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.

Família Pelho

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.

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 alguns casos, os esforços de manutenção de um projeto que adota os novos padrões é similar ou comparável com a manutenção de um projeto que prevê compatibilidade total com navegadores já obsoletos como o Internet Explorer 7 e o Firefox 3.

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*.

CAPÍTULO 3

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.

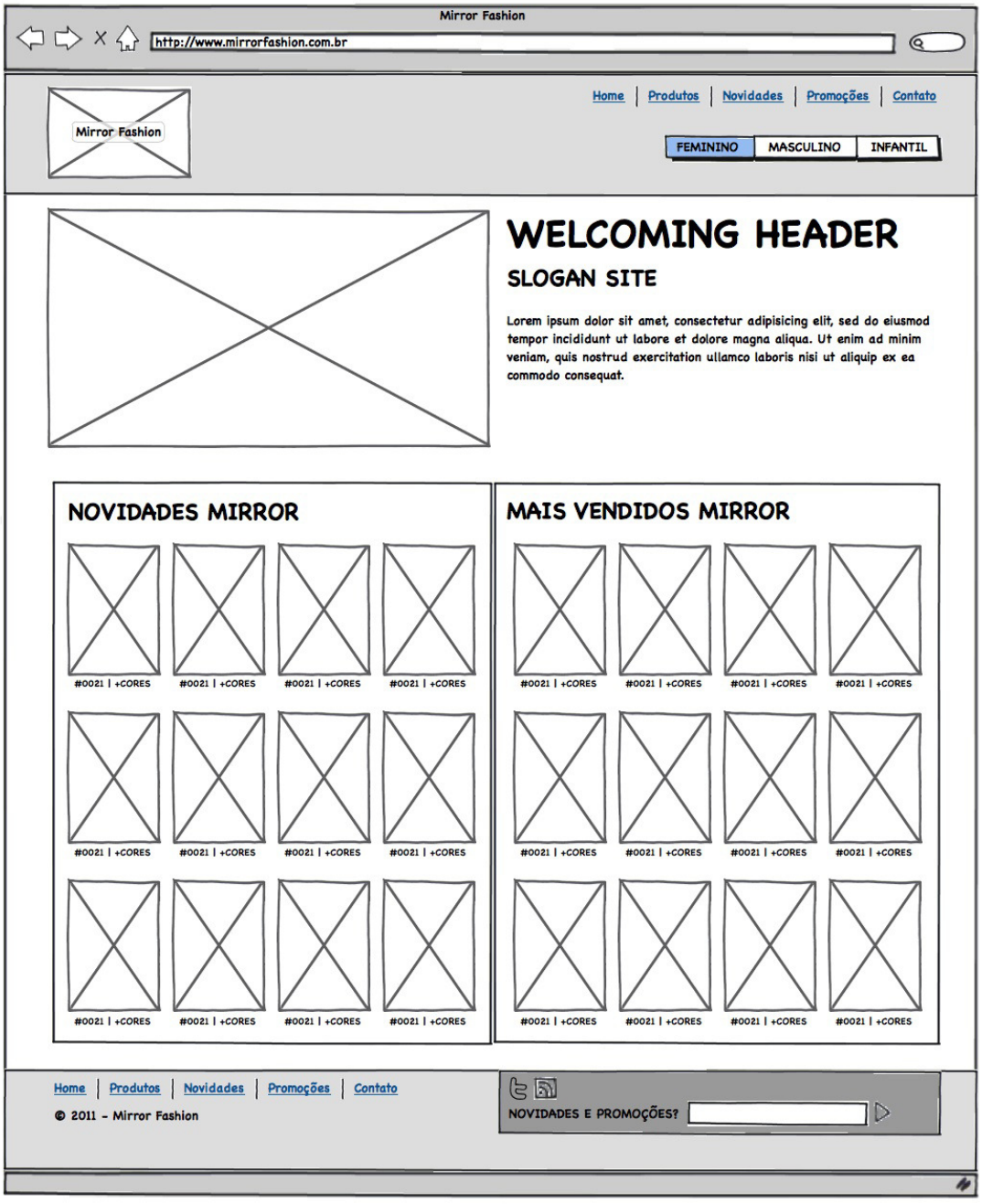
Quando detectada a necessidade do desenvolvimento de um site ou aplicação web, a empresa que tem essa necessidade deve passar todas as informações relevantes ao projeto para a empresa que vai executá-lo. A empresa responsável pelo seu desenvolvimento deve analisar muito bem essas informações e utilizar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

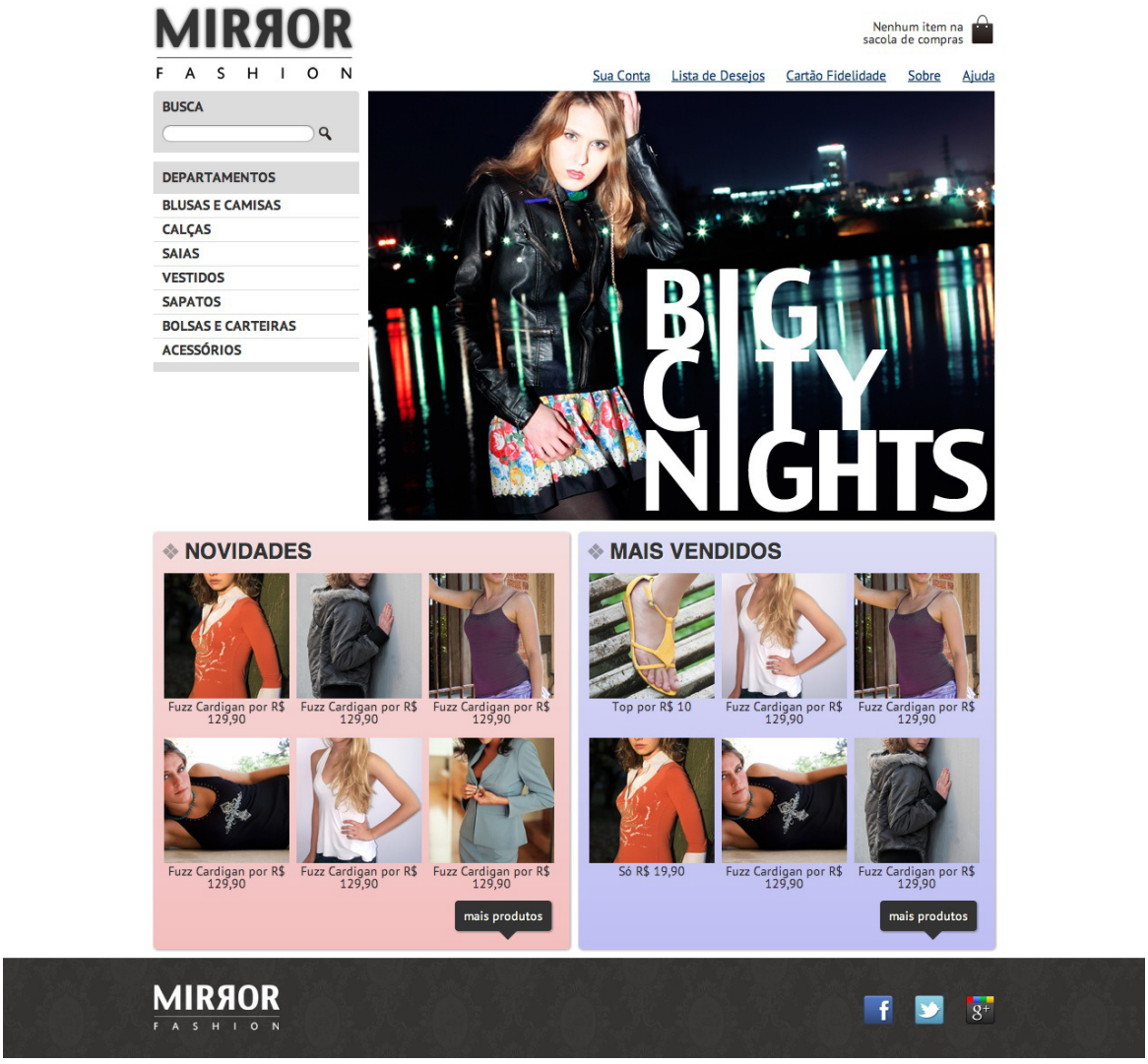


Figura 3.2: Design da Homepage

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>

  <div id="main">
    <!-- Conteúdo principal -->
  </div>

  <div id="destaques">
    <!-- Painéis com destaques -->
  </div>

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

```
  </footer>
</body>
```

Note que utilizamos o atributo `id` do HTML para identificar a `<div>` principal. O atributo `id` deve ser único em cada página, ou seja, só pode haver **um** elemento com o atributo `id="main"` . 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 - `<ul>` com `<li>` 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 `<img>` 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 `<img>` 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><img src="img/logo.png" alt="Mirror Fashion"></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 opcoes
    // 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>`, `<ul>`, `<li>`, etc. Crie links para as páginas do menu. E use `<h1>` para representar o título da página com o logo acessível.

   ```
   <body>
    <header>
      <h1><img src="img/logo.png" alt="Logo da Mirror Fashion"></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;
  padding-right: 35px;
  text-align: right;
  width: 140px;
}

.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>`, `<strong>` e `<em>` e a tag de marcação de atributos `<span>`.

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 `<li>` de uma `<ul>` tem por padrão o valor `block` para a propriedade `display`. Se quisermos os elementos na horizontal, basta alterarmos a propriedade `display` da `<li>` para `inline`:

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

```
}
```

3.10 EXERCÍCIOS: RESET E DISPLAY

1. 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.

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

   Vamos usar a propriedade `display` para mudar os `<li>` 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?

3. 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

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 (left ou right) e uma horizontal (top ou bottom) 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 (left ou right) e uma horizontal (top ou bottom) 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 antigos, 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 antigas.

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.

Saber inglês é muito importante em TI

Na **Alura Língua** você reforça e aprimora seu inglês! Usando a técnica *Spaced Repetitions* o aprendizado naturalmente **se adapta ao seu conhecimento**. Exercícios e vídeos interativos fazem com que você pratique em situações cotidianas. Além disso, todas as aulas possuem explicações gramaticais, para você entender completamente o que está aprendendo. Aprender inglês é fundamental para o profissional de tecnologia de sucesso!

Pratique seu inglês na Alura Língua.

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.

Capítulo 4

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.

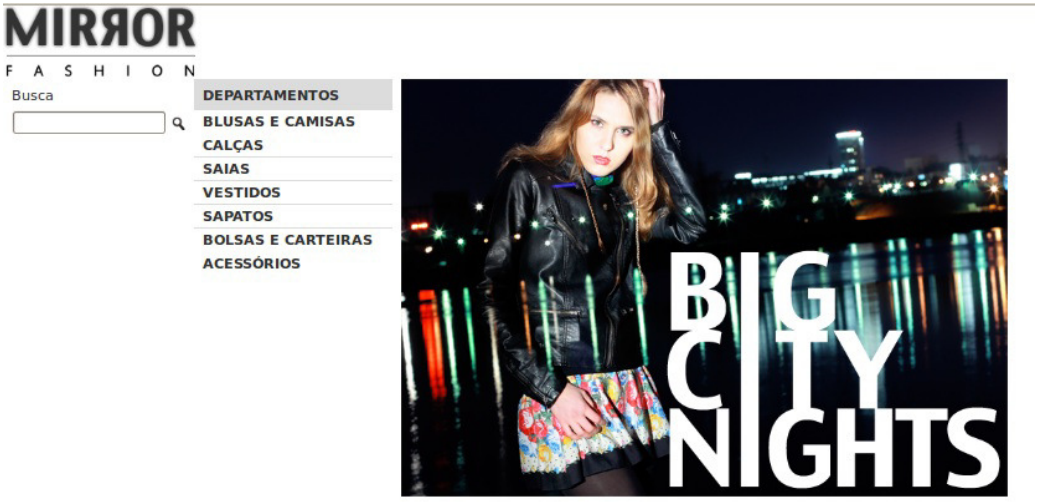
```
<form>
  <input type="search">
  <button>Buscar</button>
</form>
```

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 resolvermos este problema, precisaremos recorrer à propriedade **clear**.

A propriedade clear

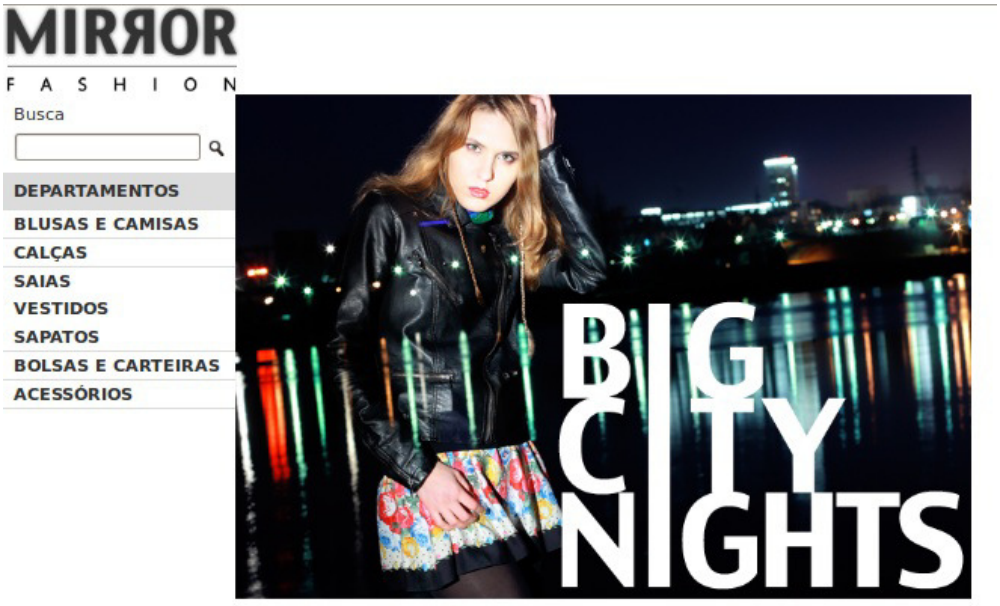
Existe uma propriedade que determina qual vai ser o comportamento de outros elementos que vêm ao redor daquele que a recebe e estão flutuando, e essa propriedade é a `clear`. A propriedade `clear` quer dizer "limpe o fluxo do documento ao meu lado" e pode receber os valores `left`, `right` ou

`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 `<del>` (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 navegabilidade. 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 sobrescrito 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>
    <img src="box-model.png" alt="box model">
</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 e, dentro dele, uma `section` para busca, outra para o menu e outra para o banner:

```
<div 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">
    <figure>
      <img src="img/destaque-home.png" alt="Promoção: Big City Night">
    </figure>
  </section>
  <!-- fim .banner-destaque -->

</div>
<!-- fim .container .destaque -->
```

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

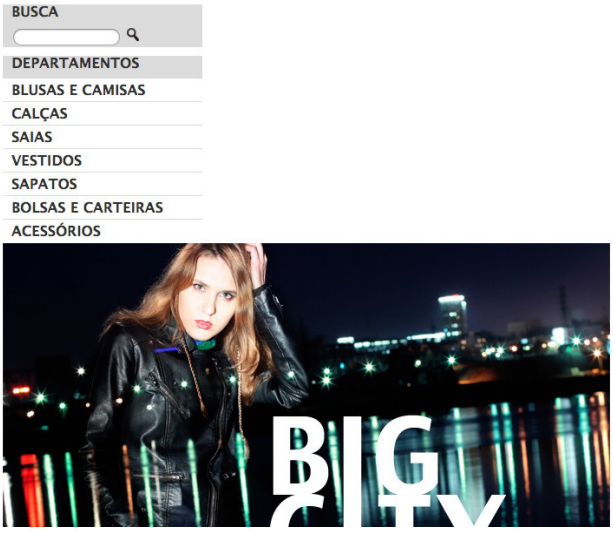
2. Aplique o estilo visual do design com CSS. Queremos um fundo cinza nas caixas de busca e no menu de departamentos. Além disso, o texto deve estar em negrito e apresentado em maiúsculas. Aplicaremos também algumas regras de tamanhos e margens.

```
.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;
}
```

3. 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 `<li>`. Já sabemos que por padrão uma `<li>` 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 `<li>`, mas agora os elementos `<li>` 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. Isto é, usando a propriedade `vertical-align` que, nesse caso, recebeu valor `top` .

Isso indica que, se tivermos vários produtos de tamanhos diferentes, eles vão se alinhar pelo topo.

4.9 EXERCÍCIOS: PAINÉIS FLUTUANTES

1. Vamos criar agora nossos painéis de novidades e mais vendidos. Crie um elemento `<div>` 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.

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

2. **Dentro da div criada 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 lista ordenada (`<ol>`) de produtos.

   Cada produto deve ser representado como um item na lista (`<li>`) 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>
 <ol>
   <!-- primeiro produto -->
   <li>
     <a href="produto.html">
       <figure>
         <img src="img/produtos/miniatura1.png" alt="miniatura1">
         <figcaption>Fuzz Cardigan por R$ 129,90</figcaption>
       </figure>
     </a>
   </li>
   <!-- fim do primeiro produto -->

   <!-- coloque mais produtos aqui! -->

 </ol>
</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.

3. Crie um segundo painel, para representar os produtos mais vendidos. Esse painel deve ficar **após** o fechamento do painel anterior, mas ainda **dentro** da div `paineis`.

   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>
 <ol>

   <!-- coloque vários produtos aqui -->

 </ol>
</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:

   - **div**: 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: #F5DCDC;
}

.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:

```
<img src="img/logo-rodape.png" alt="Logo da Mirror Fashion">
```

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 `<img>` , `<ul>` , `<li>` 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">
    <img src="img/logo-rodape.png" alt="Logo da Mirror Fashion">

    <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*.