

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.

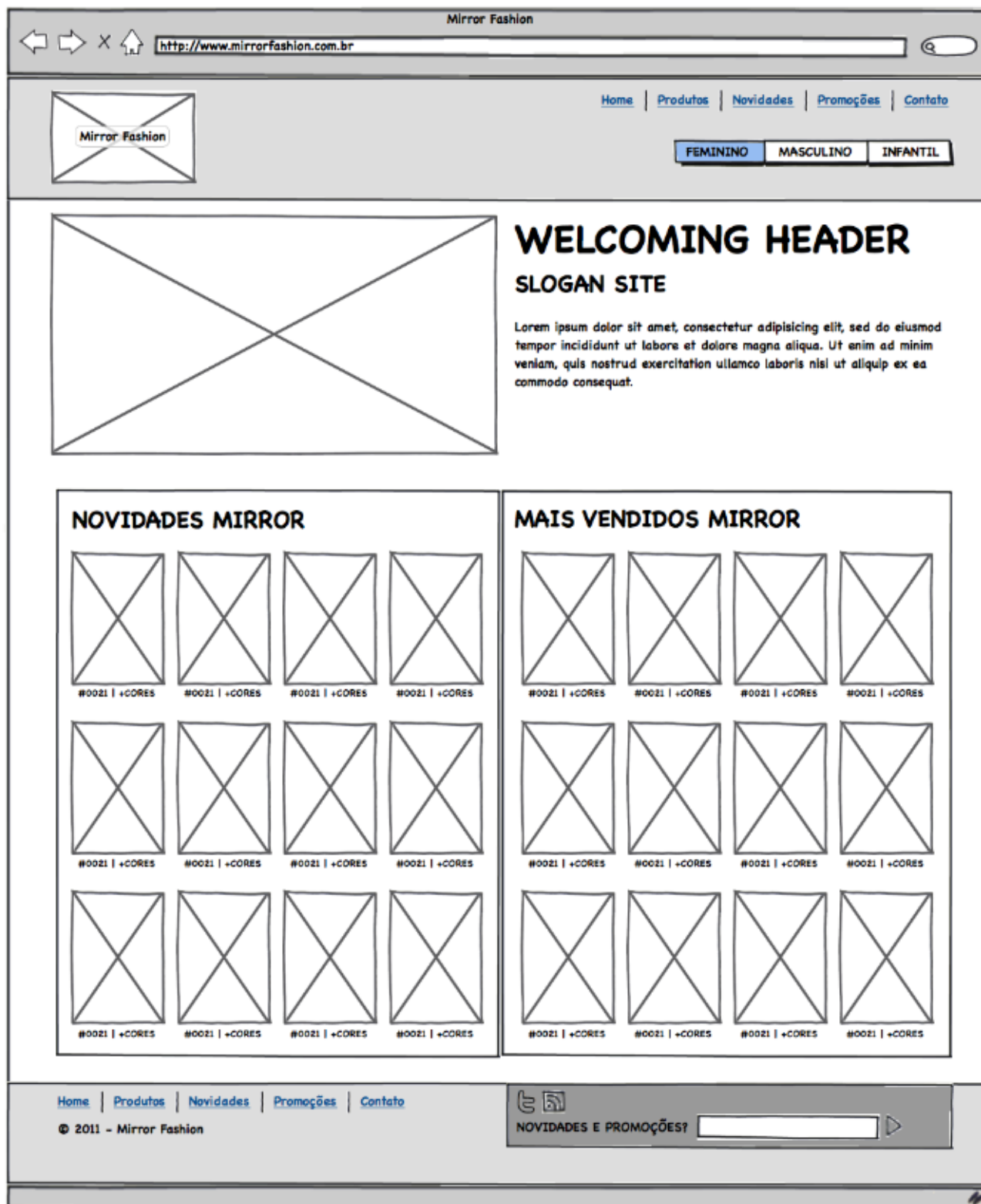
Um projeto de site ou aplicação web envolve muitas disciplinas em sua execução, pois são diversas características a serem analisadas e diversas as possibilidades apresentadas pela plataforma. Por exemplo, devemos conhecer muito bem as características do público alvo, pois ele define qual a melhor abordagem 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, exibir 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:

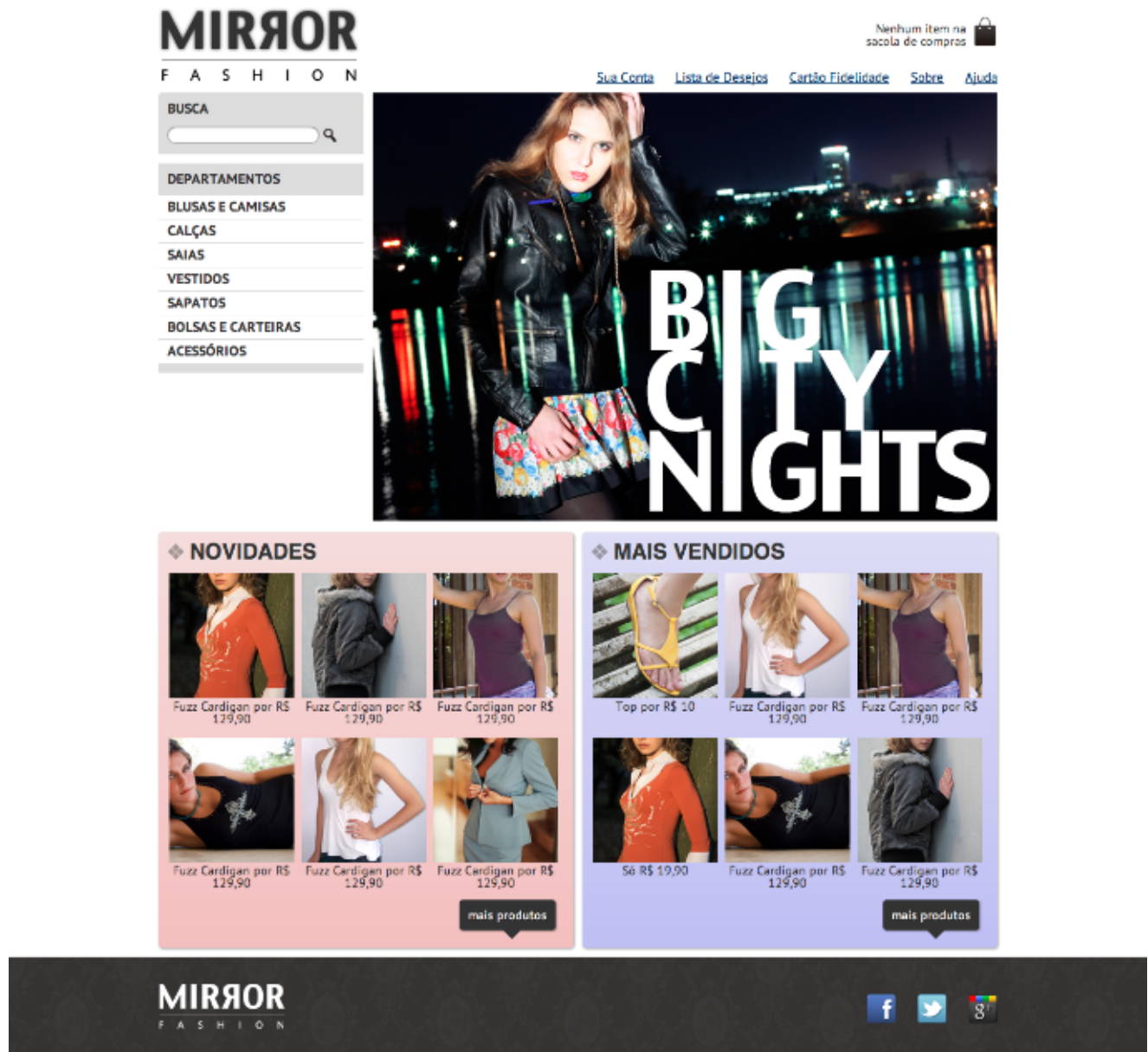


Figura 3.2: Design da Homepage

Já conhece os cursos online Alura?



A **Alura** oferece dezenas de **cursos online** em sua plataforma exclusiva de ensino que favorece o aprendizado com a **qualidade** reconhecida da Caelum. Você pode escolher um curso nas áreas de Java, Ruby, Web, Mobile, .NET e outros, com uma **assinatura** que dá acesso a todos os cursos.

[Conheça os cursos online Alura.](https://www.alura.com.br/)

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>

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

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

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

</body>
```

Note que utilizamos o atributo `id` do HTML para identificar a `<section>` 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`, `section` e `footer` – são tags novas do HTML5. Antigamente, numa situação parecida com essa, teríamos criado apenas três `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 diferença. 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 – isso é 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 será 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.

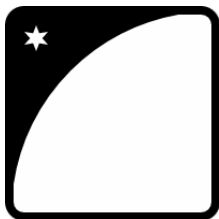
Quando o texto for lido para um cego, queremos essa mensagem lida. Quando o Google indexar, queremos que ele associe nossa página com Mirror Fashion e não com uma imagem "qualquer".

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 a o logo, colocando-o como elemento principal. E o ALT vai designar um conteúdo textual alternativo à imagem.

Você não está nessa página a toa



Você chegou aqui porque a Caelum é referência nacional em cursos de Java, Ruby, Agile, Mobile, Web e .NET.

Faça curso com **quem escreveu essa apostila**.

[Consulte as vantagens do curso *Desenvolvimento Web com HTML, CSS e JavaScript*.](#)

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, no uso de classes. O código é semelhante mas usa 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 pra deixar em aberto reaproveitar aquele elemento em mais de um ponto depois. Vamos fazer isso na sacola também:

```
<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 class "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 o CSS visual da página. Adicione também a tag `<link>` apontando para **css/estilos.css**:

```
<link rel="stylesheet" href="css/estilos.css">
```

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.

```
<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>
```

3. Já podemos até 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: url(../img/sacola.png) no-repeat 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.

MIRROR

F A S H I O N

Nenhum item na
sacola de compras 

- [Sua Conta](#)
- [Lista de Desejos](#)
- [Cartão Fidelidade](#)
- [Sobre](#)
- [Ajuda](#)

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.

Seus livros de tecnologia parecem do século passado?



Conheça a **Casa do Código**, uma **nova** editora, com autores de destaque no mercado, foco em **ebooks** (PDF, epub, mobi), preços **imbatíveis** e assuntos **atuais**.

Com a curadoria da **Caelum** e excelentes autores, é uma abordagem **diferente** para livros de tecnologia no Brasil.

Conheça os títulos e a nova proposta, você vai gostar.

[Casa do Código, livros para o programador.](#)

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

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`:

```
<link rel="stylesheet" href="css/reset.css">
```

Abra novamente a página no navegador. Percebe 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 `` 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. 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.

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 algum valor 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>
```

Estilo CSS

```
.quadrado {
  background: 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: 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`.

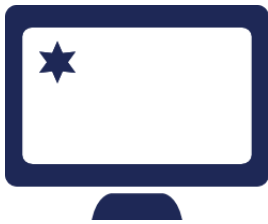
- **fixed**

- 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**

- o Uma configuração de posicionamento vertical (left ou right) e uma horizontal (top ou bottom) é obrigatória
- o 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
- o Seu tamanho não conta para calcular o tamanho do elemento pai

Agora é a melhor hora de aprender algo novo



Se você gosta de estudar essa apostila aberta da Caelum, certamente vai gostar dos novos **cursos online** que lançamos na plataforma **Alura**. Você estuda a qualquer momento com a **qualidade** Caelum.

[Conheça a Alura.](#)

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 **posicionado**.

```
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?

MIRROR
F A S H I O N

Nenhum item na
sacola de compras 

[Sua Conta](#)[Lista de Desejos](#)[Cartão Fidelidade](#)[Sobre](#)[Ajuda](#)

3.13 – EXERCÍCIOS OPCIONAIS

1. Aplique nosso novo cabeçalho também na página **sobre.html**.

CAPÍTULO ANTERIOR:

[Introdução a HTML e CSS](#)

PRÓXIMO CAPÍTULO:

[Mais HTML e CSS](#)

Você encontra a Caelum também em:

Blog Caelum

Cursos Online

Facebook

Newsletter

Casa do Código

Twitter