

Treinando :



**BOOTCAMP INTRODUÇÃO
A PROGRAMAÇÃO**

DIREITOS AUTORAIS ADQUIRIDOS PELA DIGICAD INFORMÁTICA E CONSULTORIA LTDA. PROIBIDA A REPRODUÇÃO DOS TEXTOS ORIGINAIS, MESMO PARCIAIS, POR QUALQUER PROCESSO, SEM PRÉVIA AUTORIZAÇÃO DA DIGICAD INFORMÁTICA E CONSULTORIA LTDA.

**Revisão: 00
10/12/2018**



INFORMAÇÕES AOS ALUNOS

Prezado aluno a Digicad tem um imenso prazer em contar com sua presença em nossos treinamentos. Esperamos que você tenha um excelente aproveitamento, replicando os conhecimentos na área profissional e pessoal.

Abaixo algumas informações e procedimentos importantes para um perfeito andamento de seu treinamento.

- Assinar a lista de presença em todas as aulas em que estiver presente, conferindo a grafia do nome, pois assim será impressa no certificado.
- Todo atraso e saída antecipada será anotada pelo instrutor na lista de presença.
- Desligar ou colocar em modo silencioso seu aparelho celular.
- Não comer ou beber em sala de aula.
- Haverá, em todas as aulas, um intervalo de 15 minutos para café.
- Para a certificação, o aluno deve ter frequência mínima de 80% e aproveitamento mínimo, na avaliação do instrutor.
- Para o curso de Autocad, além do item acima, será efetuada uma prova de avaliação prática.
- Cancelamento de aula em cursos VIP: conforme contrato os alunos de curso VIP devem desmarcar aulas com 12 horas de antecedência para as aulas diurnas e 6 horas de antecedência para as aulas noturnas, caso contrário as horas agendadas serão consideradas como ministradas.
- **Garantia de aprendizado:**
 - Repita o curso quantas vezes forem necessárias durante o prazo de 01 ano após o termino do curso.
 - Escolha uma nova turma que possua vaga disponível e matricule-se um dia antes do seu início;
 - Pague uma pequena taxa correspondente a 5% do valor atual do curso para cada nova turma;
 - Não há quantidade estabelecido para as repetições, desde que o mesmo curso, com a mesma versão do software, ainda esteja sendo oferecido pela escola;
 - Caso o curso tenha sido descontinuado, modificado ou substituído, consulte novas condições comerciais com o atendimento da escola e conheça nosso programa de reciclagem profissional, mais uma vantagem para o aluno Digicad.
- Você terá suporte permanente mesmo após o curso podendo tirar suas **dúvidas pertinentes ao treinamento** e obter outras informações pelo e-mail:

duvidas@digicad.com.br

Agradecemos sua confiança em nossos treinamentos.

Atenciosamente
Direção

1. INTRODUÇÃO À *HTML*

Segundo a World Wide Web Consortium (W3C), principal organização para a padronização da Rede Mundial de Computadores (*World Wide Web*, em inglês), a *Web* é baseada em três pilares:

- **URI** – *Uniform Resource Identifier* (Identificador Uniforme de Recurso), um conjunto de caracteres utilizados para identificar recursos da *Web*;
- **HTTP** – *HyperText Transfer Protocol* (Protocolo de Transferência de Hipertexto), um protocolo de comunicação para acessar estes recursos;
- **HTML** – *HyperText Markup Language* (Linguagem de Marcação de Hipertexto), uma linguagem de criação e edição de hipertexto, para que o usuário possa navegar por estes recursos.

Nosso foco será neste último pilar.

1.1 HISTÓRIA

A **HTML**, como dito anteriormente, é baseada em **Hipertexto**. Hipertexto é um conjunto de informações, podendo ser texto, imagem, vídeo, áudio, etc., onde seu acesso se dá através de referências específicas, chamadas de **hiperligações** (*hyperlinks*, em inglês). Estas hiperligações interconectam os vários conjuntos de informações espalhados pela Rede Mundial de Computadores, apresentando ao usuário a informação que os desenvolvedores criaram, e que o usuário procura.

Para que este compartilhamento de informações fosse facilmente criado, disponibilizado e acessado pelo mundo inteiro, foi criada uma linguagem universal, a **HTML**. Sua história inicia-se em 1980 com Tim Berners-Lee, engenheiro e cientista da computação britânico que criou um conjunto de ferramentas com o intuito de facilitar a comunicação e compartilhamento de pesquisas entre ele e seus colegas. Este conjunto de ferramentas, em combinação com a internet pública, criada alguns anos antes, e a criação do Mosaic, navegador desenvolvido por Marc Andeerssen na década de 90, deu início a um novo “universo” de possibilidades, despertando o interesse do mundo.

A partir daí, a **HTML** foi sendo utilizada por desenvolvedores, e aperfeiçoada com a criação de novas versões, mantendo sempre suas especificações graças ao W3C, com auxílio dos fabricantes de *softwares*. Em 1997, a W3C designou um grupo de trabalho focado no desenvolvimento de uma especificação da **HTML**, a **XHTML**, uma reformulação baseada em **XML** (*Extensible Markup Language* ou Linguagem de Marcação Estendida) que tornaria a **HTML** mais simples para ser processada e entendida. Entretanto, com a publicação da **HTML5** em 2014, a **XHTML** viu a maioria de sua utilização se tornar obsoleta. A versão mais atual da **HTML** é a versão 5.2, publicada em 2017.

1.2 SEMÂNTICA E SINTAXE

Semântica, em programação, pode ser entendida como a correta utilização dos códigos, de acordo com sua finalidade, enquanto a sintaxe se preocupa com a estrutura dos códigos. Para escrever estes códigos, é preciso um **editor de texto**. O Bloco de Notas, presente em sistemas operacionais Windows, da Microsoft, é uma simples ferramenta que pode ser utilizada para este fim. Existem vários outros editores no mercado, com os mais variados recursos, ficando a critério do desenvolvedor escolher aquele que lhe mais agrada e que mais se adapta às ferramentas utilizadas e seus objetivos. Nestes editores, o desenvolvedor escreverá as linhas de códigos, estruturadas através de marcadores, *tags* em inglês. Estas *tags* são escritas por caracteres específicos para cada ação, onde são colocados entre **parênteses angulares** (“<” e “>”), podendo, alguns destes, receber além do nome, atributos, valores, elementos e até mesmo outras *tags*. Veja o exemplo abaixo:

```
<!DOCTYPE html>
<html lang="pt-BR">
  <head>
    <meta charset="utf-8">
    <title>Bootcamp Introdutório Digicad</title>
  </head>
  <body>
    <p>Este é apenas o começo!</p>
    <!--Isto é um comentário-->
  </body>
</html>
```

Figura 1: Exemplo da estrutura HTML.

1.2.1 !DOCTYPE

O **<!DOCTYPE>**, presente na primeira linha do exemplo acima, não é uma *tag HTML*, mas sim uma instrução especial. Para o navegador exibir corretamente o conteúdo da página, é preciso informar qual o tipo do documento. Há vários comandos para a **!DOCTYPE** que eram utilizados nas versões anteriores da *HTML* e da *XHTML*. Atualmente, recomenda-se indicar para o navegador utilizar sempre a última versão da *HTML*, usando a instrução **<!DOCTYPE html>**, como no exemplo. A instrução <!DOCTYPE> deve ser sempre a primeira linha de código do documento. Isto porque a leitura das linhas de códigos segue uma ordem: de cima para baixo, da esquerda para a direita. Tanto esta instrução quanto todas as demais *tags* em *HTML* não são sensíveis a letra maiúscula ou minúscula, portanto em **<!DOCTYPE>** seriam aceitos **<!doctype>**, **<!Doctype>**, **<!DoCtYpE>**, etc. A exceção é em *XHTML*, apesar de não ser obrigatório o uso desta instrução em *XHTML*, caso decida utilizar, é obrigatório o uso em letras maiúsculas. Esta é apenas uma observação para que você esteja ciente que existem diferenças entre *HTML* e *XHTML*, pois *XHTML* não será o foco deste curso intensivo, mas sim o *HTML5*.

Agora que você já sabe o que significa a primeira linha de código da Figura 1, abra o seu Bloco de Notas e escreva a primeira linha, como no exemplo. Vá fazendo o mesmo com as demais linhas, conforme você for aprendendo o que cada uma delas significa. Escrever auxilia no processo de memorização.

Nota 1: A princípio utilizaremos o Bloco de Notas como editor de texto, pois este início é bem simples, porém você utilizará outros editores mais a frente.

Nota 2: É muito importante prestar atenção ao que você digita, pois um único caractere errado pode afetar o correto funcionamento de um documento inteiro!

1.2.2 HTML

Um documento *HTML* deve obrigatoriamente ter uma estrutura composta pelas *tags* `<html>`, `<head>` e `<body>`, nesta ordem. Poderão haver outras *tags* e elementos distribuídos em várias linhas de códigos entre estas três *tags* (como na Figura 1), entretanto esta ordem deve ser sempre respeitada.

```
<!DOCTYPE html>
<html lang="pt-BR">
  <head>
    <meta charset="utf-8">
    <title>Bootcamp Introdutório Digicad</title>
  </head>
  <body>
    <p>Este é apenas o começo!</p>
    <!-- Isto é um comentário-->
  </body>
</html>
```

Figura 2: Destaque para `<html>`.

A primeira *tag* é a `<html>`. Esta *tag* define o que será o documento *HTML*, portanto todos as *tags* e elementos *HTML* (com exceção da `!DOCTYPE`) devem vir **aninhados** (dentro) do intervalo da `<html>`. Como o navegador precisa saber onde começa e onde termina este intervalo, este elemento precisa ter uma *tag* de **abertura** e outra de **fechamento**. A *tag* de fechamento é idêntica à de abertura, porém com uma barra antes do nome. No caso da *tag* `<html>`, sua *tag* de fechamento fica `</html>`. Esquecer de fechar uma *tag* que requer fechamento é um erro muito comum, principalmente no início do aprendizado, portanto fique atento.

Observe que a *tag* `<html>` possui um **atributo** (**lang**). Atributos são informações extras que adicionamos dentro das *tags* para personalizá-las. Os **valores** destes atributos estão localizados após o sinal de igual e estão entre aspas. No exemplo da Figura 1, estamos dizendo que o documento *HTML* está (em sua totalidade ou maioria) no idioma “português do Brasil”. É recomendado que a *tag* `<html>` sempre venha acompanhada do atributo “lang”, pois algumas ferramentas de leitura ou tradução podem utilizar esta informação, o que facilita sua acessibilidade, mas este atributo não fica restrito a esta *tag*, podendo ser utilizado em qualquer outra parte do documento.

1.2.3 HEAD

Em seguida, dentro da *tag* `<html>`, vem a *tag* `<head>`. Esta *tag* define a área do cabeçalho do documento, contendo informações sobre o documento que são de interesse apenas do navegador, e não dos usuários do nosso site. Portanto, são informações que não serão visíveis na área do documento no navegador. Como é preciso delimitar a área do cabeçalho, esta *tag* também exige fechamento. Também é necessária a presença da *tag* `<title>` aninhada à *tag* `<head>`. A *tag* `<title>` define o título do documento, normalmente exibido na barra de título da janela ou aba do navegador.

```
<!DOCTYPE html>
<html lang="pt-BR">
  <head>
    <meta charset="utf-8">
    <title>Bootcamp Introdutório Digicad</title>
  </head>
  <body>
    <p>Este é apenas o começo!</p>
    <!--Isto é um comentário-->
  </body>
</html>
```

Figura 3: Destaque para `<head>`.

Outra *tag* muito comum de aparecer nesta área é a `<meta>`, que fornece **metadados** (ou metainformações) ao navegador, ou seja, define as propriedades da página (muito utilizados por sistemas de busca). No exemplo da Figura 1, esta *tag* vem acompanhada do atributo **charset**, que define o conjunto de caracteres que serão utilizados. O padrão para *HTML5* é o **UTF-8**, pois é o que possui a maior quantidade de caracteres e símbolos utilizados no mundo inteiro. Alguns editores de texto trazem a opção de você selecionar este tipo de codificação, portanto verifique para que não haja conflito. A *tag* `<link>` também é outra muito comum de se ver em `<head>`, e define conexões da página com outros arquivos externos. Mas esta *tag* deixaremos para ver mais a frente, em *CSS*.

Repare que na Figura 1, a *tag* `<head>` aparece embaixo e deslocada ligeiramente para a direita em relação a *tag* `<html>`. Esta técnica é chamada de **indentação**, onde adiciona-se espaços em branco (ou pressionando a tecla Tab) a *tags* aninhadas logo abaixo. Isso deixa claro a hierarquia das *tags* à quem lê as linhas de códigos: `<title>` está “dentro” de `<head>`, que está “dentro” de `<html>`.

1.2.4 BODY

Nossa terceira *tag* obrigatória em todos os documentos *HTML* é também a que recebe a maior parte do conteúdo. Trata-se do corpo do documento, ou *body*, em inglês. A *tag* `<body>` é o que delimitará a área que ficará à vista do usuário no navegador, e por este motivo também precisa de uma abertura e um fechamento. No nosso exemplo da Figura 1, dentro desta *tag* nós temos a *tag* `<p>`. Esta *tag* inicia um **parágrafo**, onde tudo o que for escrito a partir dela, aparecerá na tela do navegador como um novo parágrafo até que você feche esta *tag* com `</p>`.


```
<!DOCTYPE html>
<html lang="pt-BR">
  <head>
    <meta charset="utf-8">
    <title>Bootcamp Introdutório Digicad</title>
  </head>
  <body>
    <p>Este é apenas o começo!</p>
    <!--Isto é um comentário-->
  </body>
</html>
```

Figura 4: Destaque para <body>.

1.2.5 COMENTÁRIOS

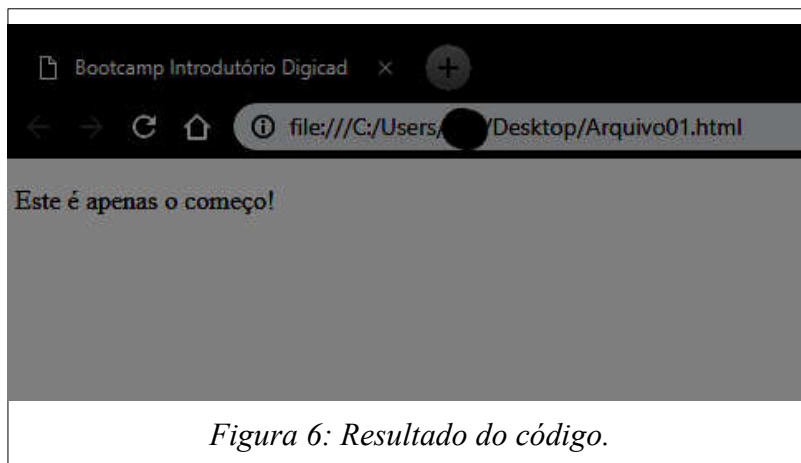
A linha seguinte ao parágrafo é um **comentário**. Comentários são anotações que não são visíveis ao usuário do navegador, apenas à quem lê as linhas de comando. Da mesma forma, comentários são ignorados pelo navegador, não possuindo qualquer efeito sobre a página. Para escrevê-los, inicie com `<!--` e termine com `-->`. Em documentos *HTML* mais longos, é comum o uso de comentários antes e depois de *tags* que elementos aninhados. Isto facilita a leitura e o entendimento das linhas de códigos, assim como faz a indentação.

```
<!DOCTYPE html>
<html lang="pt-BR">
  <head>
    <meta charset="utf-8">
    <title>Bootcamp Introdutório Digicad</title>
  </head>
  <body>
    <p>Este é apenas o começo!</p>
    <!--Isto é um comentário-->
  </body>
</html>
```

Figura 5: Destaque para o comentário.

Agora que você já aprendeu sobre todas as linhas de códigos presentes no exemplo da Figura 1 e as redigiu, salve o documento, mas adicione “**.html**” ao final do nome do arquivo. A seguir, localize o local onde você salvou este arquivo e clique para abri-lo com o navegador de sua escolha. Você deve ver algo similar a figura a seguir.

Nota: O navegador utilizado nas Figuras desta apostila é o Google Chrome versão 70.



Parabéns, esta é a sua primeira criação! Ainda que simples, ela representa o início de uma jornada e, quem sabe, de uma carreira de muito sucesso. Ao longo deste Bootcamp, você aprenderá muitos outros códigos e técnicas que lhe permitirão criar páginas muito mais complexas. Você irá inclusive criar sua própria página pessoal, onde poderá colocar em seu portfólio os projetos que você criará ao longo deste treinamento e, se assim optar, os projetos que você criará ao realizar os demais treinamentos aqui na Digicad.

Antes de seguirmos adiante, algumas observações ainda sobre a Figura 6. Note que aquilo que foi digitado por você no título, agora aparece na aba do navegador. Veja também que onde normalmente apareceria o endereço do site, na verdade aparece o local onde o arquivo está salvo. Isto porque o arquivo *HTML* que você criou não está hospedado na *web*, está apenas em seu computador, por isso ainda não será possível localizá-lo a partir de buscas pela internet, por exemplo. Por último, note que apenas o seu parágrafo aparece visível no conteúdo. Isto porque, como dito, somente o que é colocado entre as *tags* `<body>` e `</body>` ficarão visíveis ao usuário do navegador. Os comandos em si, porém, também não aparecem.

Conhecendo o básico da estrutura e dos códigos de um documento *HTML*, já podemos ir além. A partir de agora, daremos início à sequência de vários outros comandos que lhe auxiliarão a criar as páginas mais complexas.

1.2.6 TÍTULOS E SUBTÍTULOS

A próxima *tag* que aprenderemos será a de cabeçalhos, ou **títulos e subtítulos** do seu conteúdo. Elas vão de `<h1>` a `<h6>`, respeitando uma ordem hierárquica, indo da mais importante, `<h1>`, até a de menor importância, a `<h6>`. Todas elas exigem *tags* de fechamento. E aqui vale uma diferenciação:

- **`<head>`** - Define o cabeçalho do documento HTML, onde são incluídas informações de interesse do navegador, e não do usuário da página;
- **`<title>`** - *Tag* que vai aninhada à `<head>`, e que define o título do documento HTML, aquele que aparecerá como resultado de buscas na internet e na aba ou janela do navegador;

- **<h1> a <h6>** - *Tags* que vão aninhadas ao **<body>**, e que definem os títulos e subtítulos no conteúdo da sua página, como se fossem os capítulos desta apostila (Introdução à *HTML*, História., etc).

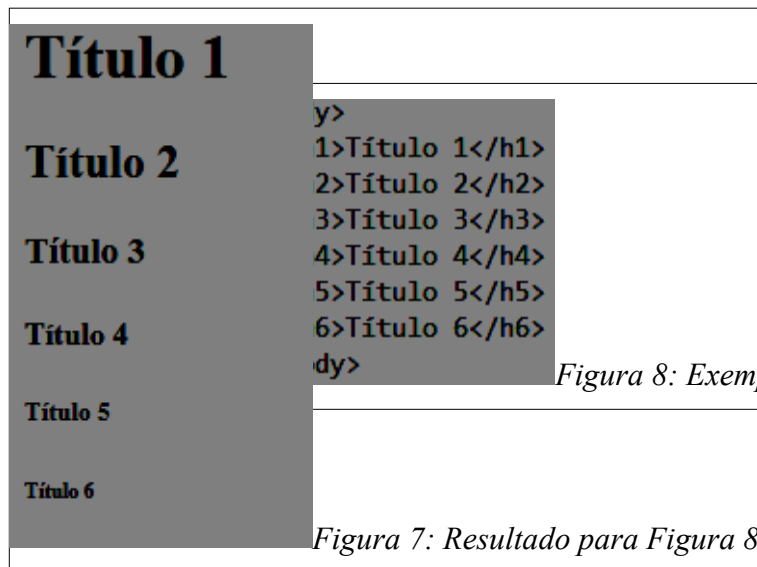


Figura 7: Resultado para Figura 8.

```

y>
1>Título 1</h1>
2>Título 2</h2>
3>Título 3</h3>
4>Título 4</h4>
5>Título 5</h5>
6>Título 6</h6>
dy>

```

Figura 8: Exemplo da utilização de **<h1>** a **<h6>**.

Para facilitar seu entendimento, veja o exemplo das Figuras 7 e 8. Perceba como a hierarquia afeta a apresentação do texto, e esta ordem deve ser sempre respeitada. O que é mais importante aparece maior, e vai diminuindo gradativamente até chegar ao último título, de menor importância. Esta ordem é percebida, não só pelos usuários da página, mas também por ferramentas de busca e acessibilidade, que levam em consideração esta relevância. Por isso, a **<h1>** é a mais importante e normalmente é um título que descreve bem o conteúdo. Não existe, porém, um padrão exato de tamanho para cada título, cada navegador pode adotar um tamanho diferente. Para termos sempre um tamanho exato, precisamos aplicar certa “maquiagem” ao código, mas isso veremos mais a frente quando aprendermos sobre *CSS*.

Note também que cada título aparece um embaixo do outro. Isto não se deve ao fato de você ter escrito os códigos um embaixo do outro (mesmo se você tivesse escrito todos eles em uma única linha, eles ainda apareceriam em linhas diferentes no navegador), mas sim pelo fato de estes elementos serem em bloco. Existem os que são em bloco e os que são em linha. Elementos **em bloco** ocupam todo o espaço horizontal disponível da linha. Mesmo que não haja nada mais escrito na linha, o conteúdo seguinte será jogado para a linha de baixo, como ocorre na Figura 4. Parágrafos, que aprendemos a pouco, também possuem esta característica. Já os elementos **em linha** ocupam apenas o espaço horizontal necessário, não iniciando uma nova linha.

1.2.7 QUEBRAS E ESPAÇAMENTO

Caso você queira, é possível forçar uma quebra de linha onde normalmente não haveria. Para isso, inserimos **
** (do inglês *break*, ou quebra, em português) onde desejamos que ocorra a quebra. Palavras escritas após esta *tag* serão jogadas para a linha de baixo. Outro detalhe, espaços excessivos e quebras de linhas serão desconsiderados pelo navegador. Observe as Figuras 9 e 10 para melhor entendimento:

```
<body>
  <p>
    Note como podemos adicionar uma<br>quebra forçada. Observe também como o espaço      excessivo é
    desconsiderado pelo navegador.
  </p>
</body>
```

Figura 9: Exemplo de quebra e espaçamento.

Note como podemos adicionar uma
quebra forçada. Observe também como o espaço excessivo é desconsiderado pelo navegador.

Figura

10: Resultado da Figura 9.

A tag **<pre>** (*preformatted text*, ou texto pré-formatado em português) é muito similar à **<p>**, porém ela mantém os espaços e quebras de linhas que estão nas linhas de comando. Veja o exemplo:

```
<body>
  <pre>
    Note como podemos adicionar uma<br>quebra forçada. Observe também como o espaço      excessivo é
    mantido pelo navegador, neste caso.
  </pre>
</body>
```

Figura 11: Utilização de <pre>.

Note como podemos adicionar uma
quebra forçada. Observe também como o espaço excessivo é
mantido pelo navegador, neste caso.

Figura 12: Resultado

da Figura 11.

Através da tag **<wbr>** (do inglês *word break*, ou quebra de palavra, em português) nós podemos indicar possíveis locais nas palavras onde poderão ocorrer a quebra de linha. Isto proporciona um maior controle, já que evita que as palavras sejam quebradas em locais errados (o que pode ocorrer em alguns casos específicos), e evita também, no caso de endereços eletrônicos, que haja uma quebra logo após uma pontuação, podendo levar o usuário a acreditar erroneamente que a pontuação é o fim do endereço. Veja os exemplos:

```
<p>http://www<wbr>.digicad<wbr>.com<wbr>.br</p>
```

Figura 13: Exemplo da utilização de `<wbr>`.

```
<p>Pa<wbr>ra<wbr>le<wbr>le<wbr>pí<wbr>pe<wbr>do.</p>
```

Figura 14: Outro exemplo da utilização de `<wbr>`.

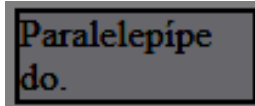


Figura 15: Resultado da Figura 14.

Note que não ocorre a adição automática do hífen na quebra de palavra acima. Para que isto ocorra, você pode utilizar `­` no lugar de `<wbr>`. Este é a entidade de caractere do *soft hyphen*, um hífen que só é adicionado em caso de quebra de palavra, ao contrário do *hard hyphen*, o qual aparece sempre. Você aprenderá sobre estas entidades mais a frente, em um capítulo próprio. Existe ainda a possibilidade de acrescentarmos hifens através de CSS, que também não será abordado agora. Por hora, esteja apenas ciente destas possibilidades.

```
<p>Pa&shy;ra&shy;le&shy;le&shy;pí&shy;pe&shy;do.</p>
```

Figura 16: Exemplo da utilização de “`­`”.

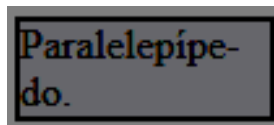
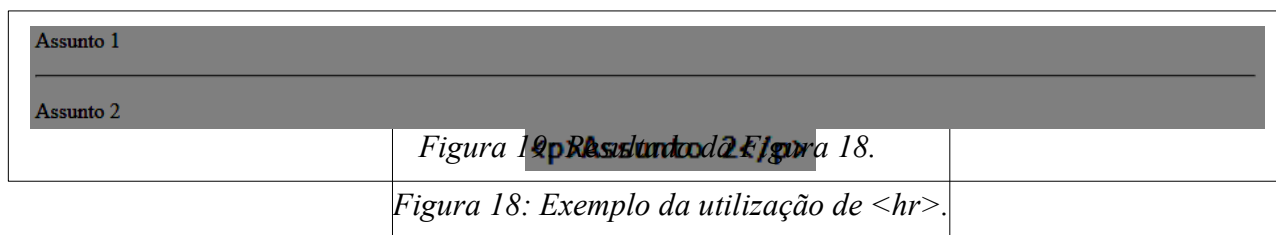


Figura 17: Resultado da Figura 16.

É possível também separar diferentes seções em uma página, como por exemplo, quando há uma mudança no assunto a ser tratado em seguida. Para isso, utilizamos `<hr>`, que fará esta divisão e adicionará uma linha horizontal na separação. Veja:



1.2.8 IMAGENS

Para adicionar imagens a sua página, utilizamos a *tag* ``. Entretanto, esta *tag* precisa ter dois atributos: **src** (do inglês *source*, fonte ou origem em português), que indica o nome do arquivo, quando salvo em seu computador, ou o endereço da imagem na internet; e **alt**, que é o texto que será exibido, caso a imagem não seja carregada (também é importante para que serviços de busca e leitores de tela consigam “ver” o conteúdo da imagem). Esta *tag* também pode vir acompanhada pelo atributo **title**, que define o texto que aparecerá ao colocar-se o mouse sobre a imagem. A *tag* `` não necessita de outra *tag* de fechamento.

Digamos que você queira adicionar à sua página, uma bandeira do Brasil que está salva em seu computador, no mesmo local de seu documento *HTML*, com o nome “bandeira.jpg”. Para isso, basta digitar:

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

Figura 20: Utilizando uma imagem do arquivo.



Figura 21: Resultado, caso a imagem carregue.

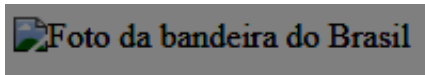


Figura 22: Resultado, caso a imagem não carregue.

Toda via, se a imagem estiver salva em um local diferente do seu documento *HTML*, então será preciso especificar o local completo. Em vez de escrever apenas “bandeira.jpg”, seria necessário escrever, por exemplo, “C:\Users\Usuario1\Desktop\Outra pasta\bandeira.jpg”.

Agora, se você deseja adicionar uma imagem que está na internet, então será preciso adicionar o endereço desta imagem na internet, como o exemplo abaixo:

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

Figura 23: Utilizando uma imagem da internet.

A *HTML5* introduziu duas novas *tags* neste assunto: `<figure>` e `<figcaption>`. A *tag* `<figure>` é utilizada para marcar uma ou mais imagens, ilustrações, diagramas ou até mesmo um trecho de códigos. Semanticamente, esta *tag* deve ser utilizada apenas quando o conteúdo marcado é independente do fluxo principal da página, ou seja, pode ser movida de lugar sem alterar o entendimento do documento. Ela normalmente vem acompanhada da `<figcaption>`, a qual designa um cabeçalho ou legenda à `<figure>`. Vale ressaltar que, mesmo quando há mais de um elemento, a *tag* `<figure>` aceita apenas uma `<figcaption>`. Ambas necessitam de *tags* de fechamento. Acompanhe os exemplos abaixo:

```
<figure>  
    
  <figcaption>Bandeira do Brasil</figcaption>  
</figure>
```

Figura 24: Imagem com legenda.



Bandeira do Brasil

Figura 25: Resultado da Figura 24.

```
<figure>
  <figcaption>Bandeira do Brasil</figcaption>
  
</figure>
```

Figura 26: Imagem com cabeçalho.



Figura 27: Resultado da Figura 26.

```
<figure>
  
  
  
  <figcaption>Bandeiras do Brasil, Estado de São Paulo e São Bernardo do Campo, respectivamente.</figcaption>
</figure>
```

Figura 28: Três imagens com uma legenda.



Figura 29: Resultado da Figura 28.

1.2.9 LISTAS

Existem três tipos de listas em *HTML*: com ordem, sem ordem e de descrição.

- **Listas com ordem:** Define-se o que será a lista através da *tag* ``. Para criar cada tópico desta lista, utilizamos a *tag* ``. Tanto `` como `` necessitam de fechamento. Exemplo:

```
<ol>
  <li>Tópico 1</li>
  <li>Tópico 2</li>
  <li>Tópico 3</li>
</ol>
```

Figura 30: Exemplo de lista com ordem.

```
1. Tópico 1
2. Tópico 2
3. Tópico 3
```

Figura 31: Resultado da Figura 30.

É possível substituir os números ordenando os tópicos por letras. Para isto, basta adicionar o atributo **type** à `` com o valor **a**.

```
<ol type="a">
  <li>Tópico 1</li>
  <li>Tópico 2</li>
  <li>Tópico 3</li>
</ol>
```

Figura 32: Alternativa para a Figura 30.

```
a. Tópico 1
b. Tópico 2
c. Tópico 3
```

Figura 33: Resultado da Figura 32.

- **Listas sem ordem:** Define-se o que será a lista através da *tag* ``. Para criar cada tópico desta lista, utilizamos a *tag* ``. Tanto `` como `` necessitam de fechamento.

```
<ul>
  <li>Tópico 1</li>
  <li>Tópico 2</li>
  <li>Tópico 3</li>
</ul>
```

Figura 34: Exemplo de lista sem ordem.

- Tópico 1
- Tópico 2
- Tópico 3

Figura 35: Resultado da Figura 34.

- **Listas de descrição:** Esta diferencia-se um pouco das outras duas. Para definir o que será a lista, utilizamos a tag `<dl>`. Já os tópicos, cada um terá um título (tag `<dt>`) e uma definição (tag `<dd>`).

```
<dl>
  <dt>Aerovia</dt>
  <dd>Via para trânsito de aeronaves</dd>
  <dt>Hidrovia</dt>
  <dd>Via para trânsito de embarcações</dd>
</dl>
```

Figura 37: Resultado da Figura 36.

Figura 36: Exemplo de lista de descrição.

1.2.10 TABELAS

Para criarmos uma tabela em nossa página, utilizaremos a tag `<table>`, que delimitará a área da tabela, e aninhadas a esta, usamos as tags `<tr>`, para delimitar o que estará em cada linha, e `<td>` no caso das colunas. Podemos também dar um título a esta tabela através da tag `<caption>`, devendo esta ser sempre a primeira tag aninhada em `<table>`. Todas requerem tags de fechamento. Veja o exemplo de uma tabela simples abaixo:

```
<table>
  <caption>Título</caption>
  <tr>
    <td>A</td>
    <td>B</td>
  </tr>
  <tr>
    <td>C</td>
    <td>D</td>
  </tr>
  <tr>
    <td>E</td>
    <td>F</td>
  </tr>
</table>
```

Figura 38: Código para a Figura 39.

Título	
A	B
C	D
E	F

Figura 39: Exemplo de tabela.

Porém, a tabela do exemplo acima é bem simples. Podemos incrementar as tabelas através da utilização de outras *tags*. A *tag* `<th>` é utilizada para os títulos dentro da tabela, criando um destaque para o que for escrito entre esta *tag* e sua *tag* de fechamento.

```
<table>
  <caption>Veículos por empresa</caption>
  <tr>
    <th></th>
    <th>Carros</th>
    <th>Motos</th>
  </tr>
  <tr>
    <th>Empresa 1</th>
    <td>30</td>
    <td>20</td>
  </tr>
  <tr>
    <th>Empresa 2</th>
    <td>40</td>
    <td>50</td>
  </tr>
</table>
```

s por empresa

	Carros	Motos
1	30	20
2	40	50

Figura 41: Resultado da Figura 40.

Figura 40: Exemplo da utilização de `<th>`.

Podemos destacar ainda dois atributos: **colspan** e **rowspan**, **colspan** determina quantas colunas uma célula ocupará, enquanto o **rowspan** determina o mesmo, mas para as linhas. No exemplo abaixo, a última linha é ocupada por uma única célula que se estende por 3 colunas.

```
<table>
  <caption>Veículos por empresa</caption>
  <tr>
    <th></th>
    <th>Carros</th>
    <th>Motos</th>
  </tr>
  <tr>
    <th>Empresa 1</th>
    <td>30</td>
    <td>20</td>
  </tr>
  <tr>
    <th>Empresa 2</th>
    <td>40</td>
    <td>50</td>
  </tr>
  <tr>
    <td colspan="3">Dados referentes a Jan/19.</td>
  </tr>
</table>
```

Resultado da Figura 42.

Figura 42: Exemplo da utilização de *colspan*.

Para tornarmos nossos códigos ainda mais organizados, utilizamos outras três *tags*: **<thead>**, para determinarmos o cabeçalho da tabela, **<tbody>**, o corpo, e **<tfoot>**, o rodapé. Todas precisam de *tags* de fechamento. Veja:

```

<table>
  <caption>Veículos por empresa</caption>
  <thead>
    <tr>
      <th></th>
      <th>Carros</th>
      <th>Motos</th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <th>Empresa 1</th>
      <td>30</td>
      <td>20</td>
    </tr>
    <tr>
      <th>Empresa 2</th>
      <td>40</td>
      <td>50</td>
    </tr>
  </tbody>
  <tfoot>
    <tr>
      <td colspan="3">Dados referentes a Jan/19.</td>
    </tr>
  </tfoot>
</table>

```

Figura 44: Exemplo da utilização de

<thead>, *<tbody>* e *<tfoot>*.

Note que todas as tabelas dos exemplos não possuem divisórias ou qualquer outro tipo de estilização (com exceção do negrito em alguns casos). Você aprenderá a fazer isso mais a frente.

1.2.11 LINKS

Um site normalmente possui vários *links* em sua página. Estes *links*, ao serem clicados, direcionam o usuário a uma página de outro site ou a uma outra página do mesmo site (os dois casos são *links* externos), ou pode até direcionar o usuário a uma parte específica da própria página (*link* interno). Podemos transformar vários elementos em *links*, como textos e imagens. Para criar um *link* em sua página, basta adicionar uma **âncora** (*anchor*, em inglês) através da *tag* `<a>`, adicionando também uma *tag* de fechamento. Assim como a ``, esta *tag* vem acompanhada de um atributo obrigatório, o **href**, que indicará o destino para onde o usuário será redirecionado. Acompanhe os casos abaixo.

1.2.11.1 Transformando um texto em um *link* externo

Para criar um texto que direcione o usuário a um conteúdo externo, basta seguir o exemplo abaixo:

```
<p>Este <a href="http://www.digicad.com.br/">link</a> abrirá uma outra página.</p>
```

Figura 45: Transformando texto em link externo.

No exemplo acima, a palavra “link” foi transformada em um **link externo**, pois o *link* nos levará a um conteúdo externo, no caso o site da Digicad. Note que o restante da frase permanece sendo um texto comum. Isto porque posicionamos apenas a palavra “link” entre as *tags* `<a>` e ``. Assim, você poderá variar a extensão daquilo que deseja transformar em um *link*, ao variar a posição das *tags* `<a>` e ``. Por padrão, o navegador carregará este novo conteúdo na mesma janela ou aba em que o usuário estiver. O mesmo efeito poderia ser obtido ao adicionarmos o atributo **target**, que determina onde será carregado este novo conteúdo, acompanhado do valor **_self**, como no exemplo abaixo:

```
<p>Este <a href="http://www.digicad.com.br/" target="_self">link</a> abrirá uma outra página.</p>
```

Figura 47: Alternativa para a Figura 45.

Mas se você quiser que o novo conteúdo seja carregado em uma nova janela ou aba, basta substituir o valor do **target** por **_blank**:

```
<p>Este <a href="http://www.digicad.com.br/" target="_blank">link</a> abrirá uma outra página.</p>
```

Figura 48: Neste caso, o conteúdo abrirá em uma nova janela ou aba.

1.2.11.2 Transformando uma imagem em um *link* externo

Mas e se você quiser transformar um outro elemento em *link*, como uma imagem, por exemplo? Basta posicionar as *tags* âncoras antes e depois da sua imagem. No exemplo abaixo, transformaremos nossa imagem da bandeira do Brasil em um *link* que levará o usuário à sua página da Wikipédia. Acompanhe:

```
<a href="https://pt.wikipedia.org/wiki/Bandeira_do_Brasil"></a>
```

Figura 49: Transformando a imagem da bandeira vista anteriormente em link externo.

1.2.11.3 Transformando um texto em um *link* interno

Digamos que você queira criar um *link* com a palavra “bandeira” para que o usuário possa clicá-lo, e assim, ser direcionado à imagem da bandeira que se encontra mais abaixo, na mesma página. Isto é um **link interno**. A estrutura deste *link* é praticamente idêntica à de um externo, entretanto, em vez de colocarmos uma referência externa em **href**, referenciaremos um **id**. O

atributo `id` dá uma identidade a um elemento. Esta identidade deve ser única para que não haja conflito de referências. Por exemplo, podemos adicionar um `id` à nossa imagem da bandeira, ficando assim:

```

```

Figura 50: Atribuindo um id a uma imagem.

Agora, ao criarmos o *link*, basta referenciar o `id` do elemento no valor de `href`. Para referenciar um `id`, acrescente uma **cerquilha** “#” (também conhecida como “jogo-da-velha”, ou então *hashtag* ou *pound* em inglês) antes do nome dado ao `id` do elemento. Para referenciar nossa bandeira, digitamos:

```
<p>Clique <a href="#bandeira">aqui</a> para ir até a bandeira.</p>
```

Figura 51:

Criando um link interno.

Assim, quando o usuário clicar no *link*, a página irá se deslocar até a parte onde estiver localizado o `id` referenciado, no nosso exemplo, a imagem da bandeira.

1.2.12 RESERVANDO UM ESPAÇO

Muitas vezes, enquanto estamos escrevendo nossos códigos, nós colocamos um parágrafo ou um *link*, por exemplo, mas deixamos um texto ou caractere que reserva o espaço para o `href` do *link* ou o texto que será escrito no parágrafo, seja porque ainda não temos o que colocar no lugar, seja porque estamos apenas fazendo um teste.

No caso do *link*, ao colocarmos apenas uma cerquilha (#) como valor de `href`, estamos criando um *link* morto, ou **dead link** em inglês. Isto previne o redirecionamento quando o usuário clicar no *link*.

```
<p>Este <a href="#">link</a> abrirá uma outra página</p>
```

Figura 52: Transformando um

link em dead link.

No caso do texto, nós utilizamos um texto provisório, falso (*dummy text*, em inglês) em latim, conhecido como **lorem ipsum**. O latim é utilizado para que as pessoas saibam que é um texto provisório e não se distraiam com o que está escrito, já que o idioma é considerado uma língua morta, sem muitas pessoas que saibam lê-lo. Esta é uma prática adotada desde antes do início da programação, pois o trecho utilizado simula bem um texto verdadeiro, com palavras de variados tamanhos e pontuação. Sua origem, segundo o Prof. Dr. Richard McClintock, da Universidade Mapden-Sydney, remete à obra “*De Finibus Bonorum et Malorum*” (Os Extremos do Bem e do Mal) de Cícero, escrito em 45 A.C. Várias versões da passagem podem ser encontradas na internet, já que com o passar dos anos, muitas variações surgiram, tendo em vista que muitos têm utilizado

palavras aleatórias ou que sequer existem, mas uma das versões mais comuns de se encontrar é “*Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.*”, podendo esta ser repetida várias vezes, dependendo do tamanho do texto que se busca.

1.2.13 FORMULÁRIOS

É muito comum sites e aplicações solicitarem algum tipo de informação que será fornecida pelo usuário. Para isso, são criados **formulários** com alguns campos onde o usuário digitará ou selecionará algumas opções. Tudo aquilo que será a área do formulário em suas linhas de códigos será delimitada pelas tags **<form>** e **</form>**, e esta tag possui os atributos:

- **action:** seu valor indicará para onde será enviada a informação, podendo ser uma outra página ou um arquivo dentro de um site;
- **method:** seu valor indicará como a informação será enviada, podendo ter os valores **get** (padrão, onde a informação é enviada anexada ao final do caminho informado em action, sendo indicada quando não houver informações sigilosas ou o usuário deseje salvar um eventual resultado do envio do formulário) e **post** (mais recomendado, onde é semelhante ao get, porém ao invés da anexação, ele é enviado em um bloco de dados).

Nota: Nunca aninhe uma <form> dentro de outra, pois pode causar problemas a sua página, dependendo do comportamento do navegador.

1.2.13.1 Caixa de texto

Existem diferente tipos de formulários. O primeiro que explicaremos é a **caixa de texto**, que adiciona um campo onde o usuário poderá escrever uma informação, como na figura abaixo:

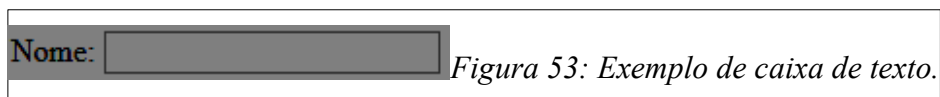


Figura 53: Exemplo de caixa de texto.

```
<form action="arquivo.php" method="post">  
  <label>Nome: </label><input type="text">  
</form>
```

Figura 54: Código para a Figura 53.

Para criarmos um campo onde o usuário adicionará uma informação, utilizamos **<input>**, e para especificarmos que este *input* seja uma caixa de texto, adicionamos o atributo **type**, com valor **“text”**. A tag **<input>** não necessita de outra de fechamento. A fim de informar ao usuário do que se trata o campo a ser preenchido, utilizamos um **rótulo**, através da tag **<label>**. Esta sim, necessita

de outra *tag* de fechamento. Esta *tag* também é importante para leitores de tela, facilitando a identificação para deficientes visuais.

A fim de melhorar a acessibilidade, adicionamos à `<label>` o atributo **for**, que cria uma associação direta com o campo correspondente, e na *tag* deste campo, adicionamos um `id`.

```
<label for="id_nome">Nome: </label><input type="text" id="id_nome">
```

Figura 55: Adicionando id e nome.

Podemos deixar um texto escrito dentro da caixa de texto até que o usuário digite algo no campo. Para tanto, utilizamos junto o atributo **placeholder**.

Nome: Nome completo

Figura 56: Caixa de texto com um placeholder.

```
<label for="id_nome">Nome: </label><input type="text" id="id_nome" placeholder="Nome completo">
```

Figura 57: Adicionando placeholder.

Se desejarmos que o preenchimento deste campo seja obrigatório, basta incluirmos o atributo **required**, como na figura abaixo:

```
<label for="id_nome">Nome: </label><input type="text" id="id_nome" placeholder="Nome completo" required>
```

Figura 58: Adicionando required.

Da mesma forma, poderíamos adicionar os atributos **disabled**, para deixar o campo desabilitado, e **readonly**, onde o campo aparece habilitado, mas o usuário não consegue selecioná-lo, apenas lê o que estiver escrito.

Para que haja uma diferenciação das informações do formulário que são enviadas ao servidor, é preciso nomear cada um dos elementos com o atributo **name**. Veja o exemplo com duas caixas de texto:

```
<label for="id_nome">Nome: </label><input type="text" id="id_nome" placeholder="Nome completo" required name="nome">  
<label for="id_idade"> Idade: </label><input type="text" id="id_idade" placeholder="Idade" required name="idade">
```

Figura 59: Adicionando name

1.2.13.2 Caixa de números

No exemplo anterior, utilizamos uma caixa de texto para criar um campo para o usuário inserir a idade. Entretanto, poderíamos utilizar um campo específico para números, utilizando uma **caixa de números**, onde trocaríamos o valor de type para **number**. Esta mudança também acarreta na aparição de um seletor, como na Figura 60.

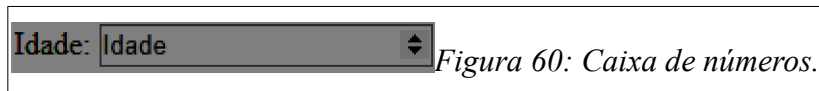


Figura 60: Caixa de números.

```
<label for="id_idade"> Idade: </label><input type="number" id="id_idade" placeholder="Idade" required name="idade">
```

Figura 61: Código para a Figura 60.

Podemos também delimitar um valor mínimo (**min**), um máximo (**max**) e a proporção em que estes números podem variar (**step**).

```
<label for="id_idade"> Idade: </label><input type="number" id="id_idade" required name="idade" min="0" max="130" step="1">
```

Figura 62: Adicionando regras à caixa de números.

No exemplo acima, estamos dizendo que a idade mínima que pode ser inserida é 0, a máxima é 130, e estamos dizendo que os números só podem variar de 1 em 1.

1.2.13.3 Caixa de busca

Se desejamos introduzir um campo onde o usuário poderá digitar algo para realizar uma **busca**, então utilizaremos um valor próprio para isto, o **search**. Apesar de a aparência ser igual ao text, isto possibilita que o navegador e ferramentas consigam saber a diferença deste campo para um de texto comum.



Figura 63: Caixa de busca.

```
<label for="id_busca">Busca: </label><input type="search" id="id_busca" name="busca">
```

Figura 64: Código para a Figura 63.

1.2.13.4 Caixa de e-mail

Seguindo a mesma linha de raciocínio, temos o campo de **e-mail** onde colocamos o valor **email** em type. Esta também, apesar de não aparentar diferença para uma caixa de texto, permite que o navegador e outras ferramentas percebam a diferença. Neste caso, o teclado virtual do *smartphone*, por exemplo, poderia ser alterado automaticamente, apresentando o “@” entre as opções.



 E-mail:

Figura 65: Caixa de e-mail.

```
<label for="id_email">E-mail: </label><input type="email" id="id_email" name="email">
```

Figura 66: Código para a Figura 65.

1.2.13.5 Caixa de telefone

Neste caso, adicionamos o valor **tel** em type. E da mesma forma do caso anterior, a diferença residiria na semântica e na possibilidade do teclado virtual apresentar apenas números, por exemplo.

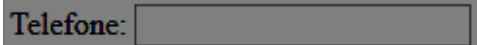

 Telefone:

Figura 67: Caixa de telefone.

```
<label for="id_tel">Telefone: </label><input type="tel" id="id_tel" name="tel">
```

Figura 68: Código para a Figura 67.

1.2.13.6 Caixa URL

URL é a sigla para *Uniform Resource Locator*, Localizador Uniforme de Recursos em português, e refere-se ao endereço de rede de um recurso. Então, se no campo será digitado um endereço de um site, por exemplo, utilizamos a caixa **URL**, através do valor **url**. E seguindo o mesmo caso do tópico anterior, o teclado virtual poderia, por exemplo, mostrar as teclas “www” e “.com”.


 Site:

Figura 69: Caixa URL.

```
<label for="id_url">Site: </label><input type="url" id="id_url" name="url">
```

Figura 70: Código para a Figura 69.

1.2.13.7 Dia e hora

Aqui temos vários valores que podem ser utilizados em type, dependendo do que queremos.

- **date:** Para inserção de data (dia, mês e ano);

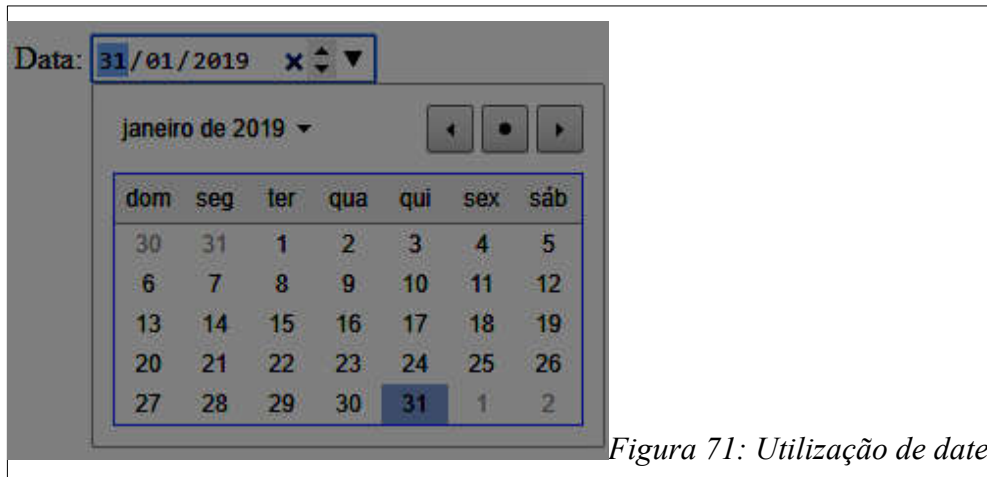


Figura 71: Utilização de date.

```
<label for="id_date">Data: </label><input type="date" id="id_date" name="date">
```

Figura 72: Código para a Figura 71.

- **datetime-local:** Para inserção de data (dia, mês e ano) e horário (hora, minuto, segundo e fração de segundo), sem o ajuste do fuso horário *UTC* (*Universal Time Coordinated*, Tempo Universal Coordenado, em português);

```
<label for="id_dt1">Data e hora: </label><input type="datetime-local" id="id_dt1" name="dt1">
```

Data e hora: 31/01/2019 22:58

Figura 74: Código para a Figura 73.

The screenshot shows a web form with a label 'Data e hora:' and an input field of type 'datetime-local'. The input field displays '31/01/2019 22:58'. Below the input field, a calendar widget is visible, showing the month of January 2019. The date 31 is highlighted in blue. The calendar has a header with days of the week (dom, seg, ter, qua, qui, sex, sáb) and a grid of dates from 30 to 31.

Figura 73: Utilização de datetime-local.

- **month:** Para inserção de mês e ano, sem fuso horário UTC;

The screenshot shows a web form with a label 'Mês:' and an input field of type 'month'. The input field displays 'janeiro de 2019'. Below the input field, a calendar widget is visible, showing the month of January 2019. The date 31 is highlighted in blue. The calendar has a header with days of the week (dom, seg, ter, qua, qui, sex, sáb) and a grid of dates from 30 to 31.

Figura 75: Utilização de month.

```
<label for="id_month">Mês: </label><input type="month" id="id_month" name="month">
```

Figura 76: Código para a Figura 75.

- **time:** Para inserção de horário (hora, minuto, segundo e fração de segundo), sem fuso horário UTC;

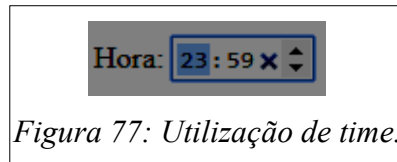


Figura 77: Utilização de time.

```
<label for="id_time">Hora: </label><input type="time" id="id_time" name="time">
```

Figura 78: Código para a Figura 77.

- **week:** Para inserção da semana e ano, sem fuso horário *UTC*.

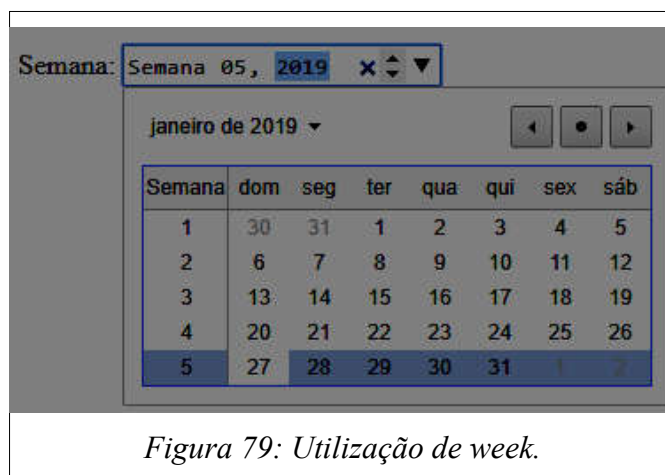


Figura 79: Utilização de week.

```
<label for="id_week">Semana: </label><input type="week" id="id_week" name="week">
```

Figura 80: Código para a Figura 79.

1.2.13.8 Caixa de senha

O valor **password** cria uma **caixa de senha**, onde o usuário digita uma senha alfanumérica, mas o que aparece são símbolos, como asteriscos (*) ou círculos (•).



Figura 81: Caixa de senha.

```
<label for="id_ps">Senha: </label><input type="password" id="id_ps" name="ps">
```

Figura 82: Código para a Figura 81.

1.2.13.9 Caixa de cores

É possível permitir que o usuário escolha uma **cor** através do valor **color**.

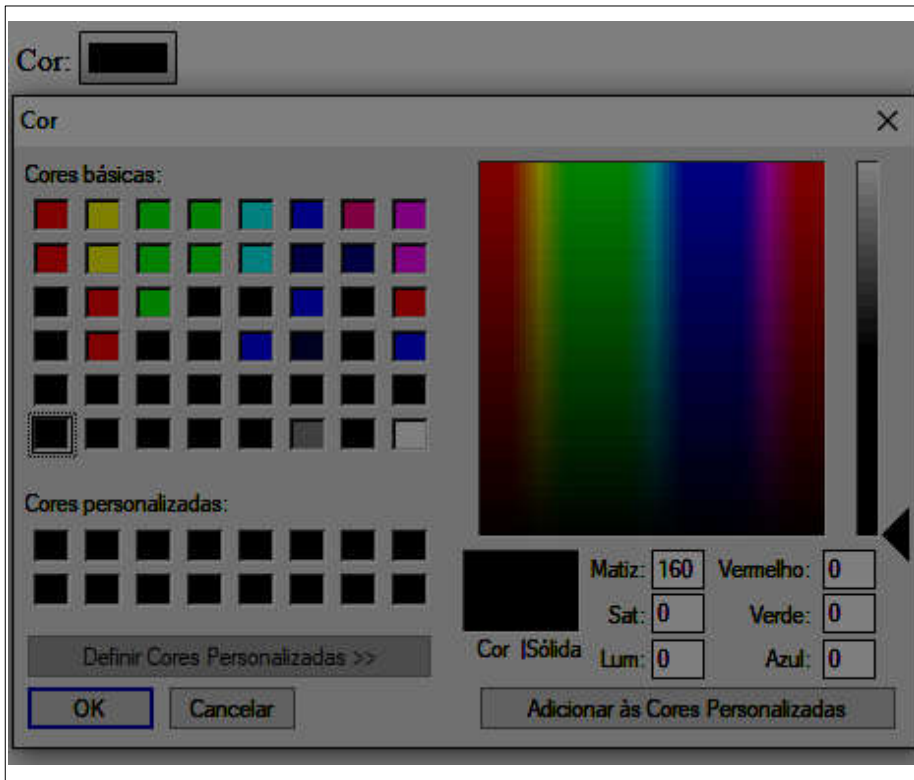


Figura 83: Caixa de cores.

```
<label for="id_color">Cor: </label><input type="color" id="id_color" name="color">
```

Figura 84: Código para a Figura 83.

1.2.13.10 Caixa de texto grande

Se desejamos criar uma **caixa de texto grande**, com mais de uma linha e coluna, então utilizamos a tag **textarea**. Juntamente, adicionamos os atributos **rows** (para a quantidade de linhas) e **cols** (para a quantidade de colunas). Esta tag necessita de fechamento. Aqui também podemos adicionar um *placeholder* e um **limitador de caracteres**, através do atributo **maxlength**.

```
<label for="id_sugestoes">Sugestões: </label><textarea id="id_sugestoes" name="sugestoes" rows="3"
cols="20" placeholder="Digite suas sugestões aqui." maxlength="50"></textarea>
```

Figura 86: Código para a Figura 85.
 Sugestões: aqui. Figura 85: Caixa de texto grande.

1.2.13.11 Radios

Utilizado quando você apresenta um conjunto de opções ao usuário e deseja que ele selecione uma.

Você se considera estudioso(a)?

☐ Sim ☐ Não

Figura 87: Radios.

```
<p>Você se considera estudioso(a)?</p>
<label for="sim"><input id="sim" type="radio" name="estudo" value="s">Sim</label>
<label for="nao"><input id="nao" type="radio" name="estudo" value="n">Não</label>
```

Figura 88: Código para a Figura 87.

Perceba a utilização de um novo atributo: **value**. O valor deste atributo será enviado ao servidor, caso a opção seja selecionada pelo usuário, para que haja a informação de se e qual opção foi selecionada, sendo este um valor qualquer que você escolher. Note também que neste caso, os *names* estão com o mesmo valor (estudo). Isto mostra ao navegador que as opções pertencem a um mesmo grupo, evitando (no caso dos *radios*) que seja selecionada mais de uma opção, por exemplo. Aqui, `<label>` também possui uma outra função: ao clicar em cima da palavra “sim”, por exemplo, a opção correspondente é selecionada, sem a necessidade de clicar exatamente sobre o campo de seleção. Isto acaba sendo extremamente útil em tempos onde a utilização de telas menores, como de *smartphones*, se torna cada vez mais comum.

1.2.13.12 Checkbox

Utilizado quando você apresenta ao usuário algumas opções e deseja que ele selecione uma ou mais destas.

Por qual(is) meio(s) você acessa a internet?

☐ Computador ☐ Tablet ☐ Celular ☐ Tv ☐ Outros

Figura 89: Checkboxes.


```
<p>Por qual(is) meio(s) você acessa a internet?</p>
<label for="computador"><input id="computador" type="checkbox" name="internet" value="cp">Computador</label>
<label for="tablet"><input id="tablet" type="checkbox" name="internet" value="tablet">Tablet</label>
<label for="celular"><input id="celular" type="checkbox" name="internet" value="celular">Celular</label>
<label for="tv"><input id="tv" type="checkbox" name="internet" value="tv">Tv</label>
<label for="outros"><input id="outros" type="checkbox" name="internet" value="outros">Outros</label>
```

Figura 90: Código para a Figura 89.

Por padrão, as opções de *checkboxes* e *radios* não vêm selecionadas. Contudo, é possível deixar selecionada determinada opção, utilizando o atributo **checked**. Observe:

Você se considera estudioso(a)?

☒ Sim ☐ Não

Figura 91: Utilizando checked.

```
<p>Você se considera estudioso(a)?</p>
<label for="sim"><input id="sim" type="radio" name="estudo" value="s" checked>Sim</label>
<label for="nao"><input id="nao" type="radio" name="estudo" value="n">Não</label>
```

Figura 92: Código para a Figura 91.

1.2.13.13 Lista suspensa

É possível criar uma lista onde as opções de escolha pelo usuário ficam escondidas e são expandidas assim que o usuário a seleciona. Para tal lista, conhecida como *dropdown list* em inglês, utilizamos a tag **<select>** com o atributo name, onde cada uma das opções são escritas com a tag **<option>** aninhadas à **<select>**. Estas tags necessitam de outra de fechamento e cada uma das opções também recebem um atributo value.

Selecione uma das

Opção 1 ▼

Opção 1

Opção 2

Opção 3

```
<p>Selecione uma das opções abaixo:</p>
<select id="lista_susp" name="opcoes">
  <option value="1">Opção 1</option>
  <option value="2">Opção 2</option>
  <option value="3">Opção 3</option>
</select>
```

Figura 93: Exemplo de lista suspensa. Figura 94: Código para a Figura 93.

1.2.13.14 Botão simples

Podemos incluir um botão utilizando um atributo input com valor **button**. O que virá escrito (apenas texto) neste botão é definido através do valor do atributo value, e o que acontecerá ao clicarmos no botão será definido através de JavaScript, que veremos mais a frente. Você também verá que podemos inserir botões mais complexos que este, mas é importante ter conhecimento deste.

```
<input type="button" value="Botão">
```

Figura 96: Código para a Figura 95.

Botão

Figura 95: Input com valor button.

1.2.13.15 Botão Reiniciar

Para criar um botão que apagará tudo que foi inserido nos campos do formulário, utilizamos o valor **reset** em type. Visualmente, ele é igual a um botão simples, igual ao do caso acima, porém com o código a seguir.

```
<input type="reset" value="Reset">
```

Figura 97: Código para o botão Reiniciar.

1.2.13.16 Botão Enviar

Depois que o usuário preencheu todo o formulário, ele clicará em um botão que enviará as informações fornecidas, seguindo as orientações que você colocou nos atributos de <form>. Para criar este botão, utilizamos o valor **submit** em type.

```
<input type="submit" value="Enviar">
```

Figura 98: type com valor submit.

Podemos utilizar uma imagem no lugar do botão tradicional para obtermos o mesmo efeito. Para isso, utilizamos **image** em type, adicionamos o atributo src para indicar o local da imagem, seja local ou endereço da imagem na internet, e incluímos um alt para caso a imagem não apareça corretamente. Exemplo:



Figura 99: Imagem como botão enviar.

```
<input type="image" src="enviar.jpg" alt="Botão enviar">
```

Figura 100: Código para a Figura 99.

1.2.13.17 A tag button

A diferença principal entre utilizar esta *tag* em vez de utilizar um input, é que esta *tag* permite adicionar conteúdo ao botão, como textos e imagens. Esta *tag* necessita de outra de fechamento. Veja o exemplo abaixo para melhor entendimento:



É recomendado sempre descrever qual o tipo (type) do botão, pois cada navegador pode utilizar um padrão diferente. Este atributo aceita os valores button (genérico), reset (reiniciar) e submit (enviar).

1.2.13.18 Fieldset

Quando o formulário for grande, ou você quiser dividir o formulário em vários campos, você poderá utilizar a *tag* <fieldset> para fazer esta subdivisão. O que ficar entre esta *tag* e sua *tag* de fechamento, ficará dentro de um contorno que será criado. Para dar um título para cada uma destas áreas, utilizamos a *tag* <legend>, que também requer outra de fechamento. Observe:

```
<button type="submit">
  
  <br>
  Enviar
</button>
```

Figura 102: Código para a Figura 96.

Internet

Por qual(is) meio(s) você acessa a internet?

☐ Computador
 ☐ Tablet
 ☐ Celular
 ☐ Tv
 ☐ Outros

Estudos

Você se considera estudioso(a)?

☒ Sim
 ☐ Não

Figura 103: Utilização de `<fieldset>` e `<legend>`.

```

<fieldset>
  <legend>Internet</legend>
  <form action="arquivo.php" method="post">
    <p>Por qual(is) meio(s) você acessa a internet?</p>
    <label for="computador"><input id="computador" type="checkbox" name="internet" value="cp">Computador</label>
    <label for="tablet"><input id="tablet" type="checkbox" name="internet" value="tablet">Tablet</label>
    <label for="celular"><input id="celular" type="checkbox" name="internet" value="celular">Celular</label>
    <label for="tv"><input id="tv" type="checkbox" name="internet" value="tv">Tv</label>
    <label for="outros"><input id="outros" type="checkbox" name="internet" value="outros">Outros</label>
  </form>
</fieldset>
<fieldset>
  <legend>Estudos</legend>
  <form action="arquivo1.php" method="post">
    <p>Você se considera estudioso(a)?</p>
    <label for="sim"><input id="sim" type="radio" name="estudo" value="s" checked>Sim</label>
    <label for="nao"><input id="nao" type="radio" name="estudo" value="n">Não</label>
  </form>
</fieldset>

```

Figura 104: Código para a Figura 103.

1.2.14 ÁUDIO

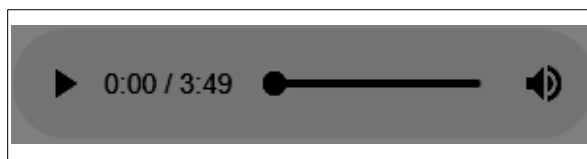


Figura 105: Reprodutor de áudio.

```

<audio src="C:\Users\User1\Music\music.mp3" controls>
  Este navegador não suporta este reprodutor de áudio.
</audio>

```

Figura 106: Código para a

Figura 105.

Você poderá incluir também reprodutores de áudio em sua página. Para tanto, basta utilizar a tag `<audio>` e sua tag de fechamento. Normalmente colocamos um texto escrito entre estas tags que somente será exibido ao usuário, caso o navegador não suporte esta tag. Além disso, esta tag

deve vir acompanhada do atributo `src`, onde seu valor indicará o arquivo de áudio que será reproduzido, sendo suportados os formatos MP3, WAV e OGG (alguns navegadores podem não suportar todos eles, por isso sempre consulte quais os formatos são aceitos no momento, para cada navegador). Além deste atributo, esta *tag* também vem acompanhada do atributo **controls**, que faz com que sejam exibidos controles do áudio, como o de iniciar a reprodução e pausá-la, por exemplo. Outros dois atributos que merecem ser mencionados são **loop**, que reinicia a reprodução do áudio automaticamente toda vez que ele chega ao fim; e **autoplay**, que faz o áudio ser iniciado automaticamente, assim que a página é carregada.

Os navegadores agora mostram um ícone na aba/janela onde está sendo reproduzido o áudio, como na figura abaixo. Isto facilita identificar de qual aba vem o som, quando várias abas estão abertas, por exemplo, e também ajuda deficientes auditivos a saber que existe um áudio sendo reproduzido.

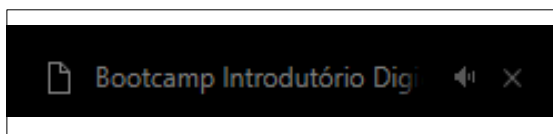


Figura 107: Ícone de áudio sendo reproduzido.

Para evitarmos problemas de compatibilidade de formato de arquivos, ou arquivos corrompidos, podemos colocar mais de um arquivo aninhados à *tag* `<audio>`, através da *tag* `<source>`. Assim, o navegador tentará executar o primeiro da lista e, caso este não seja reproduzido, ele passará para o segundo da lista, e assim sucessivamente, até que ele ache um arquivo que seja executado corretamente. Na *tag* `<source>`, você informará o atributo `src`, combinado ao atributo `type`, onde seu valor especificará o tipo de arquivo. A *tag* `<source>` não necessita de outra de fechamento. Veja:

```
<audio controls>
  <source src="C:\Users\User1\Music\music.mp3" type="audio/mp3">
  <source src="C:\Users\User1\Music\music2.ogg" type="audio/ogg">
  Este navegador não suporta este reproduzidor de áudio.
</audio>
```

Figura 108:

Utilização de `<source>` para aninhar diferentes tipos de arquivo.

1.2.15 VÍDEO

```
<video controls height="200" width="355" poster="C:\Users\User1\Desktop\img.png">
  <source src="C:\Users\User1\Video\video.mp4" type="audio/mp4">
  <source src="C:\Users\User1\Video\video2.ogg" type="audio/ogg">
  Este navegador não suporta este reprodutor de vídeo.
</video>
```

Figura 110: Código para a Figura 109.

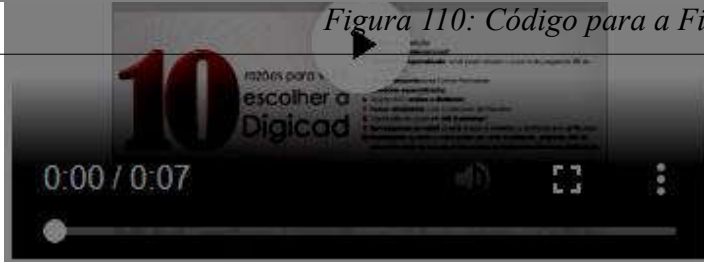


Figura 109: Reprodutor de vídeo.

Além de áudio, também podemos introduzir um vídeo à nossa página, através da *tag* **<video>**, que também requer outra de fechamento. Esta *tag* é semelhante à **<audio>**, sendo acompanhada do atributo **src** para indicar qual arquivo deve ser executado, também pode ter *tags* **<source>** aninhadas a ela, e também colocamos um texto aninhado para caso o navegador não suporte a *tag* **<video>**. A diferença é que esta *tag* também vem acompanhada dos atributos **width**, que indicará a largura do vídeo, e **height**, que indica a altura, assim temos um melhor controle sobre o espaço utilizado em nossa página (Na Figura 110, estes valores estão em pixels). Os formatos de vídeo suportados são MP4, WebM e Ogg (aqui também é recomendado verificar a compatibilidade de cada navegador). Entre outros atributos que podem vir nesta *tag*, podemos citar **controls**, **loop** e **autoplay**, como em **<audio>**, além de **muted**, caso queira que o vídeo inicie mudo; e **poster**, que define uma imagem para ser exibida enquanto o vídeo não é reproduzido.

1.2.16 SEÇÕES DO CONTEÚDO

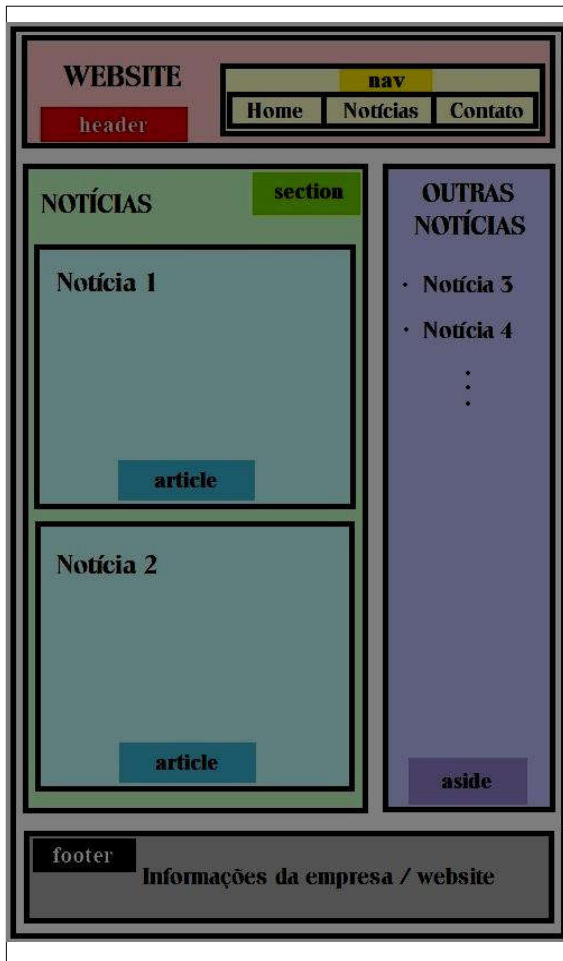


Figura 111: Exemplo da divisão de uma página.

O *HTML5* trouxe, entre suas novidades, a possibilidade de subdividir a página em novas seções para uma melhor organização, e para deixar mais claro o que cada parte representa em sua página. A seguir, uma breve descrição de algumas destas seções.

1.2.16.1 Header

A tag **<header>** define um bloco que será um conteúdo introdutório, uma seção cabeçalho do conteúdo da sua página, ou então, uma seção que contém links de navegação, onde geralmente encontramos logotipos, ícones, sumários, formulários e *tags* de títulos e subtítulos (h1 a h6) aninhadas a esta. Por delimitar uma seção, esta *tag* exige outra de fechamento, assim como todas as outras que veremos aqui no capítulo “Seções do conteúdo”. É possível ter mais de uma **<header>** em seu documento, entretanto ela não pode vir dentro de outra **<header>**, ou das *tags* **<footer>** e **<address>**, que veremos mais a frente.

1.2.16.2 Nav

A tag **<nav>** define um bloco de *links* de navegação. Não serão todos os *links* da sua página que estarão nesta seção, apenas os principais que o usuário utilizará para navegar em seu *website*.

1.2.16.3 Section

A tag **<section>** é utilizada para definir seções genéricas, agrupando tudo aquilo que você considera ser de um mesmo tema.

1.2.16.4 Article

A tag **<article>** define conteúdos que são autônomos, que podem ser redistribuídos sem perder sentido. Normalmente são postagens em fóruns, blogs, comentários e reportagens.

1.2.16.5 Aside

A tag **<aside>** define um conteúdo que não é o principal, mas ainda está relacionado a este.

1.2.16.6 Footer

A tag **<footer>** determina o rodapé do conteúdo da sua página, e traz informações sobre a seção ou documento a que ela está inserida, como por exemplo, informações de autoria, contatos, mapa do site, documentos relacionados e *links* para retornar ao topo da página. É possível ter mais de uma **<footer>** em seu documento.

1.2.17 ENTIDADES DE CARACTERES

Como você já observou, alguns caracteres possuem um significado especial em *HTML*, como por exemplo os parêntesis angulares (“<” e “>”). Se você quiser utilizar estes caracteres em seu texto, e não como códigos, pode ser que ocorram erros. Para que isto não ocorra, precisamos utilizar suas respectivas **entidades de caracteres**, ou *character entities* em inglês. Estas entidades são identificadores para cada caractere ou símbolo, e dizem ao navegador qual deve ser exibido. Estas entidades possuem três partes: começam com “&”; em seguida vem o nome da entidade (ex. *lt*) ou “#” e seu número (ex. #60); e por fim, terminam com ponto e vírgula “;”. Assim, se você quiser introduzir em seu texto o caractere “<”, você utilizará “<” ou “<”. Por exemplo, para escrever “<p>” como sendo um texto, e não código, você digitaria:



Figura 112: Exemplo da utilização de character entities.

Uma entidade muito comum é o espaço não-separável “ ” (do inglês *non-breaking space*). Este espaço evita que haja uma quebra de linha, caso não caiba tudo em uma linha só. Este espaço também permite a aplicação de vários espaços, tendo em vista que o excesso de espaço simples é desconsiderado pelo navegador. O hífen não-separável “‑” também possui o efeito de não quebrar linha.

Abaixo uma lista com as entidades mais comuns para eventual consulta:

Caractere	Descrição	Nome	Número
	Espaço não-separável	 	
-	Hífen não-separável	-	‑

Caractere	Descrição	Nome	Número
<	Menor que	<	<
>	Maior que	>	>
&	“E” comercial	&	&
¢	Centavo	¢	¢
£	Libra	£	£
€	Euro	€	€
©	Direitos autorais	©	©
®	Marca registrada	®	®
™	Marca comercial	™	™
"	Aspas	"	"
'	Apóstrofo	'	'
!	Exclamação	!	!
#	Cerquilha	#	#
\$	Cifrão	$	$
%	Porcentagem	%	%
(Parênteses esquerdo	((
)	Parênteses direito))
*	Asterisco	*	*
+	Adição	+	+
,	Vírgula	,	,
-	Subtração	−	−
-	Hífen (<i>hard hyphen</i>)	‐	-
-	Hífen (<i>soft hyphen</i>)	­	­
×	Multiplicação	×	×
÷	Divisão	÷	÷
√	Raíz	√	√
∞	Infinito	∞	∞
...	Reticências	…	…
.	Ponto	.	.
/	Barra	/	/
:	Dois pontos	:	:
;	Ponto e vírgula	;	;
=	Igual	=	=
≠	Diferente	≠	≠
≤	Menor ou igual	≤	≤

Caractere	Descrição	Nome	Número
≥	Maior ou igual	≥	≥
?	Interrogação	?	?
@	Arroba	@	@
[Colchete esquerdo	[[
]	Colchete direito]]
\	Barra Inversa	\	\
	Barra vertical	|	|
–	Subtração	_	_
{	Chave esquerda	{	{
}	Chave direita	}	}
°	Grau	°	°
º	Ordinal	º	º
±	Mais ou menos	±	±
←	Seta para a esquerda	←	←
→	Seta para a direita	→	→
↑	Seta para cima	↑	↑
↓	Seta para baixo	↓	↓
↔	Setas horizontais	↔	↔
↕	Setas verticais	↕	↕

Tabela 1: Entidade de caracteres mais comuns.

Nota 1: Apesar de ser mais fácil lembrar-se de alguns dos nomes das entidades, é preferível utilizar a identificação numérica, pois alguns navegadores podem não reconhecer os nomes, enquanto os números costumam ser mais aceitos.

Nota 2: Os identificadores são sensíveis a letras maiúsculas e minúsculas, portanto fique atento.

1.2.18 FORMATAÇÃO DE TEXTO

Para estilizarmos nossa página e seu conteúdo, nós utilizamos CSS, que veremos daqui a pouco. A *HTML* deve ser utilizada apenas para construir e estruturar semanticamente nossa página. Ainda, algumas *tags* acabam tendo efeitos de estilização sobre seus elementos, como quando os títulos de uma tabela ficam em negrito com o uso de <th>. A seguir, ensinaremos algumas destas *tags*, sendo que todas elas necessitam de uma *tag* de abertura e outra de fechamento.

1.2.18.1 Itálico e ênfase

As tags `<i>` (itálico) e `` (ênfase) causam o mesmo efeito visual: deixam o texto em itálico. Entretanto, semanticamente elas são diferentes. Enquanto a tag `<i>` apenas causa um destaque ao texto, e deve ser utilizada, por exemplo, quando digitamos uma palavra estrangeira ou termos técnicos, a tag `` causa uma ênfase ao texto, sendo esta ênfase reproduzida inclusive por leitores de tela, deixando claro a quem ouve, a importância do trecho destacado.

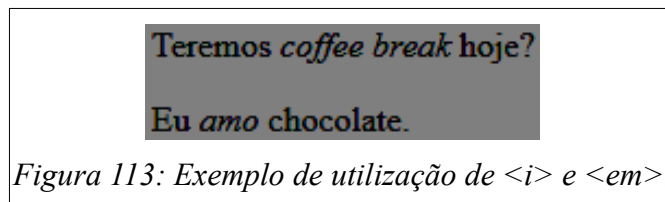


Figura 113: Exemplo de utilização de `<i>` e ``.

```
<p>Teremos <i>coffee break</i> hoje?</p>
<p>Eu <em>amo</em> chocolate.</p>
```

Figura 114: Código para a Figura 113.

1.2.18.2 Negrito e texto importante

Similar ao caso acima, as tags `` (negrito, em inglês *bold*) e `` (texto importante) possuem o mesmo efeito visual, deixam o texto em negrito, mas têm diferente utilização. Enquanto `` provoca um destaque ao trecho, como quando salientamos palavras-chave, a tag `` dá uma forte importância ao trecho destacado, seja pela importância do trecho em si, seja pela importância que ele representa ao conteúdo em que está inserido.

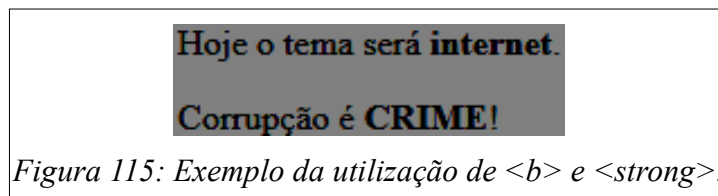


Figura 115: Exemplo da utilização de `` e ``.

```
<p>Hoje o tema será <b>internet</b>.</p>
<p>Corrupção é <strong>CRIME</strong>!</p>
```

Figura 116: Código para a Figura 116.

1.2.18.3 Incluído, excluído e substituído

Quando constatamos um erro no texto de nossa página, ou então uma informação contida nela torna-se irrelevante, imprecisa ou desatualizada, normalmente apagaríamos esta informação e, se for o caso de correção, adicionaríamos o texto correto. Entretanto, existem algumas situações onde simplesmente apagar a informação antiga não é o ideal, pois é de interesse do leitor saber que houve uma mudança no texto, como em documentos oficiais ou editoriais, por exemplo. Nesta matéria, podemos citar três tags:

- **<s>** - Utilizada quando uma informação torna-se irrelevante, imprecisa ou desatualizada. Isto cria uma linha sobre o trecho, “riscando-o”. A seguir, você pode adicionar a atualização. Exemplo:

Inscrição até 01/01/19. ~~Prorrogada até 01/02/19.~~ *Figura 117: Exemplo da utilização de <s>.*

`<p>Inscrição até <s>01/01/19</s>. Prorrogada até 01/02/19.</p>` *Figura 118: Código para a Figura 117.*

- **** - Utilizada para indicar que o trecho foi excluído. Possui o mesmo efeito visual que **<s>**. Exemplo:

Pacote: 2 cadernos, ~~10 canetas,~~ 10 lápis e 5 borrachas. *Figura 119: Exemplo da utilização de .*

`<p>Pacote: 2 cadernos, 10 canetas, 10 lápis e 5 borrachas.</p>` *Figura 120: Código para a Figura 119.*

- **<ins>** - Utilizada para indicar que o trecho foi inserido posteriormente. Neste caso, o texto vem sublinhado. Exemplo:

Aqui moram minha esposa, meus dois filhos e eu. E agora, minha sogra também. *Figura 121: Exemplo da utilização de <ins>.*

`<p>Aqui moram minha esposa, meus dois filhos e eu. <ins> E agora, minha sogra também.</ins></p>` *Figura 122: Código para a Figura 121.*

1.2.18.4 Anotação desarticulada

Anotações desarticuladas são representadas através do uso da *tag* **<u>**. Antigamente, esta *tag* era chamada de “underline”, utilizada apenas para sublinhar trechos em um texto. Entretanto, como hoje a boa prática recomenda a utilização de CSS para estilizações, esta *tag* passou a ser utilizada em situações onde deseja-se destacar um trecho como tendo alguma forma de anotação não-textual aplicada, como em palavras escritas propositalmente de forma errada.

`<p>O correto é asterisco, e não <u>asterístico</u>.</p>` *Figura 123: Exemplo da utilização de <u>.*

O correto é asterisco, e não asterístico. *Figura 124: Resultado da Figura 123.*

Normalmente veríamos a Figura abaixo, onde CSS seria utilizado. Como este não é o tema sendo abordado, mostraremos apenas o exemplo, sem os códigos.

O correto é asterisco, e não asterístico.

Figura 125: Apresentação mais comum para o exemplo utilizado.

1.2.18.5 Sobrescrito e subscrito

As tags `<sup>` e `<sub>` representam os efeitos sobrescrito e subscrito, respectivamente. Quando você quiser escrever uma potenciação, por exemplo 2^2 , você utilizará a tag `<sup>`. Exemplo:

`<p>2² = 4</p>` *Figura 126: Exemplo da utilização de <sup>.*

$2^2 = 4$

Figura 127: Resultado da Figura 126.

Mas quando você quiser que o número venha abaixo, basta utilizar a tag `<sub>`. Exemplo:

`<p>H₂O</p>`

Figura 128: Exemplo da utilização de <sub>.

H₂O

Figura 129: Resultado da Figura 128.

1.2.18.6 Marca-texto

Para que um trecho do seu texto chame a atenção do usuário, e fique destacado com o mesmo efeito visual de uma caneta marca-texto, basta utilizar a tag `<mark>`.

Lembrem-se: **Desempenho Depende De Dedicação!** *Figura 130: Exemplo da utilização de <mark>.*

```
<p>Lembrem-se: <mark>Desempenho Depende De Dedicação!</mark></p>
```

Figura 131: Código para a Figura 130.

1.2.18.7 Avisos, condições ou comentários a parte

A tag `<small>` possui como efeito diminuir o tamanho da letra de um trecho em relação ao restante do texto, e é utilizada para indicar a presença de um aviso legal, regras, condições, direitos autorais ou qualquer outro comentário a parte que deseje fazer.

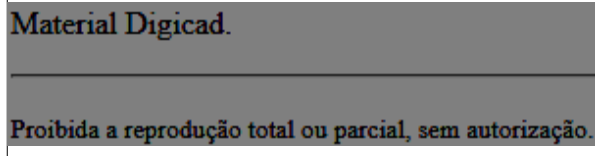


Figura 132: Exemplo da utilização de `<small>`.

```
<p>Material Digicad.</p>
<hr>
<p><small>Proibida a reprodução total ou parcial, sem autorização.</small>
```

Figura 133: Código para a Figura 132.

1.2.18.8 Mudando a direção da escrita

Utilizando-se da tag `<bdo>` (*bidirectional override*, ou sobreposição bidirecional em português) é possível mudar a direção em que um trecho é escrito. Para isto, esta tag vem acompanhada do atributo **dir**, que dará a direção em que o texto será escrito, podendo o seu valor ser **rtl** (*right to left*, da direita para esquerda em português) ou **ltr** (*left to right*, da esquerda para direita, em português), direção padrão para o nosso idioma.

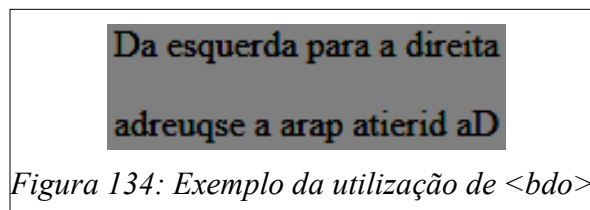


Figura 134: Exemplo da utilização de `<bdo>`.

```
<p><bdo dir="ltr">Da esquerda para a direita</bdo></p>
<p><bdo dir="rtl">Da direita para a esquerda</bdo></p>
```

Figura 135: Código para a Figura 134.

1.2.18.9 Códigos

Quando você escrever alguma linha de código em sua página como sendo um texto para o usuário ler, e não como um código funcional, utilize a tag `<code>`. Com isso, além de seu valor semântico desta tag, o trecho do código que você escrever aparecerá com uma estilização diferente do restante do texto. Aqui, pode ser que você precise utilizar entidades de caracteres para escrever determinados caracteres, como no exemplo abaixo.

Perceba a diferença quando utilizamos `<code>`. *Figura 136: Exemplo da utilização de `<code>`.*

`<p>Perceba a diferença quando utilizamos <code><code></code>.</p> Figura 137: Código para a Figura 136.`

1.2.18.10 Variáveis

Para escrever uma variável de uma equação matemática ou em programação, utilize a tag `<var>`. Além do valor semântico da tag, visualmente estas variáveis aparecerão estilizadas.

`x + y = 10`

Figura 138: Exemplo da utilização de `<var>`.

`<p><var>x</var> + <var>y</var> = 10</p>` *Figura 139: Código para a Figura 138.*

1.2.18.11 Entradas e saídas

Para escrever *inputs* e *outputs* (entradas e saídas em português, respectivamente) de programas de computador podemos utilizar as tags `<kbd>` (*keyboard*, ou teclado em português) para *inputs* e `<samp>` (*sample*, ou amostra em português) para *outputs*.

Apertei Delete sem querer...

O navegador mostrou a seguinte mensagem: Aperte F5 para atualizar.

Figura 140: Exemplo da utilização de `<kbd>` e `<samp>`, respectivamente.

1.2.18.12 Abreviações

```
<p>Apertei <kbd>Delete</kbd> sem querer...</p>
<p>O navegador mostrou a seguinte mensagem: <samp>Aperte F5 para atualizar</samp>.</p>
```

Figura 141: Código para a Figura 140.

Para indicarmos que determinado trecho é uma abreviação, acrônimo ou sigla, utilizamos a tag `<abbr>` (do inglês *abbreviation*). Opcionalmente, esta tag pode vir acompanhada do atributo `title`, onde seu valor contém a descrição completa da abreviação. Alguns navegadores exibem esta informação ao passar-se o cursor sobre a abreviação, que normalmente vem destacada (no exemplo abaixo, aparece sublinhada por uma linha pontilhada).

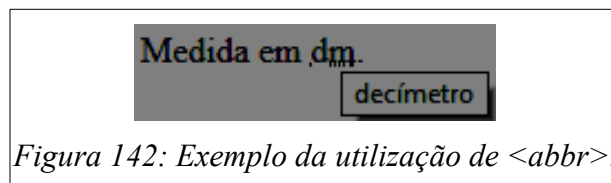


Figura 142: Exemplo da utilização de `<abbr>`.

```
<p>Medida em <abbr title="decímetro">dm</abbr>.</p>
```

Figura 143: Código para a Figura 142.

1.2.18.14 Definições

Quando formos definir um termo, utilizamos a tag `<dfn>`.

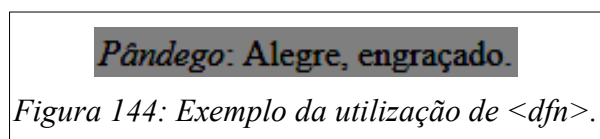


Figura 144: Exemplo da utilização de `<dfn>`.

```
<p><dfn>Pândego</dfn>: Alegre, engraçado.</p>
```

Figura 145: Código para a Figura 144.

Quando quisermos definir uma abreviação, podemos acrescentar o atributo `title` a esta tag, similar a `<abbr>`, onde o valor será a descrição completa e aparecerá ao colocarmos o cursor sobre a abreviação, ou então podemos aninhar uma `<abbr>` à `<dfn>`. Poderíamos até aninhar uma lista à `<dfn>` no caso de haver vários significados.

1.2.19 DIV E SPAN

As *tags* `<div>` e `` são amplamente utilizadas e possuem a função de agrupar, muito úteis ao utilizarmos *CSS*, pois permitem aplicarmos formatações a um grupo inteiro. A primeira *tag*, `<div>`, agrupa os códigos em um bloco, portanto cada `<div>` inicia uma nova linha, enquanto na `` o efeito é em linha. Ambas necessitam de *tags* de abertura e fechamento.

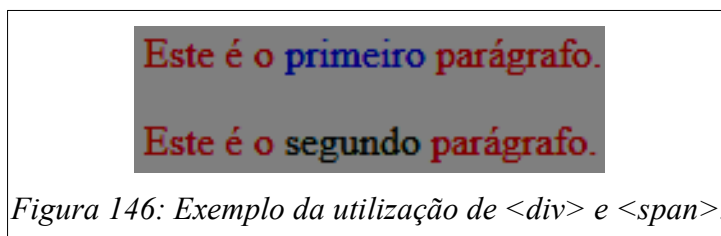


Figura 146: Exemplo da utilização de `<div>` e ``.

```
<div style="color:red;">
  <p>Este é o <span style="color:blue;">primeiro</span> parágrafo.</p>
  <p>Este é o <span style="color:green;">segundo</span> parágrafo.</p>
</div>
```

Figura 147:

Código para a Figura 146.

No exemplo acima, nós aninhamos dois parágrafos a uma `<div>`, e em seguida, orientamos que a cor da letra desta `<div>` fosse transformada em vermelha. Perceba que os dois parágrafos então foram pintados de vermelho. Isto porque eles estão aninhados a `<div>`, portanto a regra de coloração que foi aplicada ao elemento “pai”, também é aplicada aos seus “filhos”, causando um efeito “cascata”. Além disso, introduzimos duas ``, uma em cada parágrafo, orientando que a cor daquele trecho em específico fosse azul e verde, respectivamente. Esta especificidade se sobrepôs à regra mais ampla do `<div>`, tendo em vista que a regra da `` foi aplicada por último.

Nota: Atualmente é mais recomendado deixar a estilização separada do nosso código *HTML*, como já discutido. Entretanto, a fim de criarmos um exemplo simples, utilizamos a estilização junto do nosso código *HTML*.

Isto que você acabou de ver foi uma introdução bem simples à *CSS*. Neste próximo capítulo você aprenderá este e muito outros efeitos.

2. INTRODUÇÃO À CSS

Cascading Style Sheets (CSS), ou Folhas de Estilo em Cascata em português, é a linguagem utilizada para formatar as informações fornecidas pela *HTML*, transformando o conteúdo do documento em um formato com uma melhor apresentação e utilização para o usuário. Para isso, a *CSS* cria um *link* para uma página com os estilos, em vez de se ter os estilos dentro do próprio documento *HTML*. Isto facilita realizar modificações, pois pode-se modificar apenas um arquivo, e possibilita também aplicar os mesmos estilos a diferentes documentos.

Quando não aplicamos nenhuma técnica de formatação ao nosso documento *HTML*, seu conteúdo é apresentado ao usuário com a formatação padrão do navegador, a qual pode variar de navegador para navegador, criando o primeiro problema: quando criamos nossa página, nós a montamos de maneira a transmitir uma informação da melhor forma possível, e queremos que todos os usuários possam ter acesso a essa experiência de forma igual, independentemente do navegador que utilizam. Daí a importância de utilizarmos *CSS* para que nossa página não só fique bem montada, mas também uniforme entre os navegadores. E aí entra outro cuidado: cada navegador pode ter um suporte diferente aos recursos do *CSS*, por isso mais uma vez é importante que se verifique a compatibilidade de cada navegador aos recursos *CSS* que você utilizar. A W3C também cuida da padronização da *CSS*, e traz diversas orientações sobre o tema em suas publicações.

2.1 HISTÓRIA

Conforme a *HTML* crescia, ela começou a abranger uma larga variedade de estilos para atender as demandas dos desenvolvedores, dando um maior controle sobre a aparência da página. Entretanto, isto começou a ter um resultado contrário, pois culminava em uma linguagem cada vez mais complexa, dificultando a tarefa de criar-se páginas com aparências mais consistentes.

A primeira tentativa sobre a técnica de *CSS* foi feita pelo ex-Diretor-Chefe de Tecnologia da Opera Software, o norueguês Håkon Wium Lie, enquanto trabalhava com Tim-Berners-Lee, criador da *World Wide Web*. Combinada a várias outras propostas, principalmente a do cientista holandês da computação Gijsbert (Bert) Bos, e do engenheiro belga da computação Robert Cailliau, a W3C publicou em 1996 a primeira Recomendação *CSS*, a **CSS1**. Através do tempo, a *CSS* foi evoluindo após passar por diversos problemas seguintes a sua publicação, chegando até sua versão mais recente, a **CSS3**, com sua primeira publicação em 1999.

A *CSS3* trouxe uma mudança ao trazer uma divisão de seu conteúdo em módulos. Cada módulo adicionava novas capacidades ou ampliava as já existentes, preservando a retrocompatibilidade com as versões anteriores. Esta divisão em módulos fez com que cada módulo evoluísse em seu próprio ritmo, fazendo com que os módulos também tivessem diferentes estabilidades e compatibilidades. Por isso a última publicação integral foi a *CSS3*, pois desde então os módulos vêm evoluindo separadamente em “módulos de nível 4”. Outros módulos foram recentemente introduzidos à *CSS3*, adquirindo assim “nível 1”. Por vezes, são publicadas coleções de módulos inteiros e parciais que são consideradas estáveis e prontas para serem utilizadas. Até o momento, quatro destes documentos foram publicados: 2007, 2010, 2015 e 2017.

2.2 SEMÂNTICA E SINTAXE

A estrutura *CSS* é bem simples e pode ser aplicada de três formas:

- ⑩ **Inline:** Ou em linha em português, onde as propriedades *CSS* são aplicadas diretamente na *tag* que deseja-se formatar, através do atributo **style**, onde seu valor é composto por uma **propriedade, dois pontos (:) e o valor desta propriedade**. As Figuras 141 e 142, utilizadas anteriormente, são um exemplo. É possível adicionar mais de uma propriedade ao separá-las por **ponto e vírgula (;)**. Muitos consideram sempre colocar um ponto e vírgula ao final, mesmo quando não há mais propriedades a se separar, como sendo uma boa prática. Veja o exemplo:

```
<p style="color:white; background-color:red;">Letra branca e fundo vermelho.</p>
```

Figura 148: Exemplo de *CSS* inline.

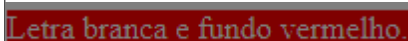


Figura 149: Resultado da Figura 148.

No exemplo acima, dois efeitos estão sendo aplicados ao parágrafo: um para mudar a cor da letra e outro, o fundo. Assim, “color” e “background-color” são as propriedades e “white” e “red” são seus respectivos valores. Explicaremos mais a frente todas estas propriedades que você verá agora, e muito mais. Por hora, foquemos apenas na estrutura.

A *CSS inline*, entretanto, não é a mais indicada, tendo em vista que atualmente procura-se separar o máximo possível a estilização da *HTML*, como discutido anteriormente.

- ⑩ **Interna:** As propriedades *CSS* são aplicadas através da *tag* **<style>** aninhada à **<head>**, requerendo a *tag* **<style>**, outra de fechamento. Neste caso, a estrutura muda um pouco. Como os comandos da *CSS* vão aninhados à **<head>**, e não diretamente nas *tags*, é preciso indicar o que desejamos formatar. O **seletor** possui exatamente esta função. Observe:

```
seletor {propriedade: valor;}
      ou
seletor {
  propriedade: valor;
}
```

Figura 150: Estrutura básica.

É muito comum vermos a segunda apresentação da Figura acima. Quando incluímos várias propriedades, esta segunda apresentação facilita sua visualização, pois vamos inserindo as propriedades uma embaixo da outra. Veja o exemplo a seguir:

```
<head>
  <meta charset="UTF-8">
  <title>Bootcamp Introdutório Digicad</title>
  <style>
    h1, h2 {
      color: green;
    }

    h2 {
      background-color: yellow;
      font-size: 35px;
    }
  </style>
</head>
<body>
  <h1>Título 1</h1>
  <h2>Título 2</h2>
</body>
```

Figura 151: Exemplo de CSS interna.

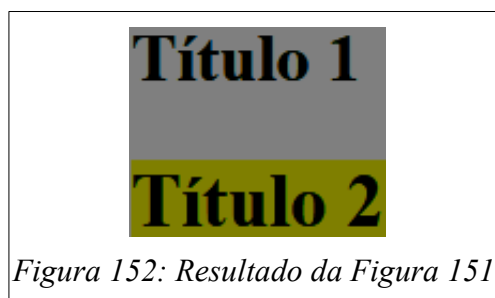


Figura 152: Resultado da Figura 151.

No exemplo da Figura 151, podemos notar algumas informações novas. Perceba que na primeira linha após `<style>` há dois seletores. Isto porque a CSS permite indicar mais de um seletor para aplicarmos as mesmas regras, sem a necessidade de reescrevermos as regras para todos. Podemos fazer isto ao separarmos os seletores através de **vírgulas** (.). No caso acima, estamos dizendo que a cor do texto de `h1` e `h2` devem ser verde. *Tags* são referenciadas sem os parênteses angulares (“<” e “>”).

Na regra de baixo, note que há apenas um seletor, porém há dois comandos: o primeiro transforma o fundo do texto em amarelo, enquanto o segundo estabelece o tamanho da fonte em 35px. Lembre-se que por padrão, `h1` possui um tamanho de fonte maior que `h2`. Entretanto, através de CSS nós mudamos isto, com o único intuito de mostrar-lhe que podemos nos sobrepor à apresentação padrão do navegador.

- ⑩ **Externa:** Esta é a prática mais frequente. Neste caso, as propriedades CSS ficam em um ou mais arquivos independentes, separados do código *HTML*, com a extensão “.css”. Este arquivo não pode conter códigos *HTML*, deve possuir apenas códigos CSS. A interligação entre o documento *HTML* com o arquivo CSS é feita através da *tag* `<link>` aninhada à `<head>`. Para repetirmos o resultado da Figura 149, por exemplo, faríamos igual as Figuras a seguir:

```
<head>
  <meta charset="UTF-8">
  <title>Bootcamp Introdutório DP {
  <link rel="stylesheet" href="e    color: white;
                                   background-color: red;
</head>
<body>                                }
  <p>Letra branca e fundo vermelho.</p>
</body>
```

Figura 154: Código no arquivo CSS.

Figura 153: Código no arquivo HTML.

Letra branca e fundo vermelho.

Figura 155: Resultado das Figuras 153 e 154.

Observe a *tag* `<link>` da Figura 153 e note a presença de dois atributos: **rel** e **href**. O primeiro estabelece a relação entre o documento do *link* e o documento *HTML*. No caso do exemplo, utilizamos o valor “**stylesheet**” que indica ser uma “Folha de Estilo”, já que estamos interligando com um documento *CSS*. O segundo atributo, como visto anteriormente, indica a **URL** do arquivo que está sendo interligado.

2.2.1 COMENTÁRIOS

Os comentários em *CSS* possuem a mesma função dos comentários em *HTML*, são anotações que não são visíveis ao usuário do navegador, apenas à quem lê as linhas de comando. Da mesma forma, comentários são ignorados pelo navegador, não possuindo qualquer efeito sobre a página. Todavia, a forma de escrevê-los em *CSS* muda: iniciamos um comentário com “**/***” e terminamos com “***/**”.

```
/* Isto é um comentário em CSS. */
```

Figura 156: Exemplo de comentário em CSS.

2.2.2 ID E CLASS

Agora, veja o caso abaixo:

Parágrafo 1.

Parágrafo 2.

Figura 157: Resultado das Figuras 158 e 159.

```
<p>Parágrafo 1.</p>
<p>Parágrafo 2.</p>
```

Figura 158.

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

Figura 159: Código CSS.

Perceba que ao referenciarmos “p” no código CSS, a formatação foi aplicada a todos os parágrafos. Mas e se não quisermos aplicar a formatação a todos eles? Para isto, devemos utilizar ids e classes.

Como vimos em *HTML*, **id** é um atributo que cria uma identidade única para a *tag*, e para referenciarmos um id, utilizamos uma **cerquilha** (#) seguida do id. Portanto, se quisermos aplicar a formatação do exemplo anterior apenas para o Parágrafo 1, poderíamos adicionar um id à primeira *tag*, e depois referenciarmos este id no código CSS. Veja:

```
#par1 {
  color: red;
}
```

Figura 160: Código CSS.

```
<p id="par1">Parágrafo 1.</p>
<p>Parágrafo 2.</p>
```

Figura 161: Código HTML.

Parágrafo 1.

Parágrafo 2.

Figura 162: Resultado das Figuras 160 e 161.

O atributo **class**, por outro lado, identifica um grupo. Portanto, é possível termos vários atributos class com o mesmo valor, onde todos eles pertencerão a um mesmo grupo. Para referenciarmos uma class, adicionamos um **ponto** (.), seguido do class. Um atributo class pode inclusive ter mais de um valor, todos eles separados por **espaços**. Veja:

```
<p class="cereais">Trigo.</p>
<p class="cereais">Cevada.</p>
<p class="frutas">Banana.</p>
<p class="frutas">Laranja.</p>
```

Figura 163: Código HTML.

```
.frutas {
  color: red;
}
```

```
.cereais {
  color: green;
}
```

Figura 164: Código CSS.

Trigo.
Cevada.
Banana.
Laranja.

Figura 165: Resultado das Figuras 163 e 164.

Figura 165: Resultado das Figuras 163 e 164.

2.2.3 CORES

Em CSS, há diferentes formas de nos referirmos a cores:

- ⑩ **Nome:** Podemos escrever o nome da cor que queremos. Em CSS, existem 147 nomes para cores atualmente, os quais você poderá utilizar. Entretanto, todos eles são baseados no idioma inglês, assim como quase tudo em programação, como você já deve ter percebido. Citando as cores mais comuns, temos:


Cor	Nome	Tradução	Cor	Nome	Tradução
	black	Preto		pink	Rosa
	blue	Azul		purple	Roxo
	green	Verde		red	Vermelho
	grey	Cinza		white	Branco
	orange	Laranja		yellow	Amarelo

Tabela 2: Nome das cores mais comuns.

Você pode ver todos os 147 nomes em <https://www.w3.org/TR/css-color-3/#svg-color>.

- ⑩ **Hexadecimal** (conhecido como *hex code*, em inglês): Podemos escrever o código hexadecimal da cor que queremos. Este código é formado por uma **cerquilha (#)**, **seguida de seis caracteres** que podem ser de 0 a 9, e de A a F, sendo os dois primeiros caracteres a quantidade de vermelho, os próximos dois, a quantidade de verde, e os dois últimos, a quantidade de azul, as três cores primárias. Eles podem ir, por exemplo, de #000000 (preto) a #FFFFFF (branco). Isto permite uma seleção de mais de 16 milhões de cores. Esta é a forma mais comum de ser utilizada, pois existe uma maior compatibilidade destes códigos com versões mais antigas de navegadores. Assim, temos as cores mais comuns abaixo, desta vez, com seus códigos hexadecimais:











Cor	Nome	Tradução	Hexadecimal
	black	Preto	#000000
	blue	Azul	#0000FF
	green	Verde	#008000
	grey	Cinza	#808080
	orange	Laranja	#FFA500
	pink	Rosa	#FFC0CB
	purple	Roxo	#800080
	red	Vermelho	#FF0000
	white	Branco	#FFFFFF
	yellow	Amarelo	#FFFF00

Tabela 3: Hex codes das cores mais comuns.

Nota: Quando há a repetição de caractere para a cor primária, é possível escrevermos este caractere apenas uma vez. Por exemplo, a cor #112233 pode ser escrita #123, apenas.

- ⑩ **RGB:** Sigla para *Red, Green and Blue*, ou Vermelho, Verde e Azul em português, as cores primárias. Este método assemelha-se ao hexadecimal, onde descrevemos a quantidade das três cores primárias para formarmos a cor que buscamos. Contudo, a forma de escrevermos este código muda. Iniciamos com **rgb** e **entre parênteses colocamos três números**, separados por vírgula, que podem ir de 0 a 250, permitindo-nos escolher a mesma quantidade de cores que o hexadecimal. O primeiro número representa o vermelho, depois o verde, e por fim, o azul. Desta maneira, o preto seria `rgb(0,0,0)` e o branco `rgb(250,250,250)`. É muito comum ver este método ser utilizado também, já que permite um ajuste simples e fácil. A seguir, as cores mais comuns com seus códigos `rgb`:











Cor	Nome	Tradução	Hexadecimal	RGB
	black	Preto	#000000	0, 0, 0
	blue	Azul	#0000FF	0, 0, 255
	green	Verde	#008000	0, 128, 0
	grey	Cinza	#808080	128, 128, 128
	orange	Laranja	#FFA500	255, 165, 0
	pink	Rosa	#FFC0CB	255, 192, 203
	purple	Roxo	#800080	128, 0, 128
	red	Vermelho	#FF0000	255, 0, 0
	white	Branco	#FFFFFF	255, 255, 255
	yellow	Amarelo	#FFFF00	255, 255, 0

Tabela 4: RGBs das cores mais comuns.

- ⑩ **RGBA**: Semelhante ao *RGB*, mas com a adição do canal *Alpha*, que determina a **opacidade** da cor. Este parâmetro é colocado como um quarto valor dentro dos parênteses, porém varia de 0 a 1. Deste modo, se queremos uma cor com opacidade de 50%, digitamos “0.5”. Repare que utilizamos o ponto ao invés da vírgula para o decimal. Isto porque este é o padrão no inglês, e a vírgula é utilizada para separar os valores, portanto fique atento.

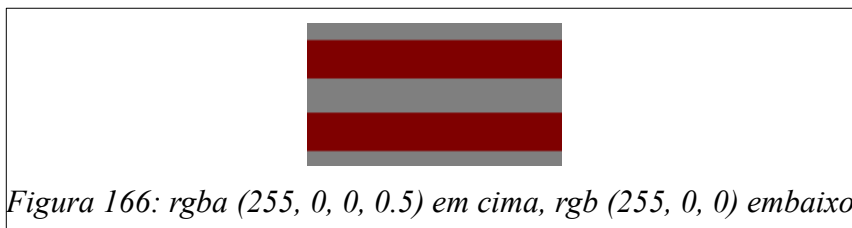
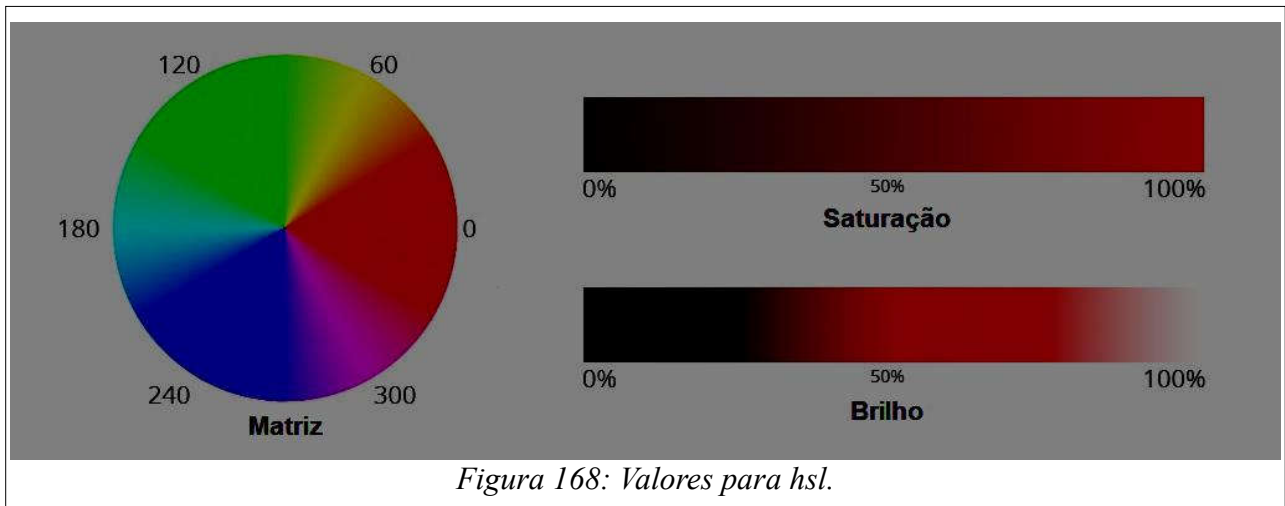
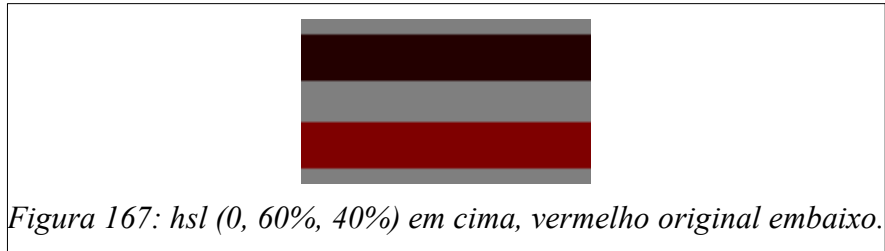
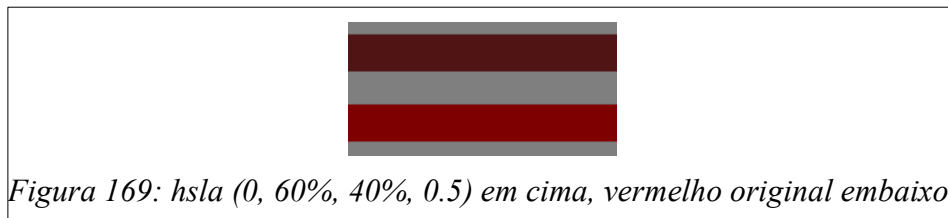


Figura 166: *rgba* (255, 0, 0, 0.5) em cima, *rgb* (255, 0, 0) embaixo.

- ⑩ **HSL**: Sigla para *Hue, Saturation and Lightness*, ou Matriz (ou Tonalidade), Saturação e Brilho (ou Luminosidade) em português. Com este método é possível alterar a saturação e o brilho de uma cor. Para este código, escrevemos **hsl** mais **três valores entre parênteses**, sendo o primeiro a matriz, que varia de 0 a 360, o segundo sendo a saturação, que varia de 0% (cinza) a 100% (cor original), e o terceiro que representa o brilho, que varia de 0% (preto) a 100% (branco), onde 50% representa a cor original.



- ⑩ **HSLA:** Semelhante ao *rgba*, este método permite controlar também a opacidade através da adição de um quarto valor ao *HSL*, variando de 0 a 1.



2.2.4 UNIDADES DE MEDIDA

Existem dois tipos de unidades que podemos utilizar em *CSS*: absolutas e relativas.

- ⑩ **Unidades absolutas:** São unidades de tamanho exato.

Unidade	Comparativo
Centímetro (cm)	1cm = 0,01m
Milímetro (mm)	1mm = 0,01cm
Polegada (in)	1in = 2,54cm

Unidade	Comparativo
Pixel (px)	1px = 1/96in
Ponto (pt)	1pt = 1/72in
Pica (pc)	1pc = 12pt
<i>Tabela 5: Unidades absolutas.</i>	

Nota: Píxel, apesar de ser classificada como unidade absoluta, varia de acordo com o tamanho e a resolução da tela.

10 Unidades relativas: Unidades que são relativas a outras unidades.

Unidade	Descrição
em	1em = o tamanho atual da fonte de seu elemento “pai”.
rem	1rem = o tamanho da fonte de <html>. Caso não esteja especificado, considera-se o tamanho padrão do navegador (geralmente 16px).
ex	1ex = a altura da letra “x” da fonte atual.
ch	1ch = a largura do número zero da fonte atual.
vw	1vw = 1% da largura da área de visualização do navegador.
vh	1vh = 1% da altura da área de visualização do navegador.
vmin	1vmin = 1% da menor dimensão da área de visualização do navegador.
vmax	1vmax = 1% da maior dimensão da área de visualização do navegador.
“Porcentagem”	Porcentagem em relação ao seu elemento “pai”.
<i>Tabela 6: Unidades relativas.</i>	

Nota: O valor da medida deve sempre vir junto da unidade, sem espaço entre eles. A exceção é quando o valor da medida for zero, quando passa a ser dispensável digitar a unidade de medida.

Muito cuidado ao escolher as unidades de medida

Hoje em dia, com o uso crescente de dispositivos móveis, dos mais variados tamanhos e resoluções, torna-se imprescindível a criação de páginas e aplicativos que sejam responsivos, que se ajustem de acordo com o tamanho da tela. Assim, as unidades relativas acabam auxiliando nesta tarefa, pois as medidas podem ser programadas para variar com o tamanho da tela, enquanto o uso de unidades absolutas podem trazer alguns problemas, justamente por serem invariáveis. Estas unidades absolutas podem ser utilizadas, como por exemplo, para casos de impressões, onde o tamanho do papel é pré-determinado. Porém, quando utilizadas para determinar o tamanho da fonte de um texto, por exemplo, podem resultar em letra muito pequenas em determinados tamanhos de tela, problema que afeta principalmente usuários com visão debilitada. Entretanto, apesar de “vw” e “vh” serem relativas, não é recomendado seu uso para o tamanho de fontes. Isto porque estes mesmos usuários com visão debilitada também teriam problema para ler, uma vez que “vw” e “vh” os impedem de usar o zoom dos navegadores para aumentar a letra. Percebe o motivo de se ter cuidado na escolha das unidades?

Atualmente, as unidades “em”, “rem” e porcentagem acabam sendo muito utilizadas para a criação de *layouts* responsivos. Todavia, é preciso estar atento a um detalhe ao utilizar a unidade “em” e porcentagem. Estas unidades são relativas ao elemento “pai”. Assim, se o elemento “pai” possui uma fonte tamanho 10px, um elemento filho com a fonte 2em teria seu tamanho equivalente a 20px (2x10). Até aí, tudo bem. O problema se dá quando possuímos vários elementos aninhados. Observe:

```
<body style="font-size:10px;">
  <div>
    Texto 1
    <div>
      Texto 2
    </div>
  </div>
</body>
```

Figura 170: Exemplo de aninhamento.

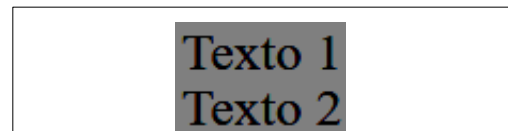


Figura 171: Resultado da Figura 170.

Perceba que no exemplo acima possuímos duas <div>, uma aninhada a outra. Se agora especificarmos que o tamanho do texto das divs será 2em, teremos o resultado abaixo:

```
<body style="font-size:10px;">
  <div style="font-size:2em;">
    Texto 1
    <div style="font-size:2em;">
      Texto 2
    </div>
  </div>
</body>
```

Figura 172: Divs com fonte tamanho 2em.



Notou que “Texto 2” ficou com o dobro do tamanho de “Texto 1”? Mas por que isto ocorre se as duas divs estão com o mesmo tamanho, 2em? Isto porque a primeira <div> tem duas vezes o tamanho do seu elemento “pai”, no caso, <body>. Como <body> tem o tamanho de 10px para a fonte, então a primeira <div> possui uma fonte equivalente a $2 \times 10 = 20\text{px}$. A segunda <div> também possui duas vezes o tamanho do seu elemento “pai”, porém o elemento “pai” neste caso é a primeira <div>, que possui a fonte tamanho 20px. Portanto, a segunda <div> possui uma fonte tamanho $2 \times 20 = 40\text{px}$. Este efeito é conhecido como **compounding**, ou composição em português, onde o fator de hereditariedade acaba tendo um efeito exponencial.

Para tentar contornar este problema, surgiu a “rem”. Como a “rem” baseia-se no tamanho do elemento raiz, ou seja, a tag <html>, o efeito de *compounding* não ocorre, pois a referência é sempre a mesma.

Mais a frente teremos um capítulo exclusivo para a criação de páginas responsivas. Por hora, tenha em mente como cada unidade de medida se comporta.

2.2.5 VALORES GLOBAIS

Existem alguns valores que são globais, podendo ser utilizados para qualquer propriedade *CSS*, bem como qualquer elemento *HTML*. Dentre eles, podemos citar *initial*, *inherit* e *unset*. O valor **initial** determina que a propriedade manterá sua característica original. Ou seja, se aplicada a cor do texto, sua cor será a cor padrão (preta). O valor **inherit** determina que a propriedade herdará as características do seu elemento “pai”. Ou seja, se aplicada a cor do texto, e seu elemento pai tiver a cor de texto vermelha, então o elemento “filho” também terá a cor de texto vermelha. Já o valor **unset** mescla o comportamento dos dois anteriores: retorna à característica herdada do elemento “pai”, quando for o caso, ou no caso de não haver esta hereditariedade, retorna à característica padrão.

```
p {  
  <p style="color:inherit;">Texto.</p>  
}
```

Figura 175: Exemplo da utilização de *inherit* em *CSS inline*.

Figura 174: Exemplo da utilização de *initial*.

2.2.6 FORMATANDO TEXTOS

Agora que você já aprendeu sobre cores e unidades de medidas, vamos ver como aplicar diferentes formatações aos nossos textos.

2.2.6.1 Cor da fonte

Para mudarmos a cor da fonte, como nos exemplos anteriores, nós utilizamos a propriedade **color**, onde colocamos a cor desejada em seu valor, através de qualquer um dos métodos que você viu no capítulo “2.2.3 Cores”. Exemplo:

```
<p>Cor vermelha.</p>
```

Figura 176: Código HTML.

```
p {  
  color: #ff0000;  
}
```

Figura 177: Código *CSS*.

Cor vermelha.

Figura 178: Resultado.

Para *CSS inline*, basta introduzir o atributo **style**. Veja:

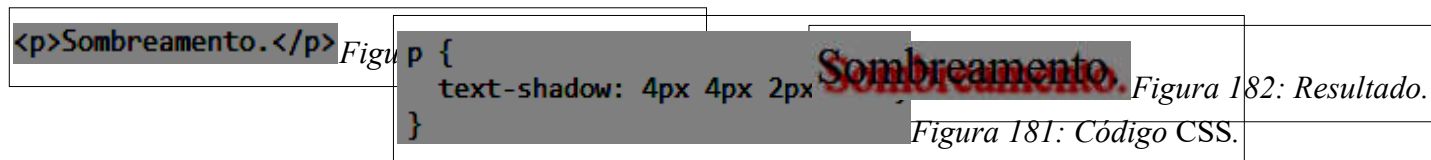
```
<p style="color: #ff0000;">Cor vermelha.</p>
```

Figura 179: *CSS inline*.

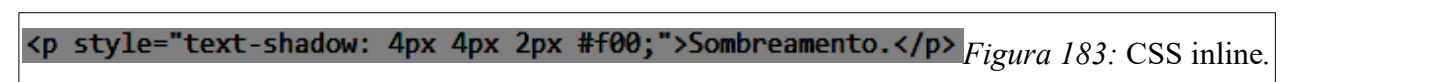
Lembrando que no caso dos exemplos acima, poderíamos ter escrito apenas *#f00* para a cor, já que há a repetição de caracteres.

2.2.6.2 Sombreamento de letras

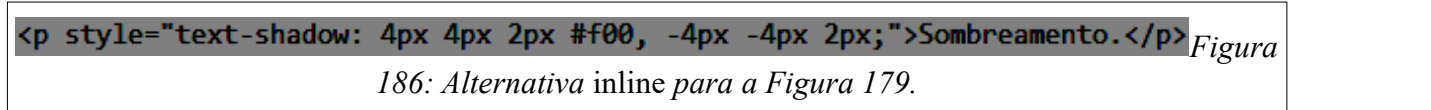
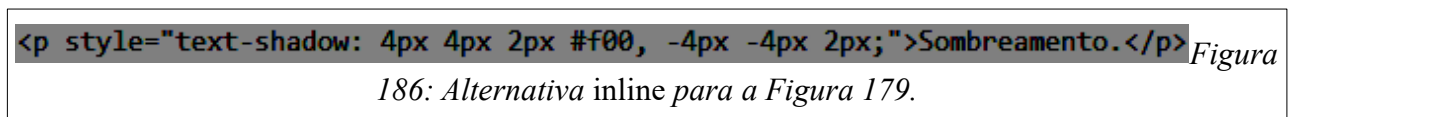
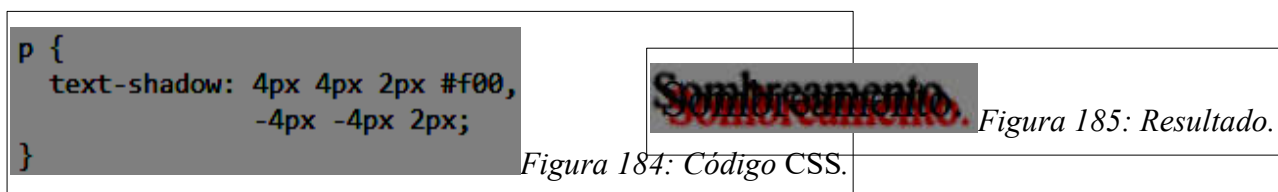
Para introduzirmos um efeito de sombra dos caracteres, utilizamos a propriedade **text-shadow**. Esta propriedade vem acompanhada de quatro valores: o primeiro indica o deslocamento horizontal da sombra em relação ao texto, o segundo indica o deslocamento vertical, o terceiro indica o quão nítida ou “embaçada” será a sombra (valor zero ou a não declaração deste valor, resulta em uma sombra 100% nítida), e o quarto indica a cor da sombra (a não declaração deste valor resulta em uma sombra preta).



Para CSS *inline*, basta introduzir o atributo **style**. Veja:



É possível adicionar mais de uma sombra a um mesmo elemento, separando-as por vírgula. Observe:



2.2.6.3 Tamanho da fonte

O tamanho padrão de fonte, na maioria dos navegadores, para o texto normal, como em parágrafos, é de 16px. Podemos alterar este tamanho através da propriedade **font-size**. Para isto, podemos incluir os seguintes valores:

Valor	Descrição
xx-large	Tamanho extra, extra grande em relação ao padrão do usuário.
x-large	Tamanho extra grande em relação ao padrão do usuário.
large	Tamanho grande em relação ao padrão do usuário.

Valor	Descrição
medium	Tamanho padrão do usuário.
small	Tamanho pequeno em relação ao padrão do usuário.
x-small	Tamanho extra pequeno em relação ao padrão do usuário.
xx-small	Tamanho extra, extra pequeno em relação ao padrão do usuário.
larger	Um tamanho maior do que o elemento “pai”.
smaller	Um tamanho menor do que o elemento “pai”.
“Porcentagem”	Pode-se indicar uma porcentagem. A porcentagem será relativa ao elemento “pai”.
“Tamanho”	Pode-se indicar um tamanho desejado.

Tabela 7: Valores para alteração do tamanho da fonte.

```
<p style="font-size:medium;">Medium</p>
<p style="font-size:smaller;">Smaller</p>
<p style="font-size:x-large;">X-large</p>
<p style="font-size:80%;">80%</p>
<p style="font-size:1.2em;">1.2em</p>
```

Figura 187: Exemplos de alteração do tamanho da fonte.

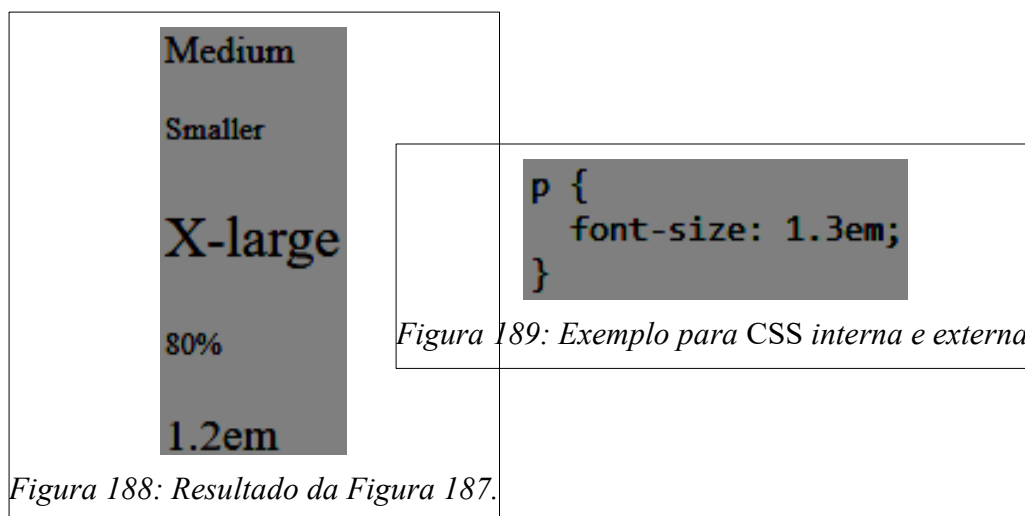


Figura 189: Exemplo para CSS interna e externa.

Nota: Procure utilizar valores relativos em suas medidas, a fim de melhorar a experiência de uso para todos, independentemente do tamanho da tela, conforme já discutido no capítulo “2.2.4 Unidades de medida”.

2.2.6.4 Estilo da fonte

Como vimos em *HTML*, as tags `<i>` e `` não devem ser utilizadas para transformar um texto em itálico, quando o único propósito é a estilização. Em vez disso, devemos utilizar *CSS*. Através da propriedade **font-style**, podemos criar este efeito através dos seguintes valores:

- ⑩ **normal**: O texto é apresentado de forma padrão;
- ⑩ **italic**: O texto é apresentado em itálico;
- ⑩ **oblique**: O texto é apresentado de forma oblíqua. Similar ao itálico, mas menos suportado.

```
p {
  font-style: oblique;
}
```

Figura 190: Exemplo da utilização de font-style.

Oblíquo.

Figura 191: Resultado da Figura 190.

2.2.6.5 Espessura da fonte

Como vimos em *HTML*, as tags `` e `` não devem ser utilizadas para transformar um texto em negrito, quando o único propósito é a estilização. Em vez disso, devemos utilizar *CSS*. Através da propriedade **font-weight**, podemos criar este efeito através dos seguintes valores:

Valor	Descrição
normal	Espessura padrão da fonte. Equivalente a espessura 400.
bolder	Uma espessura maior do que o elemento “pai”.
lighter	Uma espessura menor do que o elemento “pai”.
100	Espessura 100.
200	Espessura 200.
300	Espessura 300.
400	Espessura 400. Equivalente a “normal”.
500	Espessura 500.
600	Espessura 600.
700	Espessura 700. Equivalente a “bold”.
800	Espessura 800.
900	Espessura 900.
bold	Negrito. Equivalente a espessura 700.

Tabela 8: Valores para alteração da espessura da fonte.

Nota: Algumas fontes possuem apenas as apresentações “normal” e “bold”. Por isso, mesmo que você varie o valor numérico, você terá apresentações iguais, como no exemplo da Figura 193 (fonte padrão do Google Chrome versão 70).


```
<p style="font-weight:normal;">Texto</p>
<p style="font-weight:100;">Texto</p>
<p style="font-weight:300;">Texto</p>
<p style="font-weight:500;">Texto</p>
<p style="font-weight:700;">Texto</p>
<p style="font-weight:900;">Texto</p>
```

Figura 192: Exemplos de aplicação da espessura da fonte.

```
p {
  font-weight: 900;
}
```

Figura 194: Exemplo da utilização em CSS interna e externa.

Texto
Texto
Texto
Texto
Texto
Texto
Texto

Figura 193: Resultado da Figura 192.

2.2.6.6 Text-decoration

Como vimos em *HTML*, as tags `<s>`, `<ins>`, `` e `<u>` não devem ser utilizadas para riscarmos um trecho de texto, quando o único propósito é a estilização. Em vez disso, devemos utilizar *CSS*. Através da propriedade **font-weight**, podemos criar este efeito através dos seguintes valores:

Valor	Descrição
none	Texto normal.
overline	Traço acima do texto.
line-through	Traço riscando o texto.
underline	Traço abaixo do texto.

Tabela 9: Valores para text-decoration.

```
<p style="text-decoration:overline;">Texto</p>
<p style="text-decoration:line-through;">Texto</p>
<p style="text-decoration:underline;">Texto</p>
```

Figura 195: Exemplos de text-decoration.

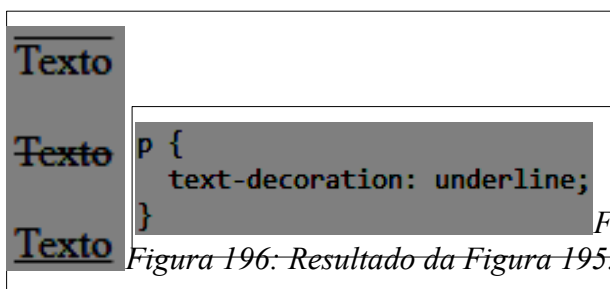


Figura 196: Resultado da Figura 195.

Figura 197: Exemplo para CSS interna e externa.

2.2.6.7 Maiúsculas e minúsculas

Através de CSS, é possível também transformar letras maiúsculas em minúsculas e vice-versa. Para tanto, utilizamos a propriedade **text-transform** com os seguintes valores:

Valor	Descrição
none	Texto normal.
uppercase	Transforma todas as letras em maiúsculas.
lowercase	Transforma todas as letras em minúsculas.
capitalize	Transforma a primeira letra de todas as palavras em maiúsculas.

Tabela 10: Valores para text-transform.

```
<p style="text-transform:uppercase;">Texto de exemplo.</p>
<p style="text-transform:lowercase;">Texto de exemplo.</p>
<p style="text-transform:capitalize;">Texto de exemplo.</p>
```

Figura 198: Exemplos de text-transform.

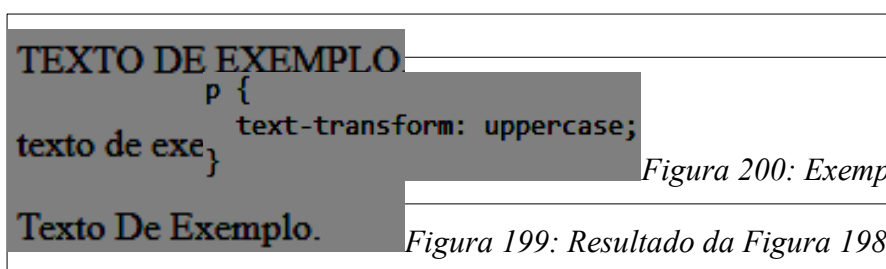


Figura 199: Resultado da Figura 198.

Figura 200: Exemplo para CSS interna e externa.

Similar a este tema, temos a propriedade **font-variant**. Através do valor **small-caps** esta propriedade transforma todas as letras minúsculas em *small-caps*, ou seja, letras maiúsculas, mas que possuem um tamanho menor do que a maiúscula. Em outras palavras, pode-se dizer que a propriedade transforma o texto em “letra de forma”. Veja:

```
<p style="font-variant:normal;">Texto de exemplo.</p>
<p style="font-variant:small-caps;">Texto de exemplo.</p>
```

Figura 201: Exemplo da utilização de font-variant.

Texto de exemplo.
 TEXTO DE EXEMPLO.

Figura 202: Resultado da Figura 201.

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

Figura 203: Exemplo para CSS interna e externa.

Para texto normal, utilize o valor **normal**, como no exemplo acima.

2.2.6.8 Mudando a fonte do texto

As fontes, em CSS, podem ser classificadas da seguinte maneira:

- ⑩ **Font Family**: uma família específica de fonte, como “Times New Roman” ou “Arial”;
- ⑩ **Generic Family**: ou família genérica, em português, que são grupos formados por famílias de aparência similar, podendo ser classificadas em:
 - **Serif**: Fontes serifadas, caracterizadas por pequenos traços ou espessamentos nas extremidades da letra;
 - **Sans-serif**: Fontes não-serifadas. Para a leitura em telas, este tipo de fonte é considerado o mais fácil de ser lido;
 - **Monospace**: Fontes com caracteres que possuem a mesma largura;
 - **Cursive** ou **script**: Fontes com letras cursivas;
 - **Fantasy**: Fontes decorativas, com estilização especial.

Fontes	Serif	Sans-serif	Monospace	Cursive	Fantasy
Exemplo	Times New Roman	Arial	Liberation Mono	Segoe Script	UNIDREPTLED

Tabela 11: Exemplos de fontes genéricas.

Para mudar a fonte do texto através de CSS, utilizamos a propriedade **font-family**, onde colocamos o nome da fonte que queremos em seu valor. Este nome pode vir entre aspas ou não. A exceção é se o nome possuir mais de uma palavra (ex. Times New Roman), aí o nome deve vir entre aspas.

Entretanto, pode ser que a fonte que escolhemos não esteja disponível para todos os sistemas. Cada dispositivo vem com uma seleção de fontes pré-instaladas, de acordo com o sistema operacional principalmente. Como medida de segurança contra este problema, normalmente escrevemos mais de uma fonte nesta propriedade, separando-as por vírgulas. Começamos escrevendo a fonte que desejamos, e terminamos com uma fonte genérica, para que em último caso, após todas as outras opções anteriores terem falhado, o navegador utilizará a fonte padrão para a família genérica que você indicou. Exemplo:

```
p {  
  font-family: "Times New Roman", Times, serif;  
}
```

Figura 204: Mudando a fonte através de CSS.

Este método que você acabou de ver utiliza o que é conhecido, em inglês, por “*Web Safe Fonts*”, ou “**fontes seguras**” em português, e são assim chamadas por proporcionarem uma maior segurança, um maior controle ao desenvolvedor sobre o comportamento de seu documento, ao indicar um grupo de fontes, gerando uma maior compatibilidade entre diferentes sistemas operacionais. Você pode consultar algumas destas fontes aqui: <https://www.cssfontstack.com>.

2.2.6.8.1 Importando fontes externas com @font-face

Contudo, não é preciso se ater somente às fontes seguras. Podemos indicar qualquer fonte que desejamos, mesmo que não estejam disponíveis no sistema do usuário. Para isto, utilizamos a regra **@font-face**. Esta regra importa um arquivo externo de fontes para ser utilizado na página, podendo ser nos formatos .ttf, .otf, .woff, .woff2, .svg e .eot (sempre verifique a compatibilidade com os navegadores). Estas regras iniciadas por “@”, chamadas de **regras atribuídas** (“*at-rules*” ou “*@ rules*”, em inglês), são regras condicionais que instruem a CSS em como se comportar. Várias destas regras podem vir agrupadas no início do arquivo CSS, e se enquadram em uma categoria especial conhecida como **regras de grupos condicionais**, ou *conditional group rules* em inglês. Veja o exemplo a seguir para melhor entendimento:

```
@font-face {  
  font-family: Fonte;  
  src: local("Fonte"),  
       url("fonte.eot?#iefix") format("embedded-opentype"), /* IE6-IE8 */  
       url("fonte.woff2") format("woff2"), /* Navegadores mais modernos */  
       url("fonte.woff") format("woff"), /* Navegadores modernos */  
       url("fonte.ttf") format("truetype"), /* Safari, Android, iOS */  
       url("fonte.svg#svgFontName") format("svg"); /* Antigo iOS */  
}
```

Figura

205: Exemplo da utilização de @font-face.

No exemplo acima, perceba que a primeira propriedade é a **font-family**. O valor de font-family será o valor que você utilizará para referenciar esta fonte, como no exemplo abaixo.

```
p {
  font-family: Fonte;
}
```

Figura 206: Usando a fonte "Fonte".

Em seguida, temos **src**, que indicara o local do arquivo. O primeiro valor para esta propriedade é **local()**, que carregará o arquivo da fonte “Fonte” salvo no sistema do usuário, caso ele já possua instalado em seu sistema. Ao indicarmos **local()** primeiro, não haverá a necessidade de baixar o arquivo novamente, demorando menos para carregar o texto, e consumindo menos a banda de internet do usuário. Mas caso o usuário não possua esta fonte instalada em seu sistema, então será necessário baixá-la. Para isto, temos na linha seguinte **url()**, que indica o local remoto do arquivo. Junto à **url()**, podemos utilizar **format()**, que especifica ao navegador o formato do arquivo, assim ele consegue selecionar o mais adequado. Como os navegadores possuem diferentes compatibilidades, é comum incluirmos vários formatos, como na Figura 166. Os formatos .woff2, .woff e .ttf costumam ser suficientes atualmente, mas se você quiser ser precavido, pode incluir os outros também.

Se você quiser utilizar esta fonte em itálico, por exemplo, você poderá criar uma nova regra **@font-face** para importar o arquivo que contenha a versão em itálico desta fonte. Assim:

```
@font-face {
  font-family: Fonte;
  src: local("Fonte Italic"),
       url("fonte-italic.woff2") format("woff2"),
       url("fonte-italic.woff") format("woff"),
       url("fonte-italic.ttf") format("truetype");
  font-style: italic;
}
```

Figura 207: Importando a versão itálica da fonte.

Desta forma, toda vez que você indicar “font-style: italic” para um texto utilizando a fonte “Fonte”, este outro arquivo que será utilizado, ao invés do arquivo padrão. Assim, você pode ter várias **@font-face** para uma mesma fonte, cada uma com um estilo diferente.

2.2.6.8.2 Importando fontes externas com **@import** e **<link>**

A regra **@import** possui a função de importar folhas de estilo dentro de outra folha de estilo. Também deve vir no topo do documento **CSS**, mas depois de eventuais **@charset**, que não abordaremos por hora. Já a **tag <link>**, como já vimos em **HTML**, é utilizada para criar uma ligação entre o documento **HTML** e arquivos externos. Ambos os casos poderiam ser utilizados para importar fontes externas, da seguinte maneira:

```
<@import url('https://fonts.googleapis.com/css?family=Roboto');>  
<link href='https://fonts.googleapis.com/css?family=Roboto' rel='stylesheet'>
```

Figura 208: Importando fontes com `@import` e `<link>`.

Nos dois exemplos acima, estamos importando a versão simples da fonte “Roboto”, do *site* do Google Fonts (<https://fonts.google.com>). No primeiro exemplo, incluiríamos a *tag* `<link>` em `<head>`. No segundo caso, adicionaríamos a regra `@import` no topo do arquivo *CSS*, como mencionado, ou aninhada no topo da *tag* `<style>`, no documento *HTML*.

Enquanto `@font-face` acaba por ser utilizada, em sua maioria, quando a fonte está hospedada em seu servidor, ou quando a maioria dos usuários provavelmente já possui a fonte instalada em seu sistema, `@import` e `<link>` acabam sendo utilizadas para importar as fontes sem ter de hospedá-las em servidores próprios. A diferença é que você não precisaria escrever o requerimento `@font-face`, isto aconteceria de forma automática, feita através de *APIs* (*Application Programming Interface*, ou Interface de Programação de Aplicativos, em português). Entretanto, isto poderia deixar o carregamento da página um pouco mais lento se utilizando `@import`, já que este permite apenas um *download* por vez, bloqueando *downloads* paralelos, por isso a maioria prefere utilizar `<link>`. Outro detalhe de `@import` é que este não é aceito por navegadores antigos. Isto poderia causar problemas em alguns casos, pois a fonte não seria importada, mas alguns preferem utilizar `@import` justamente para criar estilizações específicas para navegadores mais novos, evitando problemas de compatibilidade com os mais antigos, já que estes não seria capazes de ler a regra `@import`, e a ignorariam. Outra utilização de `@import` é poder importar várias folhas de estilos em uma única folha, e utilizar apenas uma *tag* `<link>` em `<head>` para aplicar a *CSS* ao documento *HTML*.

2.2.6.9 Alinhamento do texto

Podemos determinar o alinhamento de nosso texto através da propriedade **text-align**. Para esta propriedade poderemos utilizar os valores **left** (alinhado à esquerda), **right** (alinhado à direita), **center** (centralizado) e **justify** (justificado).

```
<p>Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et  
dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea  
commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nul  
la pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id  
est laborum.</p>
```

Figura 210: Código HTML.

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

Figura 211: Código CSS.

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Figura 212: Resultado.

Para *CSS inline*, basta introduzir o atributo **style**. Veja:

```
<p style="text-align:justify;">Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.</p>
```

Figura 213: Alternativa para CSS inline.

2.2.6.10 Espaçamento entre linhas

O espaçamento entre linhas pode ser alterado através da propriedade **line-height**. Para isto, temos os valores:

Valor	Descrição
normal	Espaçamento padrão.
“Medida”	Pode-se indicar uma medida (com a unidade de medida) desejada.
“Porcentagem”	O espaçamento será a porcentagem indicada, relativa ao tamanho atual da fonte.
“Número”	O espaçamento será o número (sem a unidade de medida) indicado, multiplicado pelo tamanho atual da fonte. Costuma ser o método mais utilizado.

Tabela 12: Valores para espaçamento entre linhas.


```
<p>Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.</p>
```

```
p {
  line-height: 2.5;
}
```

Figura 214: Código HTML.

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Figura 216: Resultado.

Para *CSS inline*, basta introduzir o atributo **style**. Veja:

```
<p style="line-height: 2.5;">Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.</p>
```

Figura 217: Alternativa para CSS inline.

2.2.6.11 Espaçamento entre palavras

Podemos alterar o espaçamento entre as palavras através da propriedade **word-spacing**, onde colocamos no seu valor, a medida que desejamos. Para espaçamento padrão, utilize o valor **normal**.

```
<p>Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.</p>
```

Figura 218: Código HTML.

```
p {
  word-spacing: 1.5em;
}
```

Figura 219: Código CSS.

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut
 labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco
 laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in
 voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat
 non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Figura 220: Resultado.

Para *CSS inline*, basta introduzir o atributo **style**. Veja:

```
<p style="word-spacing:1.5em;">Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor
incidunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco
laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse
cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui
officia deserunt mollit anim id est laborum.</p>
```

Figura 221: Alternativa para CSS inline.

2.2.6.12 Espaçamento entre letras

Para alterar o espaçamento entre as letras, utilizamos a propriedade **letter-spacing**, onde em seu valor, colocamos a medida desejada. Para espaçamento padrão, utilize o valor **normal**.

```
<p>Texto.</p>
```

Figura 222: Código HTML.

```
p {
  letter-spacing: 0.5em;
```

Text o .

Figura 223: Código CSS.

Figura 224: Resultado.

Para *CSS inline*, basta introduzir o atributo **style**. Veja:

```
<p style="letter-spacing:0.5em;">Texto.</p>
```

Figura 225: Alternativa para CSS inline.

2.2.6.13 Espaço em branco, quebra de linha e quebra de palavra

Através da propriedade **white-space** podemos controlar como os espaços em branco e a quebra de linha se comportarão. Para isto, temos os seguintes valores:

Valor	Descrição
normal	<ul style="list-style-type: none"> ⑩ Padrão; ⑩ Espaços em branco em excesso no código serão transformados em um único; ⑩ Quebra de linha no código não gera quebra de linha na página; ⑩ Quebra de linha na página serão inseridas automaticamente, conforme necessidade.
nowrap	<ul style="list-style-type: none"> ⑩ Espaços em branco em excesso no código serão transformados em um único;

Valor	Descrição
	<ul style="list-style-type: none"> ⓐ Quebra de linha no código não gera quebra de linha na página; ⓐ Quebra de linha na página não serão inseridas automaticamente. É necessária a introdução da <i>tag</i> <code>
</code> para isto.
pre	<ul style="list-style-type: none"> ⓐ Similar a <i>tag</i> <code><pre></code>; ⓐ Espaços em branco em excesso no código serão mantidos; ⓐ Quebra de linha no código gera quebra de linha na página; ⓐ Quebra de linha na página não serão inseridas automaticamente. É necessária a introdução da <i>tag</i> <code>
</code> para isto.
pre-line	<ul style="list-style-type: none"> ⓐ Espaços em branco em excesso no código serão transformados em um único; ⓐ Quebra de linha no código gera quebra de linha na página; ⓐ Quebra de linha na página serão inseridas automaticamente, conforme necessidade.
pre-wrap	<ul style="list-style-type: none"> ⓐ Espaços em branco em excesso no código serão mantidos; ⓐ Quebra de linha no código gera quebra de linha na página; ⓐ Quebra de linha na página serão inseridas automaticamente, conforme necessidade.

Tabela 13: Valores para white-space.

```
<p>Lorem ipsum dolor      sit amet,
    consectetur adipiscing elit,
    sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.</p>
```

Figura

226: Código HTML.

```
p {
    white-space: pre;
}
```

Figura 227: Exemplo da utilização de white-space.

A seguir, os diferentes efeitos:

normal

```

Lorem ipsum dolor sit amet, consectetur
adipiscing elit, sed do eiusmod tempor
incidunt ut labore et dolore magna aliqua.
```

Figura 228: white-space: normal;

nowrap

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

Figura 229: white-space: nowrap;

pre

Lorem ipsum dolor sit amet,
 consectetur adipiscing elit,
 sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

Figura 230: white-space: pre;

pre-line

Lorem ipsum dolor sit amet,
 consectetur adipiscing elit,
 sed do eiusmod tempor incididunt ut labore et
 dolore magna aliqua.

Figura 231: white-space: pre-line;

pre-wrap

Lorem ipsum dolor sit amet,
 consectetur adipiscing elit,
 sed do eiusmod tempor incididunt ut labore
 et dolore magna aliqua.

Figura 232: white-space: pre-wrap;

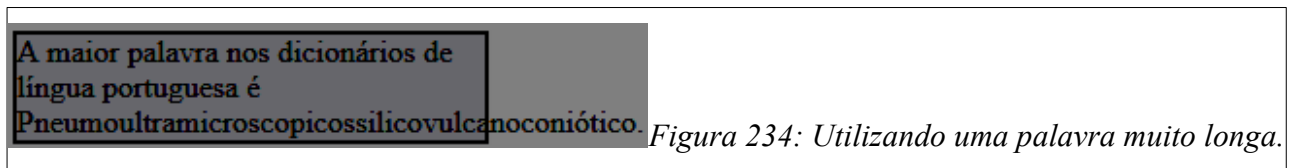
Nota: Foram utilizados outros comandos CSS para a criação da caixa colorida que você vê nos exemplos. Tais códigos serão abordados mais a frente, e estão sendo utilizados agora apenas para uma melhor visualização dos efeitos.

Para CSS *inline*, basta introduzir o atributo **style**. Veja:

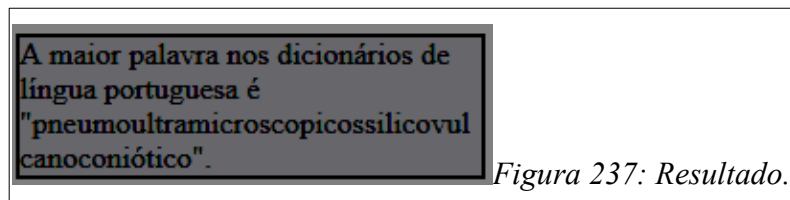
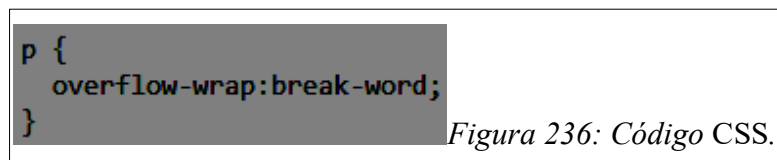
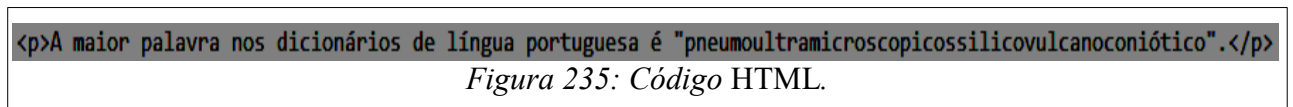
```
<p style="white-space:pre;">Lorem ipsum dolor sit amet,  
consectetur adipiscing elit,  
sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.</p>
```

Figura 233: Alternativa para CSS inline.

Agora, perceba que em nenhum dos casos houve quebra de palavras. Quando as palavras não couberam, elas foram deslocadas para a linha de baixo inteiras. Isto porque, por padrão, a quebra de palavras não ocorre. Observe o caso abaixo:



Acima, temos a maior palavra da língua portuguesa já registrada nos dicionários, que descreve o indivíduo que possui doença pulmonar causada pela inspiração de cinzas vulcânicas. Note que, como não há quebra de palavras, esta palavra gigante acaba ultrapassando o limite, pois a quebra de linha ocorre apenas nos espaços. Através da propriedade **overflow-wrap**, com valor **break-word**, podemos instruir o navegador a quebrar a palavra gigante, jogando o que não couber para a linha de baixo. Veja:



Nota: A propriedade `overflow-wrap` era originalmente chamada `word-wrap`. Como esta era uma extensão da Microsoft, foi criada a `overflow-wrap`. Verifique a compatibilidade com os navegadores antes de utilizar estas propriedades.

O mesmo resultado da Figura 237 pode ser obtido através da propriedade **word-break**, utilizando-se do mesmo valor. A diferença entre estas duas propriedades é que a `word-break` aceita também o valor **break-all**. Perceba que na Figura 237, apesar de a palavra gigante ter sido quebrada, ela ainda foi jogada para a terceira linha. O que “`word-break: break-all`” faz é determinar que haja a quebra de palavras assim que o limite do espaço for atingido, independentemente de onde ocorrer esta quebra. Outra diferença é que “`word-break: break-all`” funciona também para os idiomas mandarim, coreano e japonês, enquanto `overflow-wrap` não. Observe:

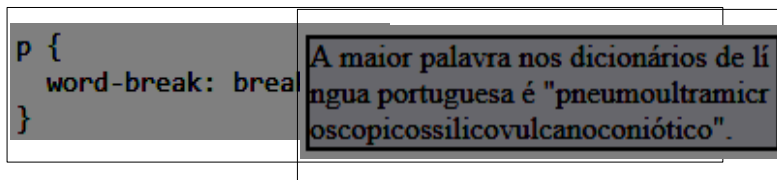


Figura 239: Resultado da Figura 238.

2.2.6.14 Indentação do texto

É possível introduzir um avanço no texto em sua primeira linha do parágrafo. Para tanto, utilizamos a propriedade **text-indent**, que pode ter como valores uma medida desejada, ou uma porcentagem, que será relativa a largura do elemento “pai”. Veja:

```
<p>Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.</p>
```

Figura 240: Código HTML.

```
p {
  text-indent: 3rem;
}
```

Figura 241: Código CSS.

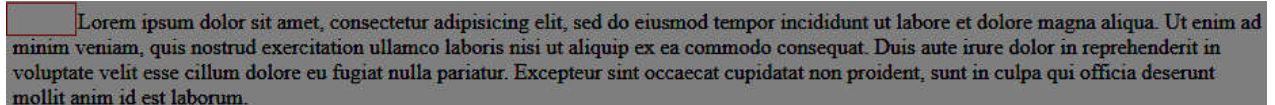


Figura 242: Resultado.

Para *CSS inline*, basta introduzir o atributo **style**. Veja:

```
<p style="text-indent:3rem;">Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.</p>
```

Figura 243: Alternativa para CSS inline.

2.2.7 HIERARQUIA EM CSS

Quando houver mais de um comando CSS sendo aplicado a um elemento, e estes comandos forem conflitantes, haverá uma hierarquia, uma ordem de importância a ser seguida para que haja apenas um resultado final. Veja o exemplo abaixo:

```
body {
  color: blue;
}

p {
  color: red;
}
```

Figura 244: Código CSS.

```
<p>Texto.</p>
```

Figura 245: Código HTML.



Observe que há dois comandos que alteram a cor de “Texto”, mas o resultado final é um texto na cor vermelha. Isto porque a regra para o parágrafo vem depois da regra do body. Agora, veja o exemplo a seguir:

```
body {
  color: blue;
}

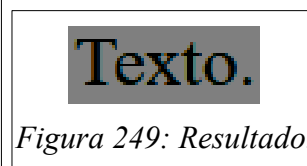
.classe-1 {
  color: green;
}

p {
  color: red;
}
```

Figura 247: Código CSS.

```
<p class="classe-1">Texto.</p>
```

Figura 248: Código HTML.



Repare que desta vez adicionamos uma **classe** ao nosso parágrafo (classe-1), e o resultado foi um texto verde. Isto porque a regra para a classe se sobrepôs às regras do parágrafo e body. Agora, observe o exemplo a seguir:

```
body {  
  color: blue;  
}  
  
#id-1 {  
  color: orange;  
}  
  
.classe-1 {  
  color: green;  
}  
  
p {  
  color: red;  
}
```

classe-1" id="id-1">Texto.</p>

Figura 250: Código HTML.

Texto.

Figura 252: Resultado.

Figura 251: Código CSS.

Note que agora o texto ficou laranja. Isto porque a regra do **id** se sobrepôs às demais. Agora, veja o exemplo a seguir:

```
<p class="classe-1" id="id-1" style="color:purple;">Texto.</p>
```

Figura 253: Código HTML.

```
body {  
  color: blue;  
}  
  
#id-1 {  
  color: orange;  
}  
  
.classe-1 {  
  color: green;  
}  
  
p {  
  color: red;  
}
```

Texto.

Figura 255: Resultado.

Figura 254: Código CSS.

Observe que a cor do texto virou roxa. Isto porque a regra *inline* se sobrepôs às demais. Agora, veja este último exemplo:

```
<p class="classe-1" id="id-1" style="color:purple;">Texto.</p>
```

Figura 256: Código HTML.

```
body {
  color: blue;
}

#id-1 {
  color: orange !important;
}

.classe-1 {
  color: green;
}

p {
  color: red;
}
```

Figura 257: Código CSS.

Texto.

Figura 258: Resultado.

Repare que a adição de **!important** à regra do id fez esta regra se sobrepôr a todas as outras, tornando nosso texto na cor laranja novamente. Devemos utilizar **!important** quando você quiser ter certeza que a regra será aplicada, independentemente de qualquer outra regra em conflito.

2.2.8 FORMATANDO O FUNDO

2.2.8.1 Cor de fundo

Para mudar a cor de fundo, utilizamos a propriedade **background-color**, onde colocamos a cor desejada em seu valor, através de qualquer um dos métodos que você viu no capítulo “2.2.3 Cores”. O valor **transparent** torna o fundo transparente (padrão). Exemplo:

```
<p>Fundo verde.</p>
```

Figura 259: Código HTML.

```
p {
  background-color: #0f9;
}
```

Figura 260: Código CSS.

Fundo verde.

Figura 261: Resultado.

Para *CSS inline*, basta introduzir o atributo **style**. Veja:


```
<p style="background-color: #0f9;">Fundo verde.</p>
```

Figura 262: CSS inline.

2.2.8.2 Utilizando uma imagem como fundo

Podemos utilizar uma ou mais imagens de fundo através da propriedade **background-image**, e utilizarmos os seguintes valores:

Valor	Descrição
url()	Adicionamos a <i>URL</i> da imagem.
none	Sem imagem (padrão).
inherit	Utiliza a mesma imagem do elemento “pai”.
linear-gradient()	Cria um gradiente linear (de cima para baixo) com pelo menos duas cores.
radial-gradient()	Cria um gradiente radial (do centro às extremidades) com pelo menos duas cores.
repeating-linear-gradient()	Cria um gradiente linear repetitivo.
repeating-radial-gradient()	Cria um gradiente radial repetitivo.

Tabela 14: Valores para background-image.

```
body {
  background-image: linear-gradient(green, yellow);
}
```

Figura 263: Exemplo da utilização

de background-image.

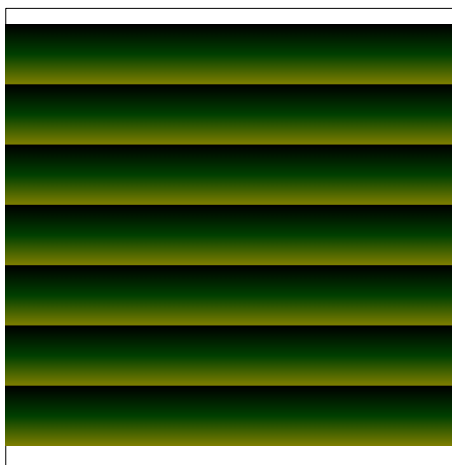


Figura 264: Resultado da Figura 263.

Quando uma imagem for pequena, esta imagem será repetida várias vezes para preencher o fundo. Para controlar este efeito, temos a propriedade **background-repeat** e os seguintes valores:

Valor	Descrição
repeat	Padrão. A imagem é repetida horizontal e verticalmente.
no-repeat	A imagem não é repetida.
repeat-x	A imagem é repetida apenas na horizontal.
repeat-y	A imagem é repetida apenas na vertical.
<i>Tabela 15: Valores para background-repeat.</i>	

```
body {
  background-image: linear-gradient(green, yellow);
  background-repeat: no-repeat;
}
```

Figura 265: Exemplo de utilização de background-repeat.



Figura 266: Resultado da Figura 265.

Podemos controlar se a imagem do fundo ficará fixa ou se moverá junto com o rolamento da página. Para isto, utilizamos a propriedade **background-attachment** com o valor **fixed** (para fixar) e **scroll** (para rolar).

```
body {
  background-image: linear-gradient(green, yellow);
  background-repeat: no-repeat;
  background-attachment: fixed;
}
```

Figura 267: Exemplo da utilização de background-attachment.



Figura 268: Resultado da Figura 267.

Para controlar o tamanho da imagem de fundo temos a propriedade **background-size**, com os seguintes valores:

Valor	Descrição
auto auto	Padrão. Tamanho original.
auto	Tamanho original.
“Tamanho” auto	Adiciona-se o tamanho desejado para a largura, e a altura será proporcional a largura.
auto “Tamanho”	Adiciona-se o tamanho desejado para a altura, e a largura será proporcional a altura.
“Tamanho” “Tamanho”	Adiciona-se o tamanho desejado para a largura e altura, respectivamente.
“Porcentagens”	Adiciona-se as porcentagens para a largura e altura, respectivamente. As porcentagens serão relativas a área do elemento “pai”. Colocar apenas um valor, torna o outro automaticamente “auto”.
“Porcentagem” auto	Adiciona-se a porcentagem para a largura, relativa a área do elemento “pai”, enquanto a altura é proporcional a largura.
auto “Porcentagem”	Adiciona-se a porcentagem para a altura, relativa a área do elemento “pai”, enquanto a largura é proporcional a altura.

auto “Tamanho”	Adiciona-se o tamanho desejado para a altura, e a largura será proporcional a altura.
cover	A imagem é redimensionada para o maior tamanho possível, para que ela seja totalmente visualizada, mesmo que a imagem ultrapasse o limite da área do conteúdo.
contain	A imagem é redimensionada para o maior tamanho possível, para que ela seja totalmente visualizada, mantendo sempre a proporção.
<i>Tabela 16: Valores para background-size.</i>	

```
body {
  background-image: linear-gradient(green, yellow);
  background-repeat: no-repeat;
  background-attachment: fixed;
  background-size: auto;
}
```

Figura 269: Exemplo da utilização de background-size.

Para controlar a posição da imagem, temos a propriedade **background-position**. Os valores são:

Valor	Descrição
left top	No topo, à esquerda.
center top	No topo, centralizado horizontalmente.
right top	No topo, à direita.
left center	Centralizado verticalmente, à esquerda.
center center	Centralizado.
right center	Centralizado verticalmente, à direita.
left bottom	Em baixo, à esquerda.
center bottom	Em baixo, centralizado horizontalmente.
right bottom	Em baixo, à direita.
left	Centralizado verticalmente, à esquerda.
center	Centralizado.
right	Centralizado verticalmente, à direita.
top	No topo, centralizado horizontalmente.
bottom	Em baixo, centralizado horizontalmente.
“Porcentagens”	Adiciona-se a porcentagem desejada que será relativa ao canto superior esquerdo.
“Distância”	Adiciona-se a distância desejada, relativas ao canto superior esquerdo.
<i>Tabela 17: Valores para background-position.</i>	

```
body {
  background-image: linear-gradient(green, yellow);
  background-repeat: no-repeat;
  background-attachment: fixed;
  background-size: auto;
  background-position: center;
}
```

Figura 270: Exemplo da utilização de `background-position`.

2.2.9 BORDAS

Podemos adicionar bordas aos elementos ao adicionarmos algumas propriedades.

2.2.9.1 Estilo da borda

Através da propriedade **`border-style`** podemos definir o estilo da borda. Podemos adicionar até quatro valores, os quais seguirão a ordem: borda superior, direita, inferior e esquerda. Os tipos de borda são:

Valor	Descrição
dotted	Pontilhada.
dashed	Tracejada.
solid	Linha contínua.
double	Linhas duplas.
inset	Efeito que simula um “afundamento” da área do elemento.
outset	Efeito que simula uma “elevação” da área do elemento.
groove	Efeito que simula um “afundamento” da borda.
ridge	Efeito que simula uma “elevação” da área da borda.
none	Nenhuma.
hidden	Omissa.

Tabela 18: Valores para `border-style`.

```
p {
  border-style: solid;
}
```

Figura 271: Exemplo de utilização de `border-style`.

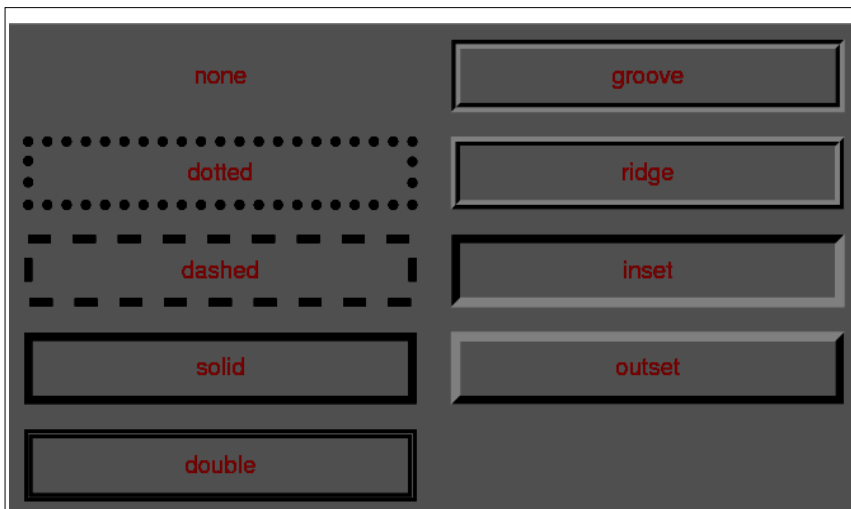


Figura 272: Diferentes estilos de borda. Fonte: W3C.

2.2.9.2 Largura da borda

Para controlar a largura da borda podemos utilizar a propriedade **border-width**, com os valores:

Valor	Descrição
thick	Borda espessa.
medium	Padrão. Borda mediana.
thin	Borda fina.
“Medida”	Podemos definir uma medida desejada.

Tabela 19: Valores para border-width.

```
p {
  border-style: solid;
  border-width: thick;
}
```

Figura 273: Exemplo de utilização de border-width.

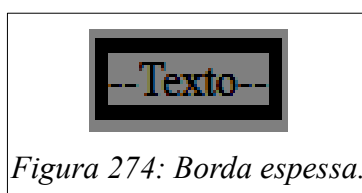


Figura 274: Borda espessa.

2.2.9.3 Cor da borda

Através da propriedade **border-color** podemos estipular uma cor para a borda, através de qualquer um dos métodos que você viu no capítulo “2.2.3 Cores”. O valor **transparent** determina uma borda transparente (padrão).

```
p {  
  border-style: solid;  
  border-width: thick;  
  border-color: red;  
}
```

Figura 275: Exemplo da utilização de border-color.

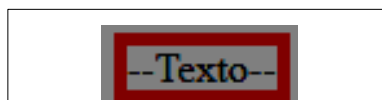


Figura 276: Borda vermelha.

2.2.9.4 Bordas arredondadas

Através da propriedade **border-radius** podemos criar bordas arredondadas, colocando no valor, uma medida. Quanto maior a medida, mais arredondada a borda. Podemos colocar até quatro valores, que seguirão a ordem: canto superior esquerdo, superior direito, inferior direito, e inferior esquerdo. Podemos também adicionar porcentagens nos valores (100% cria um círculo completo).

```
p {  
  border-style: solid;  
  border-width: thick;  
  border-color: red;  
  border-radius: 20%;  
}
```

Figura 277: Exemplo da utilização de border-radius.

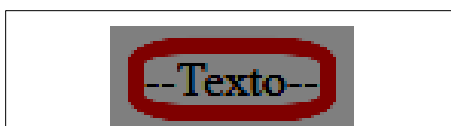
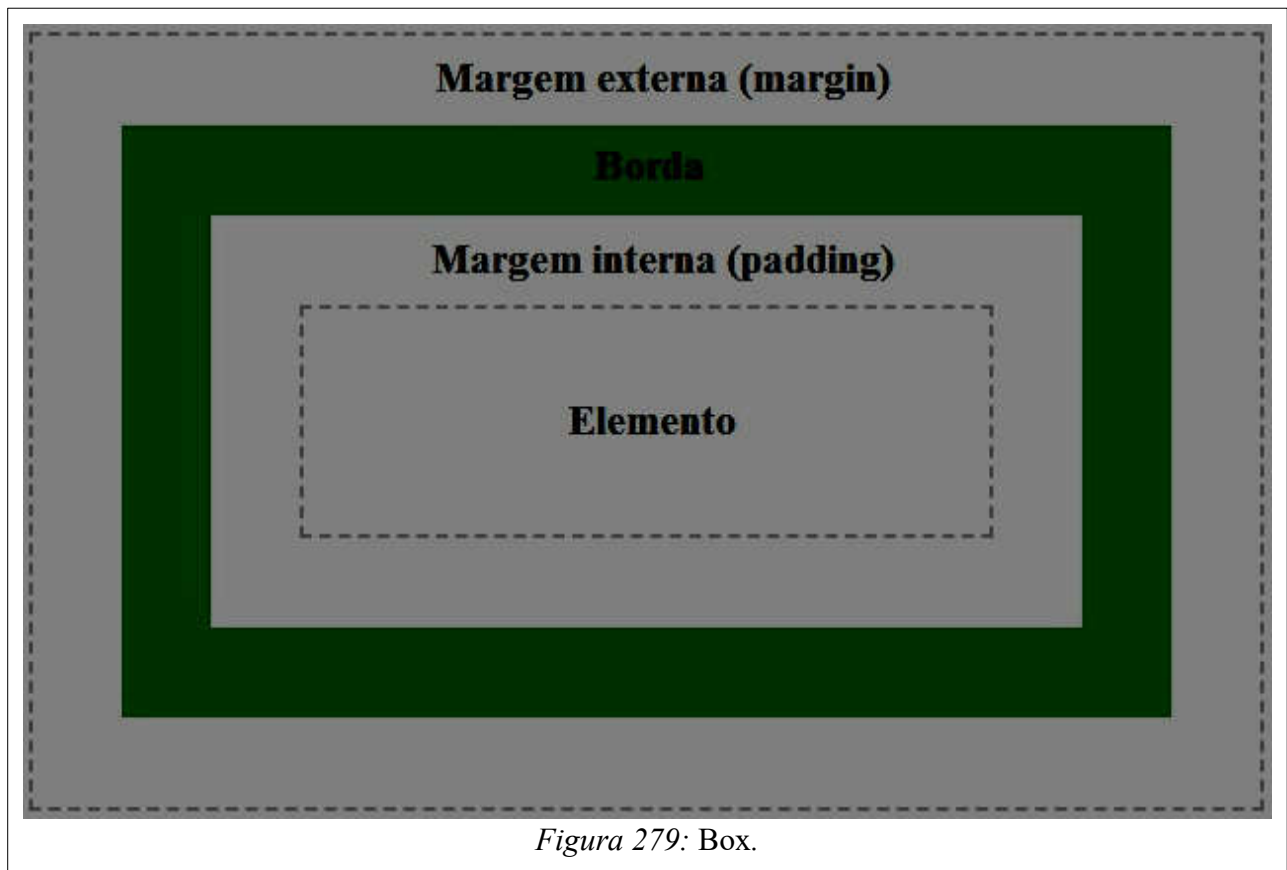


Figura 278: Bordas arredondadas.

2.2.10 BOXES E MARGENS

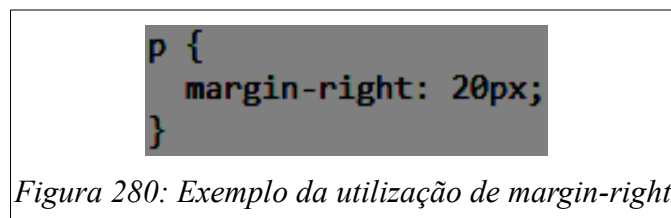
Cada elemento *HTML* cria uma “caixa” imaginária ao redor de si, ou *boxes*. O conteúdo do elemento fica dentro destas “caixas”.



Podemos alterar o tamanho das margens através de algumas propriedades.

2.2.10.1 Margem externa

Para controlarmos o tamanho da margem externa, temos as propriedades **margin-left** (margem externa esquerda), **margin-right** (margem externa direita), **margin-top** (margem externa superior) e **margin-bottom** (margem externa inferior). No valor, podemos introduzir uma medida (o padrão é 0px), ou podemos adicionar uma porcentagem, que é uma porcentagem da largura do elemento “pai”.



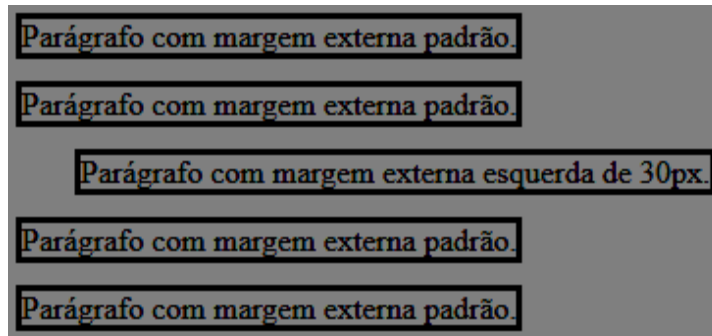


Figura 281: Exemplo de alteração da margem externa.

Para alterar mais de uma margem externa, você pode utilizar a propriedade **margin**, e adicionar até quatro valores, que seguirá a ordem: margem externa esquerda, superior, direita e inferior.

```
p {
  margin: 20px 30px 10px 30px;
}
```

Figura 282: Exemplo da utilização de margin.

2.2.10.2 Margem interna

Para controlarmos o tamanho da margem interna, temos as propriedades **padding-left** (margem interna esquerda), **padding-right** (margem interna direita), **padding-top** (margem interna superior) e **padding-bottom** (margem interna inferior). No valor, podemos introduzir uma medida (o padrão é 0px), ou podemos adicionar uma porcentagem, que é uma porcentagem da largura do elemento “pai”.

```
p {
  padding-right: 20px;
}
```

Figura 283: Exemplo da utilização de padding-right.

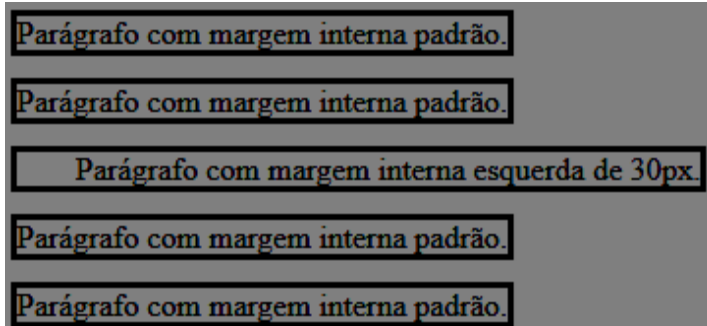


Figura 284: Exemplo de alteração da margem interna.

Para alterar mais de uma margem interna, você pode utilizar a propriedade **padding**, e adicionar até quatro valores, que seguirá a ordem: margem interna esquerda, superior, direita e inferior.

```
p {
  padding: 20px 30px 10px 30px;
}
```

Figura 285: Exemplo da utilização de padding.

2.2.10.3 Sombra

Assim como podemos adicionar sombras ao texto, podemos adicionar sombras as *boxes* através da propriedade **box-shadow**. Para os valores, podemos adicionar cinco valores, que representam respectivamente o deslocamento horizontal da sombra, o deslocamento vertical, o desfoco, o tamanho da sombra e sua cor ao fim.

```
p {
  box-shadow: 0 0 10px 3px red;
}
```

Figura 286: Exemplo da utilização de box-shadow.

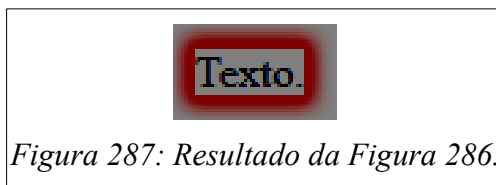


Figura 287: Resultado da Figura 286.

2.2.11 LARGURA, ALTURA E OVERFLOW

Através das propriedades **width** e **height** podemos alterar, respectivamente, a largura e altura de um elemento com os valores:

Valor	Descrição
auto	Padrão. O navegador calcula a medida.
“Medida”	Podemos definir uma medida desejada.
“Porcentagem”	Podemos definir uma porcentagem desejada.
<i>Tabela 20: Valores para width e height.</i>	

```
p {
border-style: solid;
width: 100px;
height: 150px;
}
```

Figura 288: Exemplo da utilização de height e width.

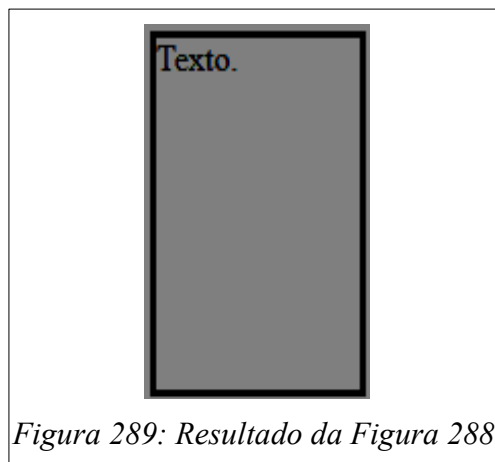


Figura 289: Resultado da Figura 288.

Podemos determinar um mínimo e máximo para a largura e altura, através das propriedades abaixo. Estas propriedades previnem que o valor de height e width ultrapassem o valor estipulado para máximo e mínimo. Os valores aceitos são os mesmos de height e width.

Valor	Descrição
min-width	Determina uma largura mínima.
max-width	Determina uma largura máxima.
min-height	Determina uma altura mínima.
max-height	Determina uma altura máxima.
<i>Tabela 21: Larguras e alturas mínimas e máximas.</i>	

```
p {
  border-style: solid;
  max-width: 500px;
  width: 15%;
  min-width: 50px;
  max-height: 500px;
  height: 15%;
  min-height: 50px;
}
```

Figura 290: Exemplo da utilização de máximos e mínimos para altura e largura.

Se o conteúdo for menor que o mínimo, o valor do mínimo será aplicado. Se o valor do conteúdo for maior que o máximo, o valor de máximo será aplicado, e o que acontecerá com o conteúdo excedente, será determinado pela propriedade **overflow**, que vem acompanhada dos seguintes valores:

Valor	Descrição
visible	Padrão. O conteúdo excede o limite da <i>box</i> .
hidden	O conteúdo excedente é “cortado” e não é exibido.
scroll	Cria-se uma barra de rolagem para caber todo o conteúdo, independente de o conteúdo ser “cortado” ou não.
auto	Cria-se uma barra de rolagem para caber todo o conteúdo, se houver conteúdo que foi “cortado”.

Tabela 21: Valores para *overflow*.

```
p {
  border-style: solid;
  max-width: 500px;
  width: 15%;
  min-width: 50px;
  max-height: 500px;
  height: 15%;
  min-height: 50px;
  overflow: auto;
}
```

Figura 291: Exemplo da utilização de *overflow*.

3. INTRODUÇÃO À ACESSIBILIDADE

Alguns usuários possuem certas limitações que requerem alguns cuidados dos desenvolvedores para que haja a inclusão de todos. O *Web Content Accessibility Guidelines (WCAG)*, ou Diretrizes de Acessibilidade para Conteúdo Web, em português, traz uma série de orientações que auxiliam nesta tarefa. Algumas destas preocupações já foram discutidas ao longo dos capítulos anteriores, mas citaremos mais algumas agora. Você pode consultar todas elas no *site* da W3C

(<https://www.w3.org/Translations/WCAG20-pt-PT/>). Existem várias ferramentas na web que ajudam os desenvolvedores a verificar se seus sites estão acessíveis a todos.

3.1 CORES

Uma das preocupações é escolher cores que possuam um contraste adequado entre a cor do texto e o fundo. Cores com contraste muito próximos podem ser um problema para todos os usuários, em especial, para aqueles com limitações visuais. O *WCAG* recomenda um contraste de pelo menos 4,5:1, em uma escala que varia entre 1:1 (mesma cor) e 21:1 (preto e branco).

O daltonismo, também conhecido como discromatopsia ou discromopsia, é uma condição que afeta a percepção de cores. Existe mais de um tipo de daltonismo, sendo o mais comum o que reduz a sensibilidade de detecção da cor verde. Assim, utilizar dois tons de verde juntos, pode causar dificuldade de visualização. Algumas ferramentas *online* permitem a simulação de como pessoas daltônicas enxergariam seu *site*.

3.2 ATALHOS E FOCO

Alguns usuários utilizam apenas o teclado para navegar. O atributo **accesskey** permite o uso de atalhos para navegar pelo *site*. Este atributo pode ser utilizado para qualquer elemento, mas é particularmente útil quando utilizado junto de elementos interativos, como *links*, botões e formulários. Ao habilitar as accesskeys, quando o usuário aperta a tecla de atalho definida, o elemento que possuir a accesskey receberá o foco. Cada navegador possui um jeito diferente de ativar as accesskeys, mas é sempre bom informar ao usuário sobre a presença de accesskeys em seu site, a fim de evitar acionamentos acidentais. Se for utiliza-los, procure utilizar caracteres comuns a maioria dos idiomas, e que não entre em conflito com atalhos já existentes e comuns aos navegadores.

```
<a href="http://www.digicad.com.br" accesskey="d">Digicad</a>
```

Figura 292: Exemplo da utilização de accesskey.

O atributo **tabindex** permite que elementos recebam o foco do usuário ao apertar a tecla “Tab”. Alguns elementos, como *links* e formulário já possuem por padrão este recurso. O atributo aceita números inteiros como valor, com diferentes resultados:

- ⑩ **valores negativos:** O elemento recebe o foco, mas não pode ser acessado através de teclas de navegação, e são utilizados para criar *widgets* acessíveis através de JavaScript;
- ⑩ **valor zero:** Determina que o elemento deve receber foco, mas deixa a ordem do foco ser definida pelo documento;
- ⑩ **valores positivos:** Criam uma ordem. O “1” receberá o foco antes do “2”, que receberá o foco antes de “3” e assim por diante. Pode ser útil em alguns casos, porém, dependendo de como você utilizar, pode causar confusão na sequência de navegação, por isso tente evitar.

```
<p tabindex="0">Tenhamos foco!</p>
```

Figura 293: Exemplo da utilização de tabindex.

4. INTRODUÇÃO À BOOTSTRAP E DESIGN RESPONSIVO

Bootstrap é um *framework* gratuito e open-source (código fonte aberto) de *front-end* muito popular usado para criar páginas e aplicações web responsivas. Ele inclui estilos e classes CSS pré-definidos e funcionalidades JavaScript. Também pode dividir o layout verticalmente em até 12 colunas (sistema *grid*), de maneira responsiva para o uso de:

- Botões
- Imagens
- Tabelas
- Modais
- Formulários
- Menu de navegação

Para adicionar o Bootstrap ao seu projeto, basta usar, dentro de sua *tag* **<head>**, o código:

```
<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/bootstrap.min.css" integrity="sha384-Gn5384xqQ1aoWXA+058RXPxPg6fy4IWvTNh0E263XmFcJISAwIGgFAW/dAiS6JXm" crossorigin="anonymous">
```

A documentação oficial do *framework* pode ser encontrada no link: <https://getbootstrap.com/>.

4.1 LAYOUT DE COLUNAS (GRID) RESPONSIVO

4.1.1 COLUNAS BÁSICAS (CLASSE “COL”)

Uma das principais funções do Bootstrap é facilitar a criação de colunas de forma responsiva na sua página *HTML*. Abaixo encontra-se um exemplo de código utilizando a classe do Bootstrap “**container**”, em uma *tag* **<div>** englobando todo o código, para transformar o layout em **responsivo**, em seguida a classe “**row**”, que é responsável por englobar as classes de colunas “**col**”.

```
<div class="container">
  <h5>Colunas Básicas</h5>
  <div class="row">
    <div class="col">coluna</div>
    <div class="col">coluna</div>
    <div class="col">coluna</div>
  </div>
</div>
```

Colunas Básicas		
coluna	coluna	coluna

Figura 294: Resultado de colunas básicas.

Neste exemplo todas as colunas possuem o mesmo tamanho, pois a classe utilizada (“col”) não especificou o tamanho de cada uma. Logo, estas colunas utilizarão todo o tamanho disponível na página e cada uma ocupará 1/3 do tamanho da tela (que foi limitado pela classe container).

4.1.2 COLUNAS COM TAMANHO DEFINIDO (CLASSE “COL-[1-12]”)

O segundo exemplo da função “colunas” é utilizar colunas com tamanhos diferentes. Para fazer isso, devemos colocar um valor entre 1 e 12 depois da palavra “col”, como exemplo: “col-6”. Neste caso a coluna ocupará metade da tela (pois seu tamanho é dividido em 12). Para um *layout* padronizado, em muitos casos a soma do tamanho das colunas é 12. Assim toda a tela será ocupada, sem deixar espaços em branco, como no exemplo abaixo:

```
<div class="container">
  <h5>Tamanhos Definidos</h5>
  <div class="row">
    <div class="col-md-6">coluna (6/12)
      <div style="border: none;">
        
      </div>
    </div>
    <div class="col-md-4">coluna (4/12)
      <h5>Exemplo de um título usando colunas Bootstrap</h5>
      <p>Exemplo de um parágrafo usando colunas Bootstrap</p>
    </div>
    <div class="col-md-2">coluna (2/12)
      <div style="border: none;">
        <i class="fas fa-columns fa-2x"></i>
      </div>
    </div>
  </div>
</div>
```



Tamanhos Definidos		
coluna (6/12)	coluna (4/12)	coluna (2/12)
	Exemplo de um título usando colunas Bootstrap Exemplo de um parágrafo usando colunas Bootstrap	

Figura 295: Resultado colunas com tamanho definido.

Como pode ser observado, a primeira coluna ocupou metade do tamanho disponível, a segunda 1/3 e a terceira o restante (1/6). Este exemplo mostra que apesar do Bootstrap fazer a divisão da página verticalmente, ainda precisamos terminar a parte visual, utilizando cores, margens, *padding*, tamanhos, etc.

4.1.3 COLUNAS COM TAMANHO RESPONSIVO (CLASSE “COL-SM/MD/LG/XL-[1-12]”)

O exemplo acima mostra como a tela será dividida, porém ela ainda não faz isso de forma responsiva. Por exemplo, se abrirmos ela usando um celular, não haverá nenhuma mudança e as colunas poderão não ficar muito legíveis (principalmente as menores).

Para resolver este problema, o Bootstrap fornece o recurso de quebra de colunas de acordo com o tamanho do dispositivo, isto é, se a tela de visualização for, por exemplo, de um celular e queremos que as colunas deixem de ser mostradas na mesma linha para melhorar a visualização, as mesmas serão exibidas uma em cada linha.

Para isto, utilizaremos as classes: “**sm**” (*small*) para casos de quebra de colunas apenas para celulares (telas menores de 576 pixels), “**md**” (*medium*) para dispositivos com telas menores de 768 pixels, geralmente tablets, “**lg**” (*large*) para laptops pequenos com telas menores de 992 pixels e “**xl**” para telas menores de 1200 pixels. Lembrando que quando não quisermos a quebra de colunas em nenhum caso, basta não utilizar nenhuma *tag*, como: “col-5”.

Usando o mesmo código do exemplo acima modificando apenas as classes col-x para col-md-x conforme abaixo, temos os resultados:

```
<div class="container">
  <h5>Tamanhos Definidos</h5>
  <div class="row">
    <div class="col-md-6">coluna (6/12)
      <div style="border: none;">
        
      </div>
    </div>
    <div class="col-md-4">coluna (4/12)
      <h5>Exemplo de um título usando colunas Bootstrap</h5>
      <p>Exemplo de um parágrafo usando colunas Bootstrap</p>
    </div>
    <div class="col-md-2">coluna (2/12)
      <div style="border: none;">
        <i class="fas fa-columns fa-2x"></i>
      </div>
    </div>
  </div>
</div>
```

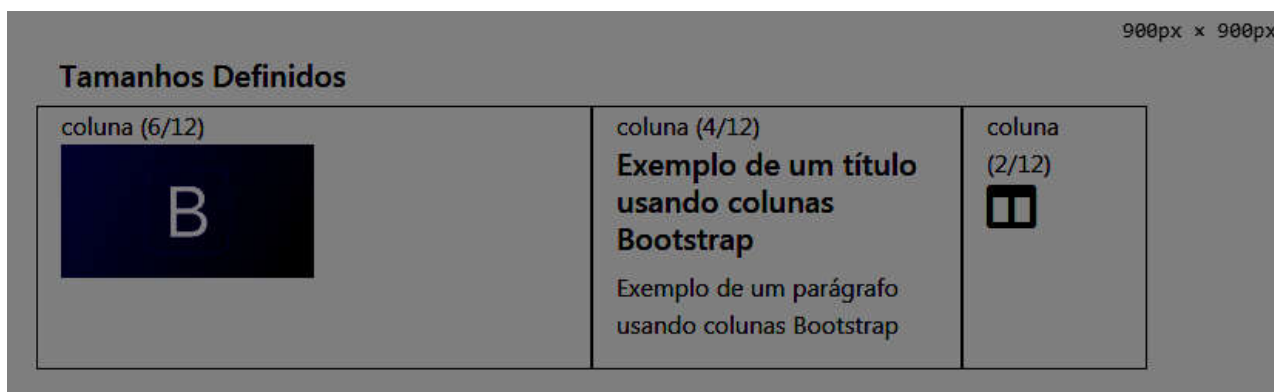



Figura 296: Resultado para uma tela de 900 pixels.

Podemos observar que não houve quebras de colunas para uma tela de 900 pixels de largura (geralmente um pequeno notebook).

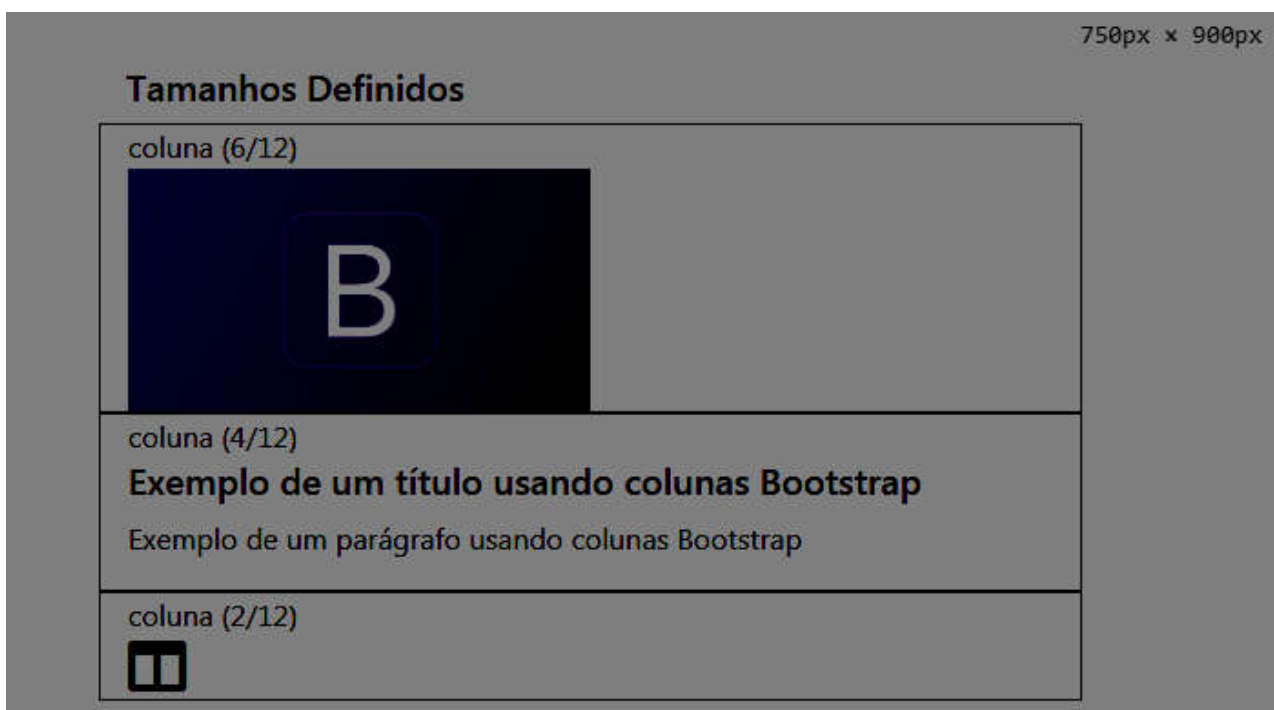


Figura 297: Resultado para uma tela de 750 pixels.

Agora, em uma tela de 750 pixels, podemos observar a quebra de colunas.

Este conceito é fundamental, pois com apenas um único código podemos criar um design responsivo para diversas formas de visualização.

4.1.4 OUTRAS APLICAÇÕES COM A CLASSE COL

Conforme citado anteriormente, o Bootstrap é uma ferramenta muito poderosa e pode-se fazer diversos tipos de aplicações com ele. Agora serão mostrados alguns outros exemplos com a classe col.

4.1.4.1 Colunas sem conteúdo (offset)

No Bootstrap é possível criar colunas sem conteúdo através da classe **offset**, utilizada em conjunto e de acordo com a classe **col** existente no divisor. Por exemplo, para uma cola de tamanho 3/12 com a classe “sm” para quebra de dispositivos celulares temos o seguinte exemplo:

```
<div class="col-md-3 offset-md-3"></div>
```

Para exemplificar melhor iremos analisar o seguinte trecho de código:

```
<div class="container">
  <h5>Colunas Sem Conteúdo</h5>
  <div class="row">
    <div class="col-md-4">.col-md-4</div>
    <div class="col-md-4 offset-md-4">.col-md-4 .offset-md-4</div>
  </div>
  <div class="row">
    <div class="col-md-3 offset-md-3">.col-md-3 .offset-md-3</div>
    <div class="col-md-3 offset-md-3">.col-md-3 .offset-md-3</div>
  </div>
  <div class="row">
    <div class="col-md-6 offset-md-3">.col-md-6 .offset-md-3</div>
  </div>
</div>
```

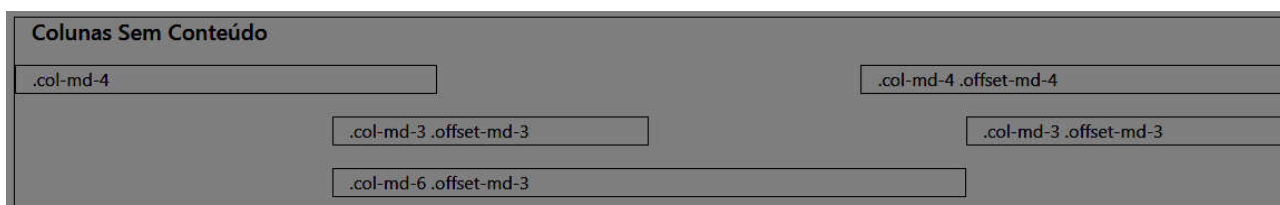


Figura 298: Resultado colunas sem conteúdo.

É possível observar que no primeiro exemplo temos a primeira coluna de tamanho 4, em sequência uma coluna vazia e outra coluna de tamanho 4. A coluna vazia virá sempre à frente de quem a implementa, assim, no segundo exemplo as duas divisórias têm um espaço vazio antes de seus conteúdos.

Por último, é possível ver como este recurso pode ser usado para centralizar o conteúdo com um tamanho definido de forma simples.

4.1.4.2 Alinhamento de colunas (classe justify-content-x)

Outra ferramenta poderosa do Bootstrap é o alinhamento horizontal do conteúdo das classes “col”. O alinhamento pode ser feito com 5 classes: “**justify-content-start**” deixando as classes “col” juntas, no começo da linha, “**justify-content-end**” deixando as classes “col” juntas, no final da linha, “**justify-content-center**” deixando as classes “col” juntas e centralizadas, “**justify-content-around**”

centralizando o conteúdo, porém deixando espaços em branco no começo e entre eles; e por último, “**justify-content-between**” separando as classes “col” com espaços em branco apenas entre elas.

Abaixo podemos observar os comportamentos descritos:

```
<div style="border: 1px solid black" class="container">justify-content-start
  <div class="row justify-content-start">
    <div class="col-4">
      Uma de duas colunas
    </div>
    <div class="col-4">
      Uma de duas colunas
    </div>
  </div>
justify-content-center
<div class="row justify-content-center">
  <div class="col-4">
    Uma de duas colunas
  </div>
  <div class="col-4">
    Uma de duas colunas
  </div>
</div>
justify-content-end
<div class="row justify-content-end">
  <div class="col-4">
    Uma de duas colunas
  </div>
  <div class="col-4">
    Uma de duas colunas
  </div>
</div>
justify-content-around
<div class="row justify-content-around">
  <div class="col-4">
    Uma de duas colunas
  </div>
  <div class="col-4">
    Uma de duas colunas
  </div>
</div>
justify-content-between
<div class="row justify-content-between">
  <div class="col-4">
    Uma de duas colunas
  </div>
  <div class="col-4">
    Uma de duas colunas
  </div>
</div>
```

</div>

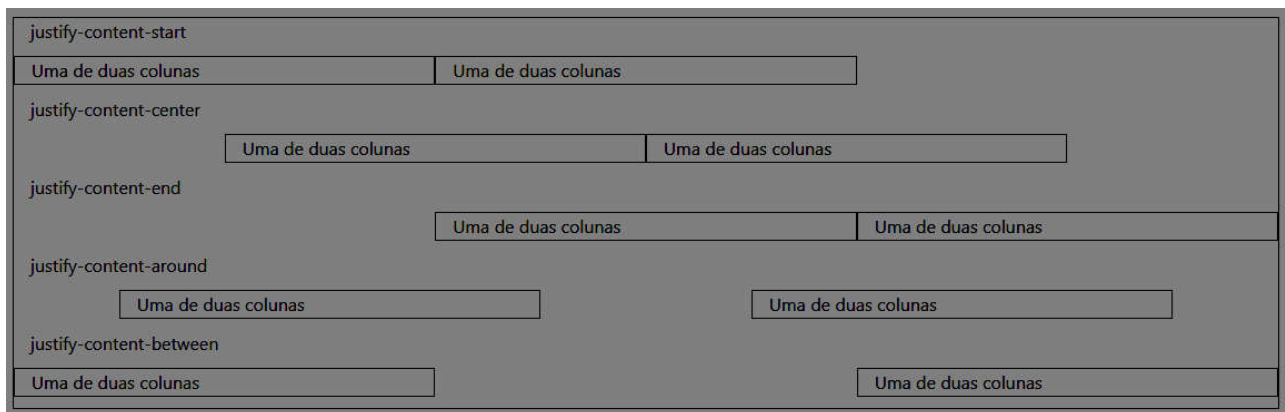


Figura 299: Resultado dos 5 comportamentos de alinhamento horizontal.

4.1.4.3 Colunas aninhadas (nested columns)

Colunas aninhadas é o conceito de colunas dentro de colunas, isto pode ser muito útil quando temos uma coluna com um tamanho grande e seu conteúdo pode ser dividido uma ou mais vezes, tornando o conteúdo em profundidades. Conforme o código abaixo:

```
<div class="container">
  <div class="row">
    <div class="col-sm-3">
      Nível 1: .col-sm-3
    </div>
    <div class="col-sm-9">
      Nível 1: .col-sm-9
      <div class="row">
        <div class="col-sm-6">
          Nível 2: .col-sm-6
          <div class="row">
            <div class="col-sm-6">
              Nível 3: .col-sm-6
            </div>
            <div class="col-sm-6">
              Nível 3: .col-sm-6
            </div>
          </div>
        </div>
        <div class="col-sm-6">
          Nível 2: .col-sm-6
        </div>
      </div>
    </div>
  </div>
</div>
```

Nível 1: .col-sm-3	Nível 1: .col-sm-9	
	Nível 2: .col-sm-6	Nível 2: .col-sm-6
	Nível 3: .col-sm-6	Nível 3: .col-sm-6

Figura 300: Resultado de colunas em mais de uma camada.

4.2 BARRA DE NAVEGAÇÃO (NAVBAR) RESPONSIVO

Outra poderosa ferramenta do Bootstrap são as classes para criar uma barra de navegação responsiva. Esta barra de navegação é muito presente em diversos *sites*, como por exemplo o *site* para *download* do Visual Studio Code:

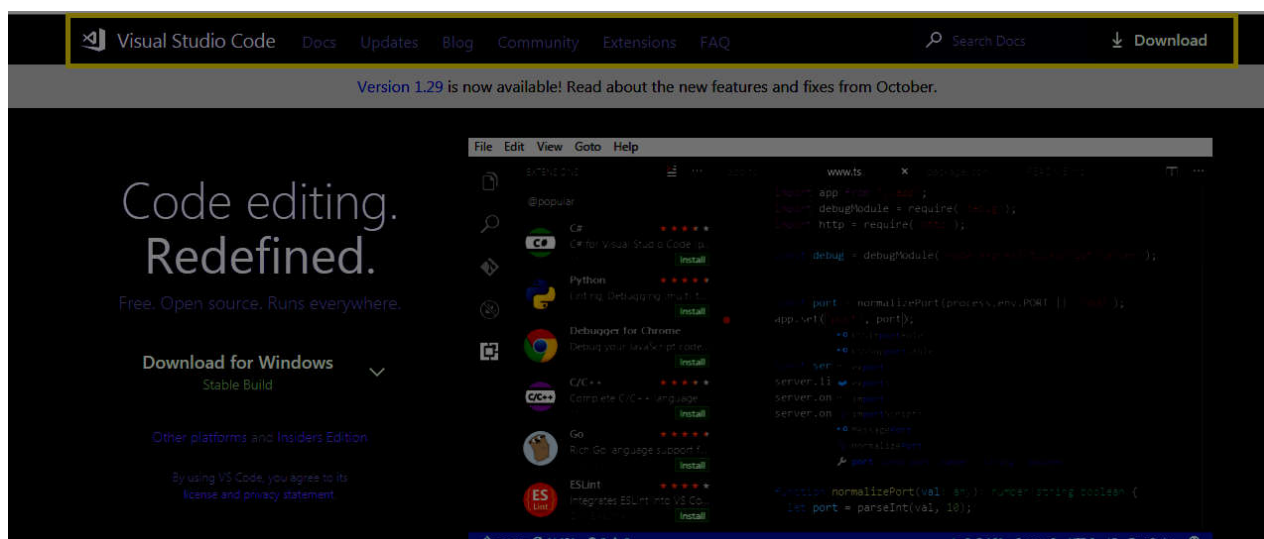


Figura 301: Exemplo de uma barra de navegação (janela grande).

Entretanto, o que aconteceria se o espaço da tela não coubesse todas as opções? A resposta para isto é agrupar os links da barra em um menu que pode ficar oculto ou expandido, conforme os exemplos abaixo.

Opções ocultas:

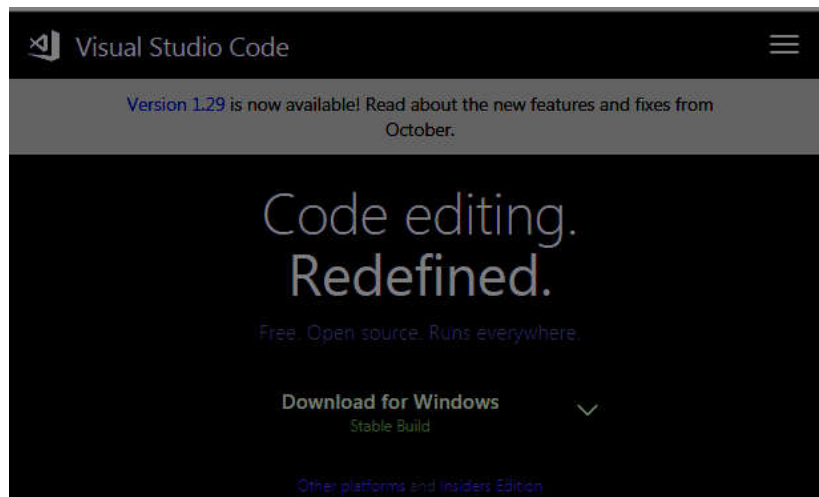


Figura 302: Exemplo de uma barra de navegação oculta em uma tela menor.

Opções visíveis:

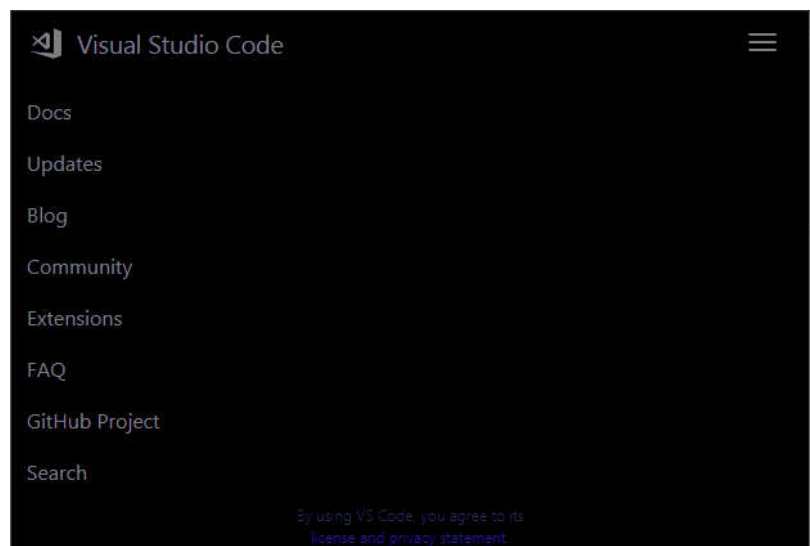


Figura 303: Exemplo de uma barra de navegação aberta em uma tela menor.

Para criarmos a nossa própria barra de navegação, começaremos utilizando a tag `<nav>` com 3 classes do Bootstrap: “**navbar**”, “**navbar-dark**” e “**bg-dark**”, englobando uma tag `<a>` (âncora) com uma referência vazia, como exemplo:

```
<nav class="navbar navbar-dark bg-dark">
  <a href="#">Meu Portfolio</a>
</nav>
```



Figura 304: Barra de navegação inicial.

Pode-se observar que a barra de navegação foi criada com a primeira *tag* âncora. Neste exemplo, utilizamos as classes “**navbar-dark**” e “**bg-dark**” para deixar a barra em um tom escuro e facilitar a sua exemplificação. Porém, existem outras classes do Bootstrap para o tema de sua barra, como: “**navbar-light**” e “**bg-light**”, ou “**navbar-dark**” e “**bg-primary**” (mais exemplos na documentação do site Bootstrap).

Apesar de termos uma primeira ideia de como a barra de navegação será montada, ainda precisamos adicionar outras classes e uma lista de links com a *tag* (lista desordenada). Conforme abaixo:

```
<nav class="navbar navbar-dark bg-dark">
  <a href="#" class="navbar-brand">Meu Portfolio</a>
  <ul class="navbar-nav">
    <li class="navbar-item">
      <a href="#" class="nav-link">Início</a>
    </li>
  </ul>
</nav>
```



Figura 305: Barra de navegação com o primeiro link.

No código acima, foi adicionada a classe “**navbar-brand**” à nossa *tag* <a>, pois se trata do título da página, e automaticamente a cor foi ajustada pelo Bootstrap para uma melhor visualização. Em sequência, foi preciso utilizar a classe “**navbar-nav**” para nossa lista desordenada de links, para cada item da lista utilizar a classe “**navbar-item**”, e para cada link <a>, a classe “**nav-link**”.

Para o primeiro projeto, criaremos uma página de portfólio, então em vez de “Meu Portfolio” será utilizado o nome do aluno, no caso dos exemplos “Instrutor Digicad” e em seguida 3 tópicos da lista com os valores: “Sobre”, “Portfólio” e “Contato”:

```
<nav class="navbar navbar-dark bg-dark">
  <a href="#" class="navbar-brand">Instrutor Digicad</a>
  <ul class="navbar-nav">
    <li class="navbar-item">
      <a href="#" class="nav-link">Sobre</a>
    </li>
    <li class="navbar-item">
      <a href="#" class="nav-link">Portfólio</a>
    </li>
    <li class="navbar-item">
      <a href="#" class="nav-link">Contato</a>
    </li>
  </ul>
```

```
</nav>
```



Figura 306: Barra de navegação com 3 links.

Notamos que a lista ficou de forma vertical e queremos que a mesma fique assim apenas para dispositivos menores e com o menu aberto. Para transformá-la em forma horizontal, iremos colocar a nossa lista desordenada `` dentro de um divisor `<div>` com as classes **“collapse”** e **“navbar-collapse”**. Tendo o seguinte resultado:

```
<nav class="navbar navbar-dark bg-dark">
  <a href="#" class="navbar-brand">Instrutor Digicad</a>
  <div class="collapse navbar-collapse">
    <ul class="navbar-nav">
      <li class="navbar-item">
        <a href="#" class="nav-link">Sobre</a>
      </li>
      <li class="navbar-item">
        <a href="#" class="nav-link">Portfólio</a>
      </li>
      <li class="navbar-item">
        <a href="#" class="nav-link">Contato</a>
      </li>
    </ul>
  </div>
</nav>
```



Figura 307: Barra de navegação com as classes `collapse` e `navbar-collapse`.

Agora que usamos as classes **“collapse”** e **“navbar-collapse”** os *links* foram ocultos, então vamos construir um botão para ocultar/exibir os itens. Para a primeira demonstração, vamos criar um botão simples com a classe **“navbar-toggler”**, as propriedades `data-toggle=“collapse”` e `data-target=“#navbarExemplo”` (`#navbarExemplo` será o id do div contendo a nossa lista desordenada).

Além destas mudanças, iremos adicionar 2 *links* de *scripts* ao final do *HTML*, para o botão funcionar de acordo com o que queremos:

```
<script src="https://code.jquery.com/jquery-3.2.1.slim.min.js" integrity="sha384-
KJ3o2DKtIkvYIK3UENzmM7KCKRr/rE9/Qpg6aAZGJwFDMVNA/GpGFF93hXpG5KkN" cros-
sorigin="anonymous"></script>
<script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/js/bootstrap.min.js" inte-
grity="sha384-JZR6Spejh4U02d8jOt6vLEHfe/JQGiRRSQQxSfFWpi1MquV-
dAyjUar5+76PVCmYI" crossorigin="anonymous"></script>
```

Assim, temos:

```
<body>
<nav class="navbar navbar-dark bg-dark">
  <a href="#" class="navbar-brand">Instrutor Digicad</a>
  <button class="navbar-toggler" data-toggle="collapse" data-target="#navbarExemplo">Abrir</button>
  <div class="collapse navbar-collapse" id="navbarExemplo">
    <ul class="navbar-nav">
      <li class="navbar-item">
        <a href="#" class="nav-link">Sobre</a>
      </li>
      <li class="navbar-item">
        <a href="#" class="nav-link">Portfólio</a>
      </li>
      <li class="navbar-item">
        <a href="#" class="nav-link">Contato</a>
      </li>
    </ul>
  </div>
</nav>
</body>
```



Figura 308: Criação do botão abrir/fechar inicial com links exibidos.



Figura 309: botão abrir/fechar com links ocultos.

Bom, podemos notar que está evoluindo, porém há alguns detalhes finais para serem ajustados. Primeiramente, transformaremos a barra em responsiva. No nosso exemplo, os *links* serão ocultos em telas de dispositivos de até 768 pixels com a classe “**navbar-expand-md**”. Em seguida, mudaremos o nome do botão, em vez de “Abrir”, apagaremos o nome e adicionar uma *tag* `` com a classe “**navbar-toggler-icon**”. Por último, vamos adicionar a classe “**ml-auto**” (*margin left auto*) para deixar os *links* no lado direito da barra (mas isto é de acordo com a sua preferência).

Com estas mudanças temos como resultado final:

```
<nav class="navbar navbar-dark bg-dark navbar-expand-md">
  <a href="#" class="navbar-brand">Instrutor Digicad</a>
  <button class="navbar-toggler" data-toggle="collapse" data-target="#navbarExemplo">
    <span class="navbar-toggler-icon"></span>
  </button>
  <div class="collapse navbar-collapse" id="navbarExemplo">
    <ul class="navbar-nav ml-auto">
      <li class="navbar-item">
        <a href="#" class="nav-link">Sobre</a>
      </li>
      <li class="navbar-item">
        <a href="#" class="nav-link">Portfólio</a>
      </li>
      <li class="navbar-item">
        <a href="#" class="nav-link">Contato</a>
      </li>
    </ul>
  </div>
</nav>
```



Figura 310: Resultado final em telas grandes.

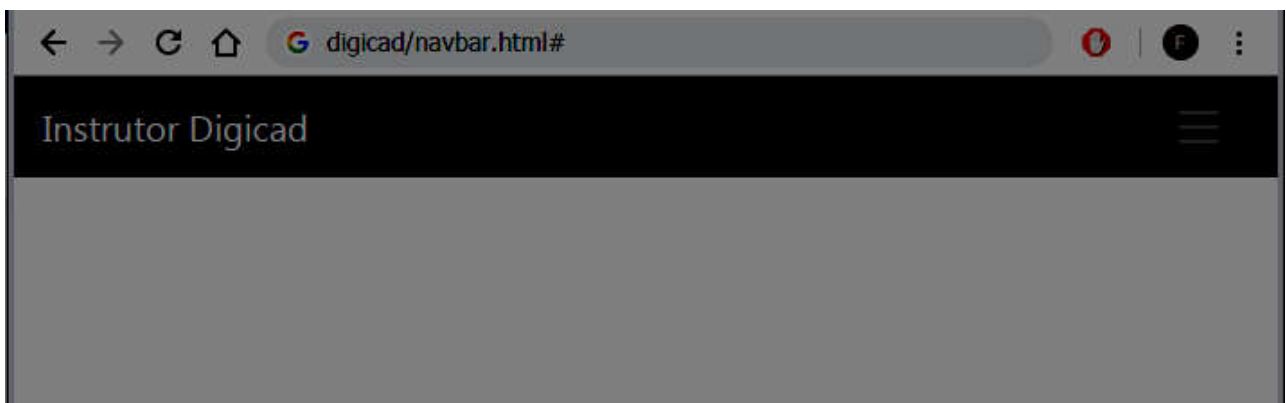


Figura 311: Resultado final em telas menores com menu oculto.

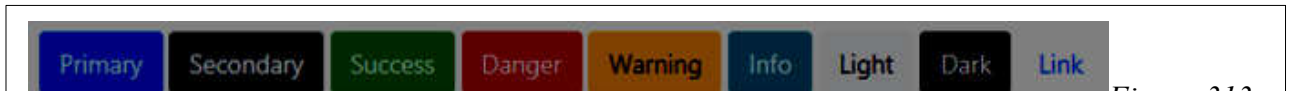


Figura 312: Resultado final em telas menores com menu expandido.

Vale reforçar que vimos um exemplo de um menu de navegação. Você pode usar estilos diferentes do Bootstrap, modificá-las ou criar novas classes. Outra dica é a opção de deixar o menu fixo (sempre visível), até mesmo quando a página for rolada para baixo, para isto basta adicionar a classe “**fixed-top**” na sua tag `<nav>`.

4.3 UTILIZAÇÃO DE BOTÕES NO LAYOUT

O Bootstrap permite incluir botões com estilos já pré-definidos, conforme adição do elemento `<button>`, podendo modificar o texto dentro do botão, bem como sua cor de fundo e borda.



```
<button type="button" class="btn btn-primary">Primary</button>
<button type="button" class="btn btn-secondary">Secondary</button>
<button type="button" class="btn btn-success">Success</button>
<button type="button" class="btn btn-danger">Danger</button>
<button type="button" class="btn btn-warning">Warning</button>
<button type="button" class="btn btn-info">Info</button>
<button type="button" class="btn btn-light">Light</button>
<button type="button" class="btn btn-dark">Dark</button>
<button type="button" class="btn btn-link">Link</button>
```

O usuário pode remover a cor de fundo do botão, basta adicionar **outline**. Por exemplo, para o botão “Dark” poderíamos escrever seu código como:

```
<button type="button" class="btn btn-outline-dark">Dark</button>
```



Além da possibilidade de modificar o tamanho dos botões, é possível também acrescentar outras propriedades e seu tamanho, como a de ocupar a largura total da tela, conforme código abaixo:

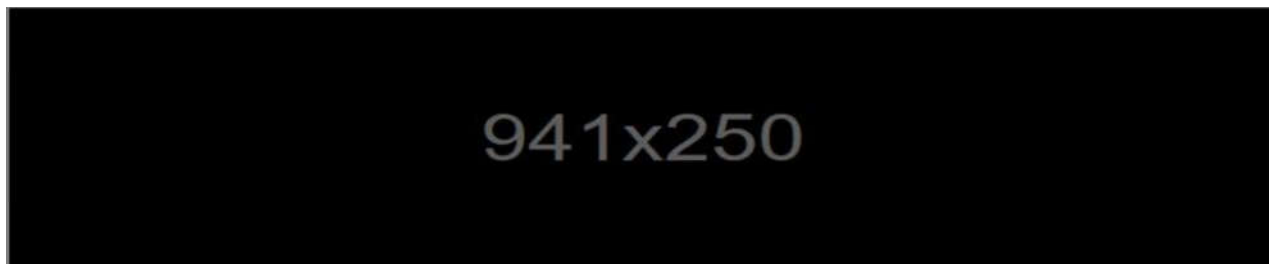
```
<button type="button" class="btn btn-primary btn-lg btn-block">Block level button</button>
```



Existem muitos outros exemplos de criação de botões de diferentes padrões e personalização no link: <https://getbootstrap.com/docs/4.0/components/>

4.4 IMAGEM RESPONSIVA NO BOOTSTRAP

O Bootstrap permite posicionar e formatar o tamanho de uma imagem de forma a manter a página com um design responsivo, assim como em botões, também podemos formatar o tamanho de uma imagem para manter o tamanho máximo da largura da página:



```

```

OBS: Lembrando que em src= “...” deve ser colocado o link da imagem.

As imagens podem ser alinhadas na borda esquerda, central ou direita da tela do *layout*.



```


```



```
<div class="text-center">
  
</div>
```

Mais detalhes podem ser vistos no link: <http://getbootstrap.com/docs/4.0/content/images/>

4.5 DESIGN RESPONSIVO USANDO @MEDIA

O recurso **@media** pode ser usado também para ajustar o tamanho da tela, assim como a resolução e a sua orientação. A **@media** também possui o recurso de criar condicionais para modificar o posicionamento, a cor, tamanho da fonte e até outras propriedades.

A modificação do tamanho da tela pode ser feita conforme código abaixo:

```
<link rel="stylesheet" media="screen and (min-width: 900px)" href="widescreen.css">
```

```
<link rel="stylesheet" media="screen and (max-width: 600px)" href="small-screen.css">
```

É possível alterar a orientação de retrato para paisagem conforme a cor de fundo do elemento:

```
@media only screen and (orientation: landscape) {  
  body {  
    background-color: lightblue;  
  }  
  .classe-exemplo {  
    width: 20em;  
    height: 15em;  
  }  
}
```

É possível também modificar as propriedades do elemento de acordo com cada situação, Por exemplo, no momento da leitura no site, a cor do texto pode ser verde, portanto quando for feito a impressão do arquivo, a cor do texto pode ser preto:

```
@media screen {  
  body {  
    color: green;  
  }  
}  
  
@media print {  
  body {  
    color: black;  
  }  
}
```

A biblioteca completa de exemplos pode ser consultada no link:

https://www.w3schools.com/cssref/css3_pr_mediaquery.asp