente(@cor1, @cor2) {
 background: @cor1; /* Navegadores antigos */
 background: -moz-linear-gradient(top,
 @cor1 0%, @cor2 100%); /* FF3.6+ */
 background: -webkit-gradient(linear, left top, left bottom,
 color-stop(0%, @cor1),
 color-stop(100%, @cor2)); /* Chrome,Safari4+ */
 background: -webkit-linear-gradient(top,
 @cor1 0%, @cor2 100%); /* Chrome10+,Safari5.1+ */
 background: -o-linear-gradient(top,
 @cor1 0%, @cor2 100%); /* Opera 11.10+ */
 background: -ms-linear-gradient(top, @cor1 0%, @cor2 100%); /* IE10+ */
 background: linear-gradient(to bottom, @cor1 0%, @cor2 100%); /* W3C */
 filter: progid:DXImageTransform.Microsoft.gradient(
 startColorstr='@cor1',
 endColorstr='@cor2',
 GradientType=0); /* IE6-9 */
}

.novidades {
 .gradiente(#f5dcfc, #f4bebe);
}

.mais-vendidos {
 .gradiente(#dcdcf5, #bebef4);
}

```

**Dica:** você pode até usar o gerador de gradientes do ColorZilla (<http://www.colorzilla.com/gradient-editor/>) para gerar o código desse mixin.

- (opcional) Podemos melhorar nosso *mixin* de gradiente fazendo uma versão que só recebe uma cor e calcula a segunda automaticamente, fazendo ela ser 10% mais escura que a cor dada. Podemos ainda

fazer com que essa nova versão já aproveite o *mixin* já existente, passando a segunda cor calculada para ele:

```
.gradiente-automatico(@cor1) {
 .gradiente(@cor1, darken(@cor1, 10%));
}
```

Faça o teste nos gradientes dos painéis: use essa versão do *mixin* e veja o efeito ser aplicado automaticamente.

# APÊNDICE - SUBINDO SUA APLICAÇÃO NO CLOUD

*"Perder tempo em aprender coisas que não interessam, priva-nos de descobrir coisas interessantes" -- Carlos Drummond de Andrade*

## 14.1 COMO ESCOLHER UM PROVEDOR

Existem muitos servidores com suporte a PHP e MySQL no mercado, tanto nacionais quanto internacionais. Na hora de escolher um provedor de hospedagem, leve em conta preço, qualidade do serviço, atendimento, suporte e reputação da empresa.

Há vários serviços disponíveis, mas em geral você deve escolher entre duas opções. Há os planos clássicos de hospedagem onde você paga um valor fixo por mês e tem direito a usar os recursos de uma máquina compartilhada com outros usuários. Costumam ter um valor mais baixo mas limitações técnicas caso seu site tenha um pouco mais de acessos.

E há planos de **cloud computing** onde o objetivo é não ter restrições técnicas com relação ao volume de acessos. Sua aplicação pode começar pequena e crescer indefinidamente sem problemas, em um ambiente escalável que cresce elasticamente conforme suas necessidades. Nesse cenário, você paga por quanto usar dos recursos em cada mês, e não uma mensalidade fixa. Costuma ter um valor mais alto que hospedagens clássicas, mas traz um ambiente bem mais robusto.

## 14.2 O JELASTIC CLOUD LOCAWEB

Nesse capítulo, vamos usar a plataforma de Cloud da Locaweb, uma das principais empresas de TI do Brasil. Eles têm um produto chamado **Jelastic Cloud** que nos permite subir uma máquina com PHP e MySQL em instantes, e enviar um ZIP com nosso projeto para ser executado.

Saiba mais sobre o produto, inclusive os preços atuais, em:

<http://jelasticcloud.com.br/testegratis>

Há um trial de 14 dias que podemos usar para testar nosso projeto sem limitações.

## 14.3 CRIANDO A CONTA

Acesse o site do cloud Locaweb e localize a caixa de iniciar trial. Coloque seu email e clique no botão:



Em instantes você vai receber um email com seu login, senha e uma URL pra iniciar o uso da plataforma. Abra esse email e clique no link de ativação. Você deve cair no **Painel de Controle** principal. Clique no botão no topo que diz **Criar ambiente**.

Na janela de criação, selecione a aba **PHP** e o servidor **Apache** com o armazenamento **MySQL**. Dê também um nome a esse ambiente:



Dentro de alguns instantes seu ambiente será criado e você vai receber um email de confirmação. Esse email é importante por conter a **senha de acesso ao MySQL**.

## 14.4 IMPORTANDO DADOS NO MYSQL

Criado o ambiente, nós teremos acesso a um banco de dados MySQL vazio. No email que você recebeu, estão usuário, senha e endereço do banco de dados.

Acesse o MySQL pelo endereço dado. Ele é do formato [https://mysql-\[nomeprojeto\].jelasticlw.com.br/](https://mysql-[nomeprojeto].jelasticlw.com.br/)

Você vai ter acesso a um phpMyAdmin, igual usamos durante o curso. Coloque o usuário e senha que recebeu no email sobre o MySQL para acessar.



Dentro do phpMyAdmin, localize a aba superior **Importar**. Selecione a opção "Procurar no computador" e aponte o arquivo **dados.sql** que usamos no curso. No fim da página, clique em **Executar**.

Tudo dando certo, você deve ver o banco de dados **WD43** criado na coluna da esquerda, com nossa tabela **produtos** populada.

## 14.5 PREPARANDO O PROJETO

Volte pro código do seu projeto. Precisamos fazer uma alteração nos códigos que acessam o banco de dados no **index.php** e no **produto.php**.

Altere a linha que abre a conexão com `mysqli_connect` e passe a usar o endereço, usuário e senhas providos no email:

```
$conexao = mysqli_connect("mysql-[nomeprojeto].jelasticlw.com.br",
 "root", "[senha]", "WD43");
```

Cuidado que o endereço a ser usado não possui http na frente, é apenas o nome do servidor. E

cuidado com o ultimo argumento, o nome do banco, pra ser em maiúscula.

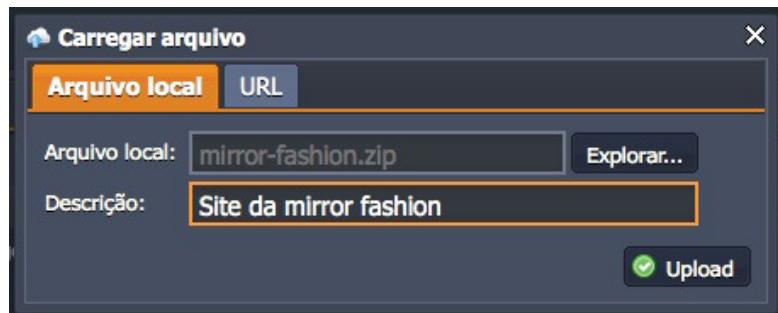
## 14.6 ENVIANDO O PROJETO E INICIALIZANDO SERVIDOR

Entre na pasta do projeto na sua máquina e crie um arquivo ZIP com todo seu conteúdo (todos os arquivos e subpastas de imagens, css, js etc). Dê o nome que quiser a esse arquivo.

De volta ao painel no **Jelastic Cloud**, localize o **Gerenciador de Instalação** no meio da página e clique em **Upload**:

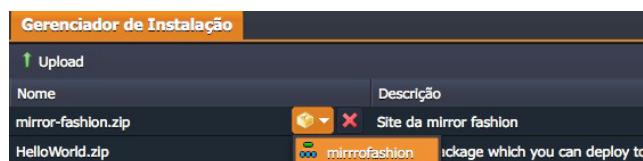


Indique o endereço do ZIP que criou com seu projeto e dê um nome a esse upload, como **Site da Mirror Fashion**:



Aguarde o upload finalizar e você verá a aplicação listada no Gerenciador de Instalação.

Na linha da aplicação, há um ícone que permite fazer a instalação do ZIP em algum ambiente criado. Clique nele e selecione o ambiente que criamos antes:



Uma janela se abre perguntando qual "contexto" você quer instalar. Isso quer dizer se queremos criar uma subpasta no servidor só pra esse projeto. Você pode deixar em branco pra criar na raiz do servidor mesmo. Clique em **Instalar**:



Aguarde alguns instantes até a instalação ser completada.

Aí é só acessar a aplicação no navegador usando o endereço cadastrado, que tem o formato:  
[http://\[nomeprojeto\].jelasticlw.com.br/](http://[nomeprojeto].jelasticlw.com.br/)

# APÊNDICE - MAIS INTEGRAÇÕES COM SERVIÇOS WEB

*"Pessoas viviam em fazendas, depois foram viver nas cidades. Agora todos nós vamos viver na Internet"*  
-- Sean Parker

## 15.1 BOTÃO DE CURTIR DO FACEBOOK

Boa parte dos sites atuais possui a funcionalidade de curtir do Facebook. É um botão simples mas integrado com a rede social que permite aos usuários curtirem a página atual e compartilhar essa informação em seus perfis.

É uma poderosa ferramenta de marketing, já que permite a recomendação pessoal de produtos e serviços de maneira viral.

Incluir essa funcionalidade no nosso site é bastante simples. O Facebook provê um código JavaScript e HTML para copiarmos na nossa página, onde podemos passar diversas configurações.



O botão é representado por um **div** vazio cheio de parâmetros:

```
<div class="fb-like" data-send="false" data-layout="box_count"
data-width="58" data-show-faces="false"></div>
```

Mas só esse div vazio, obviamente, não fará o botão aparecer. Precisamos também importar um arquivo JavaScript deles e rodá-lo:

```
<div id="fb-root"></div>
<script>(function(d, s, id) {
 var js, fjs = d.getElementsByTagName(s)[0];
 if (d.getElementById(id)) return;
 js = d.createElement(s); js.id = id;
 js.src = "http://connect.facebook.net/pt_BR/all.js#xfbml=1";
 fjs.parentNode.insertBefore(js, fjs);
}(document, 'script', 'facebook-jssdk'));</script>
```

Esse código é colocado uma vez só no final da página. Podemos ter vários botões na mesma página.

Melhor que digitar esse código é usar a documentação no site do Facebook que nos permite customizar o botão e já dá o código pronto para ser copiado:

<https://developers.facebook.com/docs/reference/plugins/like/>

## 15.2 EXERCÍCIOS: FACEBOOK

- Configure o script do Facebook na página de Produto. Antes de fechar a tag `body`, adicione:

```
<div id="fb-root"></div>
<script>(function(d, s, id) {
 var js, fjs = d.getElementsByTagName(s)[0];
 if (d.getElementById(id)) return;
 js = d.createElement(s); js.id = id;
 js.src = "http://connect.facebook.net/pt_BR/all.js#xfbml=1";
 fjs.parentNode.insertBefore(js, fjs);
}(document, 'script', 'facebook-jssdk'));</script>
```

- O botão em si é um `div` que devemos colocar na página onde queremos exibi-lo. Para nós, coloque logo após os elementos de título e preço do produto.

```
<div class="fb-like" data-href="http://www.mirrorfashion.net"
 data-send="false" data-layout="box_count"
 data-width="58" data-show-faces="false"></div>
```

Teste a página e veja o botão renderizado.



- (opcional) Posicione o botão melhor na página usando CSS.

## 15.3 PARA SABER MAIS: TWITTER

Você também pode acrescentar o botão de postar tweet. Basta seguir a documentação do Twitter:

<https://dev.twitter.com/docs/tweet-button>

O botão em si é um link que será transformado pelo script:

```
<a href="https://twitter.com/share"
 class="twitter-share-button" data-count="vertical">Tweet
```

E o script deve ser colocado no final da página também:

```

<script>
!function(d,s,id){var js,fjs=d.getElementsByTagName(s)[0];
if(!d.getElementById(id)){js=d.createElement(s);js.id=id;
js.src="http://platform.twitter.com/widgets.js";
fjs.parentNode.insertBefore(js,fjs);}}(document,"script","twitter-wjs");
</script>

```



## 15.4 PARA SABER MAIS: GOOGLE+

Podemos também colocar o botão de +1 do Google+. Para obter o código, basta ir em:

<http://www.google.com/webmasters/+1/button/>

O botão é um div vazio, parecido com o do Facebook:

```
<div class="g-plusone" data-annotation="inline"></div>
```

E o script deve ser colocado no final da página:

```

<script type="text/javascript">
window.__gcfg = {lang: 'pt-BR'};

(function() {
 var po = document.createElement('script');
 po.type = 'text/javascript';
 po.async = true;
 po.src = 'https://apis.google.com/js/plusone.js';
 var s = document.getElementsByTagName('script')[0];
 s.parentNode.insertBefore(po, s);
})();
</script>

```

Além disso, o Google+ possui metadados próprios diferentes daqueles do OpenGraph:

```

<!-- Update your html tag to include the itemscope and itemtype attributes -->
<html itemscope itemtype="http://schema.org/Product">

<!-- Add the following three tags inside head -->
<meta itemprop="name" content="Fuzzy Cardigan">
<meta itemprop="description" content="O Fuzzy Cardigan é fantástico para a
meia estação, quando o friozinho começa a chegar. Seu estilo
parisiense combina com o charme da estação.">
<meta itemprop="image"
 content="http://www.mirrorfashion.net/img/produtos/foto2-verde.png">

```

## 15.5 EXERCÍCIOS OPCIONAIS: TWITTER E GOOGLE+

1. Acrescente o botão do Twitter na página. Obtenha o código em: <https://dev.twitter.com/docs/tweet-button>

Ou use o botão vertical com:

```
<a href="https://twitter.com/share"
 class="twitter-share-button" data-count="vertical">Tweet
```

Além disso, no final da página, importe o script do Twitter:

```
<script>
!function(d,s,id){var js,fjs=d.getElementsByTagName(s)[0];
if(!d.getElementById(id)){js=d.createElement(s);js.id=id;
js.src="http://platform.twitter.com/widgets.js";
fjs.parentNode.insertBefore(js,fjs);}}(document,"script","twitter-wjs");
</script>
```

2. Coloque também o +1 do Google, cujo código está em:

<http://www.google.com/webmasters/+1/button/>