



Apostila de Introdução ao Desenvolvimento Web

- Disponibilização e Formatação de Conteúdos na Web com Introdução às Linguagens HTML5, CSS3, JavaScript, PHP e o uso do BD MySQL através do pacote WampServer -

**Disciplina: Autoria e Design na Internet I
Profa. Flávia Pereira de Carvalho**

Fevereiro/2018

Sumário

	<i>Página</i>
1 INTRODUÇÃO AO DESENVOLVIMENTO DE PÁGINAS WEB	5
<u>1.1 Front-end (client-side)</u>	5
<u>1.2 Back-end (server-side)</u>	6
<u>1.3 Model-View-Controller</u>	6
2 INTRODUÇÃO À LINGUAGEM HTML5	7
3 EDIÇÃO DE DOCUMENTOS HTML.....	9
4 PUBLICAÇÃO DE DOCUMENTOS NA INTERNET (SITES).....	10
5 DOCUMENTO HTML BÁSICO E SEUS COMPONENTES.....	12
<u>5.1 <!DOCTYPE html></u>	12
<u>5.2 <html lang="pt-br"></u>	12
<u>5.3 <head> e </head></u>	13
<u>5.4 <meta charset="utf-8"></u>	13
<u>5.5 <title>Título da Página</title></u>	13
<u>5.6 <body> e </body></u>	14
6 CABEÇALHOS	15
7 SEPARADORES.....	16
<u>7.1 Quebra de linha</u>	16
<u>7.2 Parágrafos</u>	16
<u>7.3 Linha Horizontal</u>	17
8 COMENTÁRIOS EM HTML	17
9 LISTAS EM HTML.....	18
<u>9.1 Listas Não Ordenadas</u>	18
<u>9.2 Listas Ordenadas</u>	19
10 LIGAÇÕES (USO DE LINKS).....	20
<u>10.1 Atributos</u>	21
<u>10.2 Caminhos</u>	21
10.2.1 Caminho Relativo.....	21
10.2.2 Caminho Absoluto	22
<u>10.3 Indicadores</u>	22
11 INSERÇÃO DE IMAGENS.....	23
<u>11.1 Atributos Básicos de Imagem</u>	24
12 NOVOS ELEMENTOS DA HTML5	25
<u>12.1 Elementos de Estrutura do Layout</u>	25
12.1.1 Tag header.....	26
12.1.2 Tag footer.....	27
12.1.3 Tag main.....	28
12.1.4 Tag nav.....	28
12.1.5 Tag section.....	29
12.1.6 Tag article.....	30
12.1.7 Tag aside.....	31
13 SEMÂNTICA	32
14 OUTRAS TAGS IMPORTANTES	34
<u>14.1 div</u>	34
<u>14.2 span</u>	35
<u>14.3 b e strong</u>	36
<u>14.4 i e em</u>	36
<u>14.5 small</u>	36
<u>14.6 code</u>	37

14.7 <i>audio</i>	38
14.8 <i>video</i>	38
14.9 <i>canvas</i>	39
15 ATRIBUTOS.....	40
15.1 <i>id</i>	40
15.2 <i>class</i>	40
15.3 <i>style</i>	40
16 TABELAS.....	41
16.1 <i>Elementos básicos de tabelas</i>	41
16.2 <i>Títulos, linhas e elementos</i>	41
16.3 <i>Exemplo de uma tabela simples</i>	42
16.4 <i>Títulos compreendendo mais de uma coluna ou linha</i>	42
16.5 <i>Tabelas sem borda</i>	43
17 FORMULÁRIOS HTML.....	44
17.1 <i>Criando um form</i>	44
17.2 <i>Parâmetros do form</i>	44
17.2.1 <i>get</i>	45
17.2.2 <i>post</i>	45
17.3 <i>Elementos do form</i>	46
17.3.1 <i>Campo Texto</i>	46
17.3.2 <i>Campo Senha</i>	47
17.3.3 <i>Botão Rádio (radio button)</i>	47
17.3.4 <i>Botão de Checagem (check box)</i>	48
17.3.5 <i>Botão Submeter (submit)</i>	49
17.3.6 <i>Botão Reset</i>	49
17.3.7 <i>Select</i>	50
17.3.8 <i>Área de Texto</i>	51
17.4 <i>Trabalhando com os dados enviados pelo Form</i>	52
17.5 <i>Colocando Formulários em Prática</i>	53
18 PACOTE WAMPSEVER.....	56
18.1 <i>O que é WampServer</i>	56
18.2 <i>Instalando WampServer</i>	56
18.3 <i>Usando WampServer</i>	57
19 INTRODUÇÃO ÀS REGRAS CSS.....	59
19.1 <i>Algumas considerações sobre CSS</i>	60
19.2 <i>Alguns benefícios, que justificam o uso de CSS</i>	60
19.3 <i>Sintaxe do CSS</i>	61
19.3.1 <i>Agrupar Seletores</i>	61
19.4 <i>Site que altera seu design com CSS</i>	62
19.5 <i>Formas de usar CSS</i>	63
19.5.1 <i>Inline</i>	63
19.5.2 <i>Interna</i>	63
19.5.3 <i>Externa</i>	64
19.6 <i>Exemplos Práticos</i>	65
19.6.1 <i>Exemplo1 usando CSS Inline</i>	65
19.6.2 <i>Exemplo2 usando CSS Incorporado</i>	67
19.6.3 <i>Exemplo3 usando CSS Externo</i>	69
19.7 <i>Efeito Cascata</i>	71
19.8 <i>Comentários em CSS</i>	72
19.9 <i>Seletores Classe e Id</i>	72
19.9.1 <i>Quando e Como usar Id</i>	73
19.9.2 <i>Exemplo usando Class</i>	74
19.10 <i>Box Model</i>	76
19.10.1 <i>Propriedades CSS para o Box Model</i>	83
20 INTRODUÇÃO À LINGUAGEM JAVASCRIPT.....	85
20.1 <i>Qual a diferença entre JavaScript e Java?</i>	85
20.2 <i>Programando com JavaScript</i>	85

<u>20.3 Entrada e Saída de Dados: Caixas de Diálogo</u>	86
20.3.1 alert	86
20.3.2 prompt	87
20.3.3 confirm	87
<u>20.4 Entrada e Saída de Dados: Formulários</u>	88
20.4.1 Caixa de Texto (textField).....	88
20.4.2 Caixa de Seleção (select).....	89
<u>20.5 Funções</u>	90
20.5.1 Parâmetros	90
20.5.2 Variáveis Globais e Locais.....	91
20.5.3 Retorno de Valores.....	92
<u>20.6 Validação de Dados em Formulários</u>	92
20.6.1 Validação de Dados em Entrada em Formulários	92
20.6.2 Função para Apresentar Dados do Formulário.....	95
20.6.3 Forma mais Elegante de Apresentar Dados do Formulário	97
21 REFERÊNCIAS	99

1 Introdução ao Desenvolvimento de Páginas Web

Existem basicamente duas camadas quando se trata de desenvolvimento web: **Front-end** e **Back-end**. A camada de front-end é onde fica a estrutura HTML, as regras CSS e os comandos JavaScript, ou seja, tudo que é apresentado para os usuários. Essa camada também é chamada de *client-side*. Na camada de back-end pode ter várias tecnologias diferentes, como: PHP, C# (CSharp), Java, Ruby, Python, entre outras, dependendo dos objetivos do site e “a própria opção do programador” (também chamada de *server-side*).

Por exemplo: ao criar um *script* em linguagem **back-end** que apenas calcula a soma de 2+2, será o **servidor** (*back, server*) que calculará esse resultado. Se o mesmo cálculo for feito em alguma linguagem **front-end**, como JavaScript, por exemplo, quem calculará será o **navegador** (*front, client, browser*) do usuário. Por isso o termo *client* ou *server*.

Assim, os profissionais que trabalham na interface do usuário, são chamados de Desenvolvedores Front-end, e aqueles que trabalham no *core* da aplicação, fazendo uma programação que somente o servidor irá entender, são chamados de Desenvolvedores Back-end.

1.1 Front-end (client-side)

As linguagens *client-side* são linguagens onde apenas o **navegador** vai entender. Quem vai processar essa linguagem não é o servidor, mas sim o *browser* (navegador web).

Qual a função de cada uma dessas linguagens que podem compor o **front-end** de uma página web?

- **HTML** cria a estrutura, o layout da página: determina em quantos “blocos” a página será dividida, se terá cabeçalho, rodapé, menus.
- **CSS** cria a apresentação da página: toda formatação é feita através de regras CSS. Através de CSS se posiciona todos os elementos na página, se colore, se formata.
- **JavaScript** é responsável pelas ações das páginas: faz a interatividade das páginas com os usuários. Não confunda JS com a Linguagem de Programação Java, pois são bem diferentes!

O objetivo do uso dessas três linguagens para desenvolver aplicações web, é separar a informação (conteúdo), formatação e comportamento, das páginas.

1.2 Back-end (server-side)

As linguagens *server-side* são linguagens que o **servidor** entende, ou seja, é o código que o servidor vai processar e depois vai mandar para o navegador alguma resposta, algum retorno.

Qual a função de cada uma dessas linguagens que podem compor o *back-end* de uma página web? As linguagens usadas no *server-side*, como por exemplo: PHP, C# (CSharp), Java, Ruby, Python, podem ser usadas para sites que precisam, por exemplo, acessar um Banco de Dados (BD), sites que precisam ter algum tipo de serviço (*web service*) etc.

1.3 Model-View-Controller

A seguir, é apresentada uma imagem para explicar um dos mais importantes conceitos em desenvolvimento de aplicações web, que é o padrão de projeto **MVC** (*Model-View-Controller*, em português Modelo-Visão-Controlador).

Essa é uma abordagem utilizada na arquitetura de aplicações que, em sua maior parte, determina o design de uma aplicação baseada em Banco de Dados (BD) e tornou-se um padrão de mercado para a criação de aplicações web. O padrão é tão amplamente aceito como a melhor maneira de criar aplicações web que passou a ser incluído nos *frameworks* mais populares para esse tipo de aplicação (PUREWAL, 2014).

A seguir é apresentada uma imagem explicativa do funcionamento do MVC.

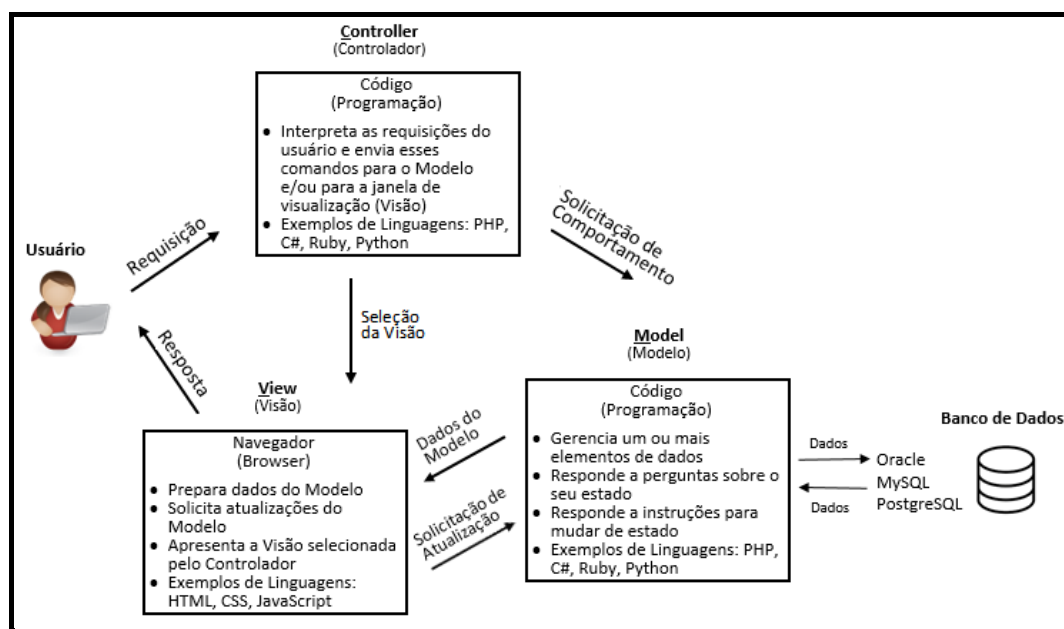


Figura 1: Model-View-Controller

Fonte: Adaptado de [Pres2016]

Nesta apostila, serão estudadas basicamente as linguagens que compõem o **Front-end** das páginas web, ou seja, é apresentada uma introdução sobre HTML, CSS e JavaScript, com exemplos teóricos e práticos.

2 Introdução à Linguagem HTML5

HTML é uma sigla, *HyperText Markup Language* que, em Português, significa Linguagem de Marcação de Hipertexto. Hipertextos são os textos da web, ou seja, todo conteúdo textual das páginas de um site é chamado de hipertexto. Já as imagens, vídeos, gráficos, sons e conteúdos não textuais, em geral são chamados de hiperlinks.

Uma gama de tecnologias (como CSS, JavaScript, AJAX, JSON) pode ser usada para definir os elementos de uma página web. Contudo, na base da estrutura de uma página web está a HTML. Sem HTML não haveria página web alguma. HTML é o que o navegador "lê" (renderiza) para apresentar a página ao usuário em frente ao computador. As especificações e padrões da HTML são geridas pelo *World Wide Web Consortium - W3C* (www.w3.org). A versão mais atual do padrão HTML é a HTML5.

Todo documento HTML apresenta elementos entre parênteses angulares (< e >). Esses elementos são as **etiquetas (em inglês, tags)** de HTML, que são os comandos de formatação da linguagem (ou, para não chamar de comando, já que não se trata de uma linguagem de programação, são os **elementos de marcação**). A maioria das etiquetas tem sua correspondente de fechamento, representada com uma "barra" (/):

```
<etiqueta>conteúdo a ser apresentado na página web</etiqueta>
```

O fechamento das *tags* HTML é necessário porque elas servem para definir a formatação de uma porção de texto (uma parte da página), e assim marca-se onde começa e onde termina o conteúdo especificado por ela. **As tags HTML servem para estruturar, hierarquizar e demarcar o conteúdo de uma página web.**

Alguns elementos são chamados "vazios", pois não marcam uma região da página, apenas inserem alguma coisa no documento, não havendo a necessidade do fechamento:

```
<etiqueta>
```

Muitos elementos podem ter atributos, que são como propriedades do elemento, escritos com um nome e valor:

```
<etiqueta nomeatributo1=valor1 nomeatributo2=valor2>...</etiqueta>
```

Alguns poucos atributos podem ter apenas um valor. Eles são atributos *booleanos* e podem ser abreviados especificando-se somente o seu nome ou deixando o seu valor em branco. Assim, os três exemplos a seguir têm o mesmo significado:

```
<input required="required">  
<input required="">  
<input required>
```

HTML é um recurso muito simples e acessível para a produção de documentos. Nesta apostila, será possível aprender uma introdução aos seus elementos. Nesta versão da HTML5 há grandes novidades como web **semântica e acessibilidade**, recursos esses que eram possíveis somente através de tecnologias alternativas.

A HTML tem um propósito específico: **estruturar e demarcar o conteúdo de uma página web**. As recomendações dos Padrões Web (*Web Standards*) foram importantes na reeducação dos desenvolvedores de sites e de navegadores, pois antes da versão 5, a HTML era usada também para definir aspectos visuais e de interação de um site.

Atenção: usa-se HTML apenas para fins estruturais. Tudo o que é referente a apresentação visual ou a interatividade (salvo *links*) é domínio das linguagens CSS e JavaScript, respectivamente.

3 Edição de Documentos HTML

É possível criar arquivos HTML usando qualquer Editor de Textos, contudo é mais conveniente e apropriado usar editores próprios para escrever marcação HTML, ou seja, existem **Editores HTML**. Existem vários Editores HTML, como por exemplo **Notepad++** (notepad-plus-plus.org), **Sublime Text** (www.sublimetext.com), **Brackets** (brackets.io).

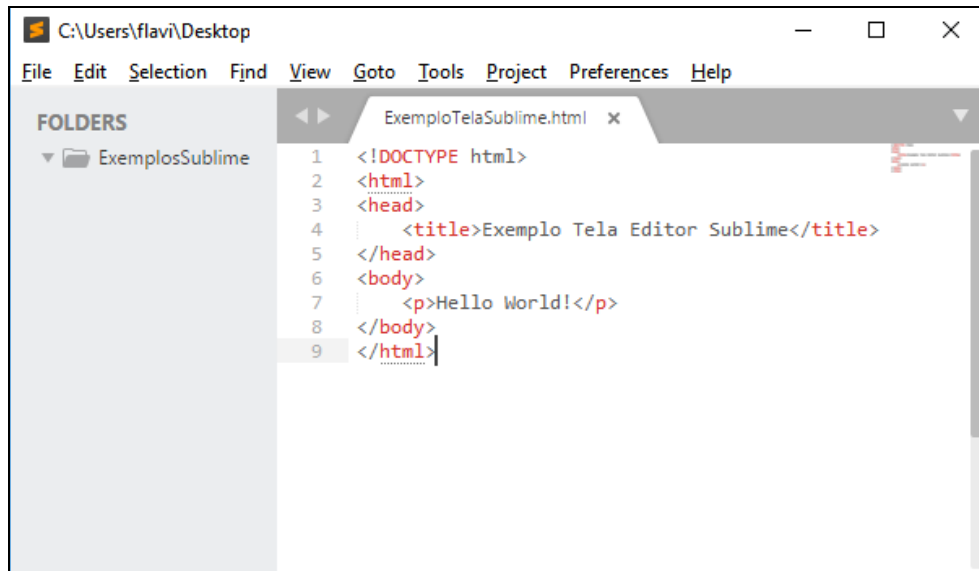


Figura 2: Tela do Editor SublimeText
www.sublimetext.com

Além dos editores específicos para HTML, Editores de Textos bastante utilizados, como o Word, Bloco de Notas, entre outros, permitem a exportação de seus documentos próprios para o formato HTML (menu Arquivo, Salvar como, Tipo). Um documento HTML, normalmente terá extensão **.html** (ou **.htm**).

Também existem vários **Editores HTML Online**, como por exemplo: Online HTML Editor (www.onlinehtmleditor.net), Mozilla Thimble (thimble.webmaker.org/pt-BR), entre outros.

Existem também alguns editores chamados **WYSIWYG**: que significa “*What you see is What you get*”, ou seja, “O que você vê é o que você obtém”. O termo é usado para classificar ferramentas de edição e desenvolvimento que permitem visualizar, em tempo real, exatamente aquilo que será publicado ou impresso, é um conceito, um modelo de software.

O WYSIWYG Web Builder implementa muito bem isso, nele é possível construir uma página web montando os elementos que deseja (texto, imagens, vídeos etc.) e o software desenvolve o código HTML. É como o modo Design do Dreamweaver, ou como se faz páginas no **WordPress**, por exemplo. No entanto, não é uma boa opção para quem deseja **aprender** HTML e CSS! 😊

4 Publicação de Documentos na Internet (Sites)

Para que uma página esteja permanentemente disponível na web, ela precisa ter um endereço fixo, alojada em um servidor.

Existem vários provedores de espaço (*hosting*) gratuitos e também os provedores de acesso geralmente oferecem espaço para os sites de seus assinantes. Sites com fins lucrativos geralmente são hospedados em provedores de espaço pagos.

Definida a hospedagem, basta enviar para o provedor os arquivos do site que deseja publicar (via FTP¹ ou por uma página de envio no próprio provedor de espaço) e o site já estará disponível para visitas no mundo todo. Um programa bastante usado para envio de arquivos de um computador local para um servidor e vice-versa, através do Windows, é o **WinSCP** (winscp.net/eng/index.php).

Para publicar seu site no servidor FIT, através do WinSCP, deve preencher os dados do login com o *Host name* (nome do servidor) que é o endereço do servidor ao qual deseja se conectar, no caso, será **fit.faccat.br**. Também deve preencher o *User name* (nome do usuário) e *Password* (senha), e clicar no botão Login, conforme a Figura 3, abaixo:

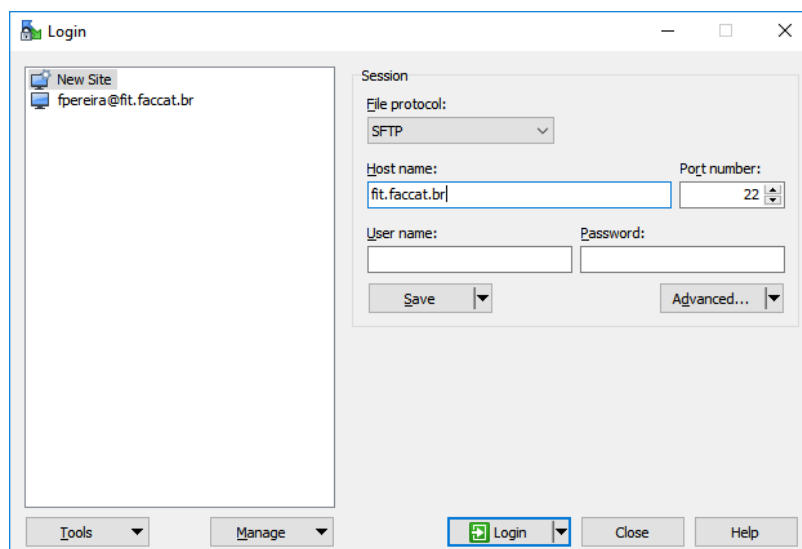


Figura 3: Tela de Login do WinSCP

Para **solicitar a criação da sua conta shell no servidor FIT** (usuário e senha), você deve preencher o formulário disponível em fit.faccat.br/~rh/formulario/ e, em alguns dias, **receberá uma confirmação por e-mail com os dados para acessar o servidor** (lembrando que a senha deve ter ao menos 8 dígitos, sendo ao menos 1 número, 1 letra maiúscula e 1 letra minúscula). Qualquer dúvida ou problema para acessar sua conta shell no servidor FIT, enviar e-mail para o laboratorista Rodrigo Henrich (rodrigohenrich@faccat.br).

¹ FTP - *File Transfer Protocol*: Protocolo de Transferência de Arquivos.

Na Figura 4, a seguir, é apresentada a tela inicial do WinSCP ao se logar no servidor FIT da FACCAT. Na parte da esquerda, aparecem os arquivos e pastas locais, ou seja, os que estão no computador que você está utilizando no momento e, na parte direita da tela, são os arquivos e pastas da sua conta shell no servidor FIT (só você tem acesso e pode acessar de qualquer computador com internet).

Para publicar seu site, deve enviar os arquivos que compõem o site para **dentro da pasta www** do servidor.

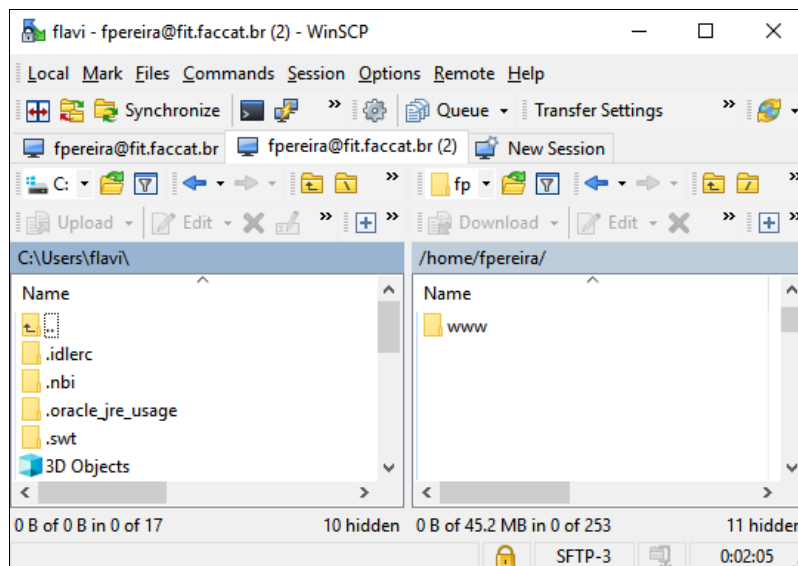


Figura 4: Acessando o Servidor FIT através do WinSCP

Em geral, tem-se como arquivo da página inicial de um site, o arquivo chamado **index.html**. Esse é o arquivo *default* (padrão) em um diretório. Ao entrar em uma URL que não especifica um arquivo em um diretório, o *browser* solicita ao servidor o arquivo **index.html**. É por isso que é possível escrever tanto www.exemplo.com.br, quanto www.exemplo.com.br/index.html. Se não existir um arquivo chamado **index.html**, o servidor poderá retornar uma mensagem de "acesso proibido à listagem do diretório" ou então retornar a listagem dos arquivos daquele diretório, o que pode não ser o que se deseja. Esse arquivo *default* é definido na configuração do servidor WWW.

Se o seu site tiver como página inicial um arquivo chamado **index.html** (que é o padrão), o endereço web da sua página será: **fit.faccat.br/~usuário**, sendo que o seu usuário deve ser aNúmeroDeMatrícula (letra "a" minúscula seguido do seu código de matrícula).

Observação: Tanto o arquivo **index.html** quanto a pasta **www**, não precisam ser citados ao acessar sua página web, pois são informações padrão!

5 Documento HTML Básico e seus Componentes

A estrutura de um documento HTML apresenta os seguintes componentes básicos:

```

1  <!DOCTYPE html>
2  <html lang="pt-br">
3  <head>
4      <meta charset="utf-8">
5      <title> Título da Página </title>
6  </head>
7  <body>
8      textos,
9      imagens,
10     links etc...
11 </body>
12 </html>
    
```

Figura 5: Componentes Básicos de um Documento HTML

A seguir são apresentadas as descrições de cada linha do código-fonte acima (da Figura 5).

5.1 <!DOCTYPE html>

Linha1: <!DOCTYPE html>

DOCTYPE é um acrônimo de DOCUMENT TYPE, ou seja, Tipo de Documento, em português. Deve ser o primeiro elemento de um documento HTML, sendo declarado antes da tag <html>. É responsável por informar ao navegador qual a versão do HTML que está sendo usada, no caso, **HTML5**, baseado nos padrões do W3C. Não é uma *tag*, é uma declaração!

5.2 <html lang="pt-br">

Linha2: <html lang="pt-br">

A *tag* <html> define o início do documento, informando ao navegador que a partir daquele ponto ele deve interpretar como um documento HTML. A identificação da língua auxilia os tradutores automáticos (softwares ou máquinas capazes de ler o conteúdo de uma página e traduzi-lo para outro idioma) e também qualquer outro software que venha a ler sua página, por exemplo, buscadores como o Google. Neste caso, está sendo informado ao navegador que o documento HTML está sendo escrito na Língua Portuguesa do Brasil (**lang** é um atributo e **pt-br** o valor da *tag* <html>).

5.3 <head> e </head>

Linhas 3 e 6: <head> e </head>

São as *tags* de abertura e fechamento do elemento **<head>**. Essa seção é o **cabeçalho** da página, onde é possível ter vários outros elementos dentro contendo informações sobre o documento.

Costuma-se colocar dentro da *tag head* definições que devem ser obedecidas pelo navegador antes que ele comece a renderizar a página web. Assim, o título da janela, meta informações, estilos e *scripts* são preferencialmente colocados nesta seção.

Ainda no cabeçalho é possível fazer referência a algum arquivo externo que seja necessário utilizar durante a execução do documento. Mais detalhes, serão estudados nos capítulos sobre CSS e JavaScript.

5.4 <meta charset="utf-8">

Linha 4: <meta charset="utf-8">

O que significa? **meta** é usado para declarar metadados (informações sobre a própria página, como resumo do conteúdo, palavras-chaves, indicações a robôs de busca, codificação de caracteres, nome do autor entre outras), **charset** é usado para indicar o formato de codificação de caracteres usados no documento e **utf-8** é um formato de codificação, ou seja, é usado para mostrar ao navegador o tipo de codificação que será utilizado no site e, assim, funcionar corretamente todas acentuações, por exemplo. A codificação UTF-8 é uma das mais abrangentes, dessa forma, evita-se problemas com caracteres especiais (como acentos e outros).

5.5 <title>Título da Página</title>

Linha 5: <title>Título da Página</title>

O elemento **<title>** define um título que irá aparecer no título da janela ou da aba do navegador. Ele é importante para buscadores como o Google. Portanto, costuma-se escrever um título de página que seja conciso e que descreva bem o conteúdo da página. É importante que contenha as palavras-chave desejadas para indexação.

```
<head>
  <title>Página do Fulano de Tal</title>
</head>
```

Todo documento web deve ter um título! Esse título é referenciado em buscas pela rede, dando uma identidade ao documento. Ao adicionar uma página aos Favoritos (*Bookmarks*) do navegador, o título da página se torna a âncora de atalho para ela. Por isso é sugerido que os títulos dos documentos sejam sugestivos, evitando-se títulos genéricos como "Introdução", por exemplo. O título também é bastante significativo para a listagem de uma página nos resultados de pesquisas nos *sites* de busca da Internet.

5.6 <body> e </body>

Linhas 7 e 11: <body> e </body>

Tudo que estiver contido entre as *tags* **<body>** será mostrado na janela principal do *browser*, sendo apresentado ao leitor. **<body>** é o “corpo” da página e pode conter cabeçalhos, parágrafos, listas, tabelas, *links* para outros documentos, imagens, formulários, animações, vídeos, sons e *scripts* embutidos. Cada documento HTML deve conter apenas uma seção <body>.

DICA: As etiquetas (*tags*) HTML não são sensíveis à caixa (**a linguagem HTML não é Case Sensitive**), ou seja, tanto faz escrever <HTML>, <Html>, <html>, <HtMl> etc. Não tem diferença entre maiúsculas e minúsculas. Mas, a prática da “boa programação” diz que **deve-se usar letras minúsculas** no código fonte de páginas web (e também de programas em geral), exceto na declaração DOCTYPE 😊

6 Cabeçalhos

Há seis níveis de cabeçalhos em HTML, de **<h1>** a **<h6>**, que servem para colocar **títulos e subtítulos** nas páginas web, conforme apresentado nos exemplos abaixo (**h** de *head* = cabeçalho):

```

<h1>Este é um cabeçalho de nível 1</h1>
<h2>Este é um cabeçalho de nível 2</h2>
<h3>Este é um cabeçalho de nível 3</h3>
<h4>Este é um cabeçalho de nível 4</h4>
<h5>Este é um cabeçalho de nível 5</h5>
<h6>Este é um cabeçalho de nível 6</h6>
    
```

Esses cabeçalhos são mostrados no *browser* da seguinte forma:

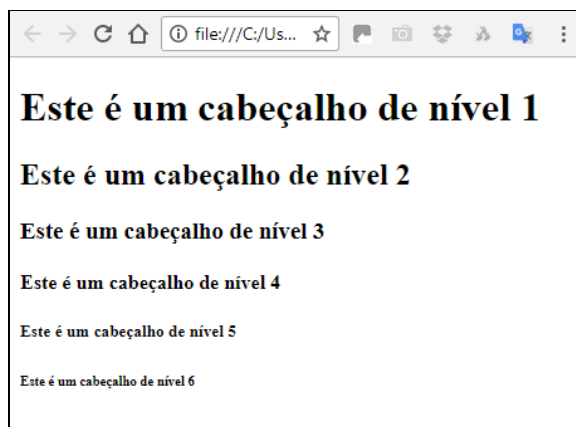


Figura 6: Visualização dos Cabeçalhos <h1> ao <h6>

7 Separadores

As quebras de linha feitas nos códigos-fonte das páginas web, não são significativas na apresentação de documentos em HTML, ou seja, ao pressionar a tecla Enter no código-fonte da página web nos comandos HTML, o resultado da página no navegador não é alterado, **os navegadores não reconhecem o Enter como quebra de linha**. Para organizar os textos, são necessários separadores, apresentados a seguir.

7.1 Quebra de linha

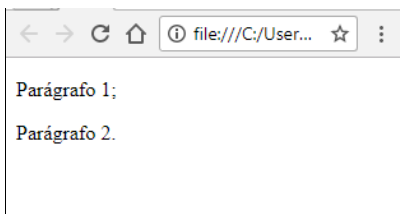
Quando deseja-se mudar de linha, deve-se usar o elemento `
`. Isso só é necessário quando precisa-se de uma quebra de linha em determinado ponto, pois os *browsers* já quebram as linhas automaticamente para apresentar os textos. O nome da *tag* vem das palavras ***break row*** (quebra de linha). Com sucessivas *tags* `
`, pode-se inserir diversas linhas em branco nos documentos HTML. Esse é um elemento vazio, pois só existe a *tag* de abertura (não precisa fechar a *tag* usando `/`).

7.2 Parágrafos

Para separar blocos de texto, usa-se o elemento `<p>` de parágrafo, por exemplo:

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

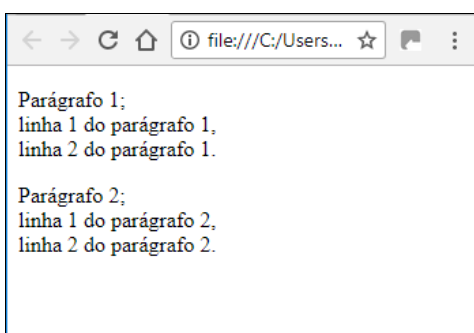
Que produz o seguinte resultado na página web:



Combinando parágrafos e quebras de linha, pode-se ter, por exemplo:

```
<p>Parágrafo 1;<br>linha 1 do parágrafo 1,<br>linha 2 do parágrafo 1.</p><p>Parágrafo 2;<br>linha 1 do parágrafo 2,<br>linha 2 do parágrafo 2.</p>
```

O resultado da marcação acima é:



7.3 Linha Horizontal

A tag `<hr>` insere uma linha horizontal de 1px (1 pixel) na página, como a apresentada abaixo:

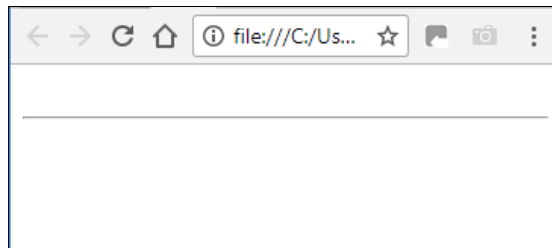


Figura 7: Visualização da Linha Horizontal da Tag `<hr>`

Esta tag, também é um elemento vazio, ou seja, não é necessário abrir e fechar a tag `<hr>`.

8 Comentários em HTML

Para fazer algum comentário no código-fonte HTML usa-se a notação: `<!-- comentários -->`. Conforme apresentado na imagem abaixo, os comentários aparecerão em cinza no código-fonte das páginas HTML, se estiver usando o Editor Sublime. Os comentários não causam efeito algum na apresentação das páginas nos navegadores, ou seja, só são vistos ao acessar o código-fonte dos arquivos.

```

1  <!DOCTYPE html>
2  <html>
3  <head>
4  |   <title></title>
5  </head>
6  <body>
7
8  <!-- blá blá blá -->
9
10 </body>
11 </html>
    
```

Figura 8: Comentários no Código-Fonte HTML

9 Listas em HTML

Listas são bastante usadas em páginas HTML, principalmente para fazer Menus de opções. Existem dois tipos de listas em HTML, apresentadas a nas seções a seguir.

9.1 Listas Não Ordenadas

As listas não ordenadas, são listas com marcadores. Para iniciar uma lista **não** ordenada usa-se a tag `` que vem das palavras em inglês *Unordered List*, que significa Lista Não Ordenada. Deve-se abrir e fechar a lista: `` e ``. E, para cada item de uma lista, usa-se a tag `` que também vem do inglês, *List Item*, ou seja, Item da Lista. Veja o exemplo a seguir:

```
<ul>
  <li>item de uma lista</li>
  <li>item de uma lista, que pode ser tão grande quanto se queira, sem que seja necessário se preocupar com a formatação das margens de texto</li>
  <li>item da lista (list item)</li>
</ul>
```

Resultado na página:

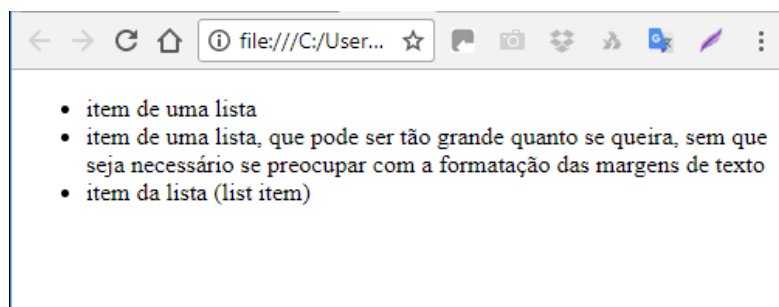


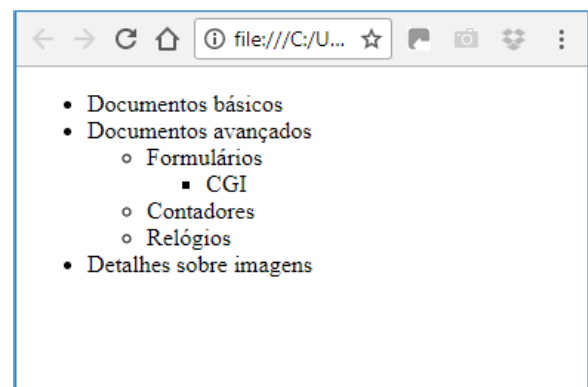
Figura 9: Lista Não-Ordenada ``

A quebra de linha de uma frase grande, se dará de acordo com o tamanho da janela do navegador. A seguir, é apresentado mais um exemplo de lista não ordenada, mas com vários níveis de informações.

Código-fonte

```
<ul>
  <li>Documentos básicos</li>
  <li>Documentos avançados</li>
  <ul>
    <li>Formulários</li>
    <ul>
      <li>CGI</li>
    </ul>
    <li>Contadores</li>
    <li>Relógios</li>
  </ul>
  <li>Detalhes sobre imagens</li>
</ul>
```

Resultado na página



9.2 Listas Ordenadas

As Listas Ordenadas, são listas numeradas (com números ou com letras). Para iniciar uma lista ordenada usa-se a tag `` que vem das palavras em inglês *Ordered List*, que significa Lista Ordenada. Deve-se abrir e fechar a lista: `` e ``. E, para cada item da lista, usa-se a tag ``, da mesma forma que nas listas não ordenadas. Veja o exemplo a seguir:

```
<ol>
  <li>item de uma lista ordenada</li>
  <li>item de uma lista numerada, que pode ser tão grande quanto se queira, sem que seja
  necessário se preocupar com a formatação das margens de texto</li>
  <li>item de lista numerada (list item)</li>
</ol>
```

Resultado na página:

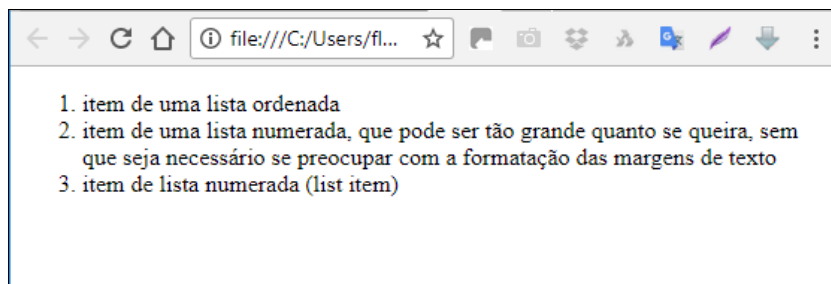


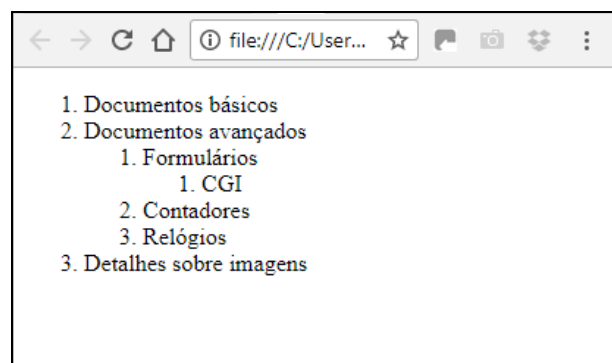
Figura 10: Lista Ordenada ``

Estas listas não apresentam numeração em formato 1.1, 1.2 etc., quando compostas, apresentam-se da seguinte forma (mesmo exemplo anterior do capítulo 9.1, mas trocando todas as tags `ul` por `ol`):

Código-fonte

```
<ol>
  <li>Documentos básicos</li>
  <li>Documentos avançados</li>
  <ol>
    <li>Formulários</li>
    <ol>
      <li>CGI</li>
    </ol>
    <li>Contadores</li>
    <li>Relógios</li>
  </ol>
  <li>Detalhes sobre imagens</li>
</ol>
```

Resultado na página



10 Ligações (uso de Links)

Com HTML é possível fazer ligações de uma região de texto (ou imagem) a um outro documento (ou a outra parte do próprio documento). Provavelmente você já deve ter visto em alguma página web exemplos dessas ligações: o *browser* destaca essas regiões e imagens do texto, indicando que são ligações de hipertexto - também chamadas *hypertext links* ou *hyperlinks* ou simplesmente *links*, onde normalmente, o mouse vira uma “mãozinha” e, ao clicar, “chama-se” (abre-se) um outro documento, uma outra página web ou ainda uma figura, por exemplo.

Para inserir um *link* em um documento, utiliza-se a tag `<a>` (de *anchor* = âncora), seguida do atributo `href` (indica o endereço que será acionado quando o *link* for clicado), da seguinte forma:

```
<a href="arquivo-destino">texto-do-link-que-aparece-na-página</a>
```

Onde:

arquivo-destino: é o endereço do documento de destino, da página ou da imagem a qual deseja-se abrir no momento em que o *link* for clicado.

texto-do-link-que-aparece-na-página: é o texto ou imagem que servirá de ligação hipertexto do documento atual para o documento que será “chamado”, ou seja, é o texto que aparece na página como um *link*.

Exemplo:

```
<a href="http://www.google.com">Clique Aqui</a>
```

Resultado na página: os links, por padrão, aparecem em azul e sublinhados, nos navegadores. Ao apontar o mouse para o texto “Clique Aqui”, como ele é um *link*, aparece na barra de status o endereço para o qual ele está apontando, no caso, “www.google.com”.

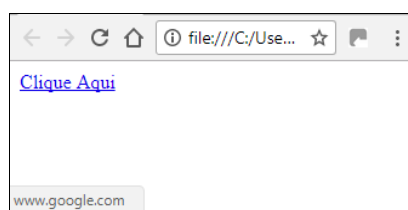


Figura 11: Exemplo de Link

10.1 Atributos

A tag `<a>` tem vários atributos que são utilizados de acordo com a ação associada ao link. Os mais comumente utilizados são apresentados a seguir.

href: indica o arquivo ou o endereço de destino da ligação de hipertexto.

target: indica o quadro (*frame*) em que será carregado o arquivo-destino. Por exemplo, **target="_blank"**, indica que a nova página (documento ou imagem) deve ser aberta em uma nova aba do navegador.

name: marca um **indicador**, isto é, uma região de um documento como destino de uma ligação. Maiores detalhes serão abordados na seção sobre indicadores (Capítulo 10.3).

10.2 Caminhos

Os *links* podem estar indicados como caminhos relativos ou absolutos, detalhados nos capítulos a seguir. Para explicar como funcionam os caminhos relativos e absolutos, os exemplos serão baseados na estrutura de pastas e arquivos da imagem apresentada na Figura 13, abaixo:

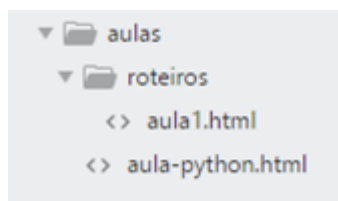


Figura 12: Exemplo de Estrutura de Pastas e Arquivos

10.2.1 Caminho Relativo

O caminho relativo pode ser usado sempre que for necessário **fazer referência a um documento armazenado no mesmo servidor do documento atual**.

Através do campo de endereços do *browser*, é possível identificar se um documento (página) que está sendo visualizado está dentro de algum diretório (pasta) ou não. Por exemplo, ao acessar o site da aula de Python hospedado no servidor da FACCAT, tem-se o seguinte endereço: `http://www.faccat.br/aulas/aula-python.html`. Conforme imagem acima (Figura 13), o que pode-se concluir é que o documento que está sendo visualizado no momento, chamado **aula-python.html**, está armazenado **dentro** de um diretório (pasta) chamado **aulas** localizado no servidor **www.faccat.br**.

Então, para escrever, por exemplo um *link* deste documento (*aula-python.html*) para um outro documento chamado **aula1.html** que está localizado no diretório **/aulas/roteiros/** do mesmo servidor *www.faccat.br*, ou seja, está dentro da pasta roteiros que, por sua vez, está dentro da pasta aulas (conforme Figura 13), é necessário apenas escrever o caminho a partir de onde se está naquele momento, da seguinte maneira:

```
<a href="roteiros/aula1.html">Exemplo de Caminho Relativo</a>
```

Para usar *links* com caminhos relativos é preciso, portanto, conhecer muito bem a estrutura do diretório do servidor no qual se está trabalhando, pois deve-se indicar todo o caminho onde está o documento no qual está se referindo no *link*. Quando há alguma dúvida, o melhor é usar o caminho absoluto, ou seja, indicar o caminho completo, para não correr o risco de errar, conforme explicado na seção a seguir.

10.2.2 Caminho Absoluto

Utiliza-se o caminho absoluto quando se deseja referenciar um documento que esteja em outro servidor (ou no mesmo, mas não se tem certeza do caminho), por exemplo:

```
<a href="http://www.google.com.br">Página do Google</a>
```

Oferece um *link* **Página do Google** que, ao ser clicado com o mouse, abrirá a página cujo endereço é *http://www.google.com.br*. Com a mesma sintaxe, é possível escrever *links* para qualquer servidor de informações da Internet.

Conforme o exemplo da seção anterior (10.2.1), caso não se conheça a estrutura de pastas ou não se queira arriscar, podendo ocorrer algum erro, é possível usar sempre o caminho absoluto, ou seja, o caminho completo. Então o exemplo ficaria da seguinte forma:

```
<a href="http://www.faccat.br/aulas/roteiros/aula1.html">Exemplo de Caminho Absoluto</a>
```

10.3 Indicadores

Uma outra forma de colocar links nas páginas web, é através de indicadores. O atributo **name** permite indicar um trecho de documento como **ponto de chegada** de uma ligação hipertexto (*link*). A formatação é apresentada da seguinte forma:

```
<a name="inicio">Link usando Indicadores</a>
```

Esse exemplo, faz com que a âncora **Link usando Indicadores** seja o destino de um *link*, então, ao escrever:

```
<a href="#inicio">Topo do Documento</a>
```

Tem-se uma ligação hipertexto para um trecho deste mesmo documento.

11 Inserção de Imagens

O elemento **img** insere imagens que são apresentadas junto com os textos. Um atributo **src** deve estar presente, da seguinte forma:

```

```

Onde: local-da-imagem é o endereço do arquivo que contém a imagem que se quer inserir; podendo ser referenciada uma imagem que esteja em um outro servidor, indicando o endereço web – URL da imagem a ser inserida na página. O que **não é muito conveniente**, pois em algum momento esse endereço pode estar fora do ar e a imagem não aparecer. Portanto, o ideal é ter todas as imagens do site, salvas localmente (ou no servidor).

Se a imagem a ser inserida na página estiver no mesmo local do arquivo `.html` onde ela está sendo inserida, não precisa indicar o local, basta apenas indicar o nome do arquivo da imagem com sua extensão:

```

```

Ou, se se a imagem estiver no mesmo servidor, mas dentro de uma outra pasta:

```

```

Por exemplo, se deseja inserir na página web uma imagem chamada **arvore.gif** localizada no mesmo servidor e na mesma pasta do arquivo `.html` ao qual deseja-se inserir a imagem, a *tag* seria simplesmente assim:

```

```

Mas, se a imagem a ser inserida, estiver em outra pasta, lembre-se de indicar o caminho completo. Também é importante **verificar o nome da imagem**, pois no comando para inserir na página, o nome do arquivo **deve estar idêntico ao que a imagem está gravada**. Por isso, aconselha-se salvar os arquivos de imagens com nomes fáceis, curtos (sem acentos, espaços ou cedilhas), em letras minúsculas e, de preferência dentro de uma pasta onde estarão todas imagens do site.

As imagens usadas na web, normalmente são armazenadas em arquivos com as seguintes extensões: **.gif**, **.jpg**, **.png**, **.bmp**.

11.1 Atributos Básicos de Imagem

alt: indica um texto alternativo, descrevendo brevemente a imagem, que é apresentado no lugar da imagem nos *browsers*, caso não seja possível abri-la. Ou quando se desabilita o carregamento de imagens em *browsers* gráficos. É recomendável que este atributo esteja sempre presente!

title: é o texto que aparecerá ao “passar o mouse sobre a imagem”.

Exemplo:

```

```

Desta forma:

```

```

Resultado no Navegador: será apresentado nos *browsers* gráficos assim: **New!** e, nos *browsers* textos, assim: [Novo!!!]. E ao passar o mouse sobre a figura aparecerá a descrição **Novo Vermelho!!!**

12 Novos Elementos da HTML5

Vários novos elementos foram introduzidos na versão 5 da HTML, todos com a finalidade de facilitar a compreensão e a manutenção do código. Alguns são uma evolução natural do elemento `<div>` com foco na **semântica**, outros surgiram da necessidade de padronizar a maneira de se publicar conteúdo, como acontece hoje com as imagens. Os principais elementos dessa nova versão são apresentados nas seções a seguir.

12.1 Elementos de Estrutura do Layout

`<header>` cabeçalho da página ou de uma seção (**não confundir** com a *tag* `<head>`).

`<footer>` rodapé da página ou de uma seção.

`<main>` seção do documento que contém o conteúdo principal da página (*não* deve conter assuntos tais como topo e/ou rodapé da página e links de navegação).

`<nav>` conjunto de *links* que formam a navegação, seja o menu principal do site ou *links* relacionados ao conteúdo da página (pode conter *links* de navegação interna ou externa da página).

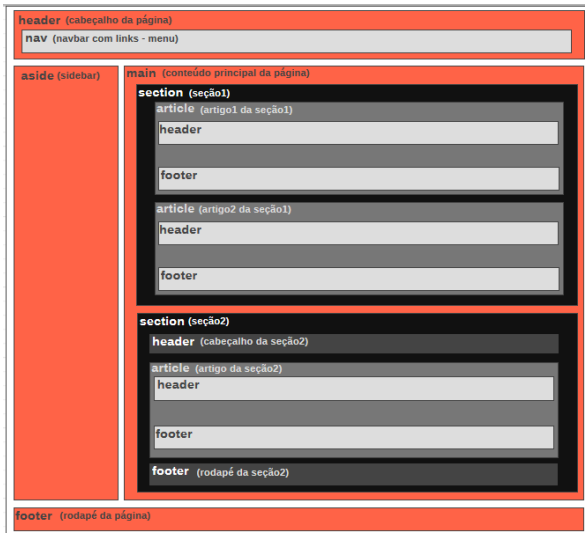
`<section>` cada seção do conteúdo, ou seja, serve para separar as seções de uma página e para identificar abordagens diferentes.

`<article>` um item do conteúdo dentro da página ou da seção, ou seja, é usado para representar conteúdos independentes de uma página. Como um *post* de um *blog*, por exemplo.

`<aside>` conteúdo relacionado ao artigo (como arquivos e *posts* relacionados em um *blog*, por exemplo). Essa *tag* é usada para a identificação de um conteúdo secundário que não seja parte da seção principal. Ela é bastante usada como uma *sidebar* (barra ou coluna lateral), alocada em uma coluna esquerda ou direita de uma página.

Na Figura 13 a seguir, é apresentado um exemplo de estrutura de *layout* de documento HTML.

Layout de documento HTML



Código-fonte para a estrutura HTML ao lado

```

Exemplo1LayoutHTML.html x
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <meta charset="utf-8">
5   <title>Titulo Barra Navegador</title>
6 </head>
7 <body>
8   <!--Início do Cabeçalho da Página-->
9   <header>
10    <nav> <!--Navbar com links = Menu-->
11      <ul> <!--Lista não ordenada = Menu -->
12        <li><a href="#"></a></li> <!--Itens da lista = links do menu -->
13        <li><a href="#"></a></li>
14        <li><a href="#"></a></li>
15      </ul>
16    </nav>
17  </header>
18  <!--Fechamento do Cabeçalho da Página-->
19  <aside></aside><!--Sidebar-->
20  <main><!--Conteúdo Principal da Página-->
21    <section><!--Seção1-->
22      <article><!--Artigo da Seção1-->
23        <header></header>
24        <footer></footer>
25      </article>
26      <article><!--Artigo2 da Seção1-->
27        <header></header>
28        <footer></footer>
29      </article>
30    </section>
31    <section><!--Seção2-->
32      <header></header><!--Cabeçalho da Seção2-->
33      <article><!--Artigo da Seção2-->
34        <header></header>
35        <footer></footer>
36      </article>
37      <footer></footer><!--Rodapé da Seção2-->
38    </section>
39  </main>
40  <footer></footer><!--Rodapé da Página-->
41 </body>
42 </html>
  
```

Figura 13: Exemplo de Distribuição de Elementos Básicos HTML (wireframe.cc)

A seguir, serão explicadas detalhadamente cada *tag* do capítulo 12.1, apresentando exemplos.

12.1.1 Tag header

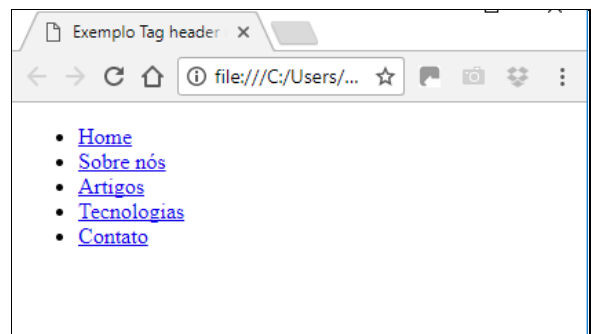
O elemento **header** especifica o cabeçalho de cada seção da página. Ele pode ser utilizado no topo da página englobando a logo da empresa, cabeçalhos dos níveis h1 a h6, formulário de busca e o menu, e ao mesmo tempo aparecer no lado direito para intitular a seção “Parceiros”, por exemplo.

Exemplo de utilização no TOPO da Página

```

1 <!DOCTYPE html>
2 <html>
3 <head>
4   <title>Exemplo Tag header no Topo da Página</title>
5 </head>
6 <body>
7   <header>
8     <nav>
9       <ul>
10        <li><a href="#">Home</a></li>
11        <li><a href="#">Sobre nós</a></li>
12        <li><a href="#">Artigos</a></li>
13        <li><a href="#">Tecnologias</a></li>
14        <li><a href="#">Contato</a></li>
15      </ul>
16    </nav>
17  </header>
18 </body>
19 </html>
  
```

Resultado no Navegador



Neste exemplo, o cabeçalho da página é simples e o **header** serve muito bem. Para cabeçalhos mais complexos que envolvam **footer** e outros **headers**, deve-se utilizar **section**.

Exemplo de utilização no MAIN da página

```

1 <!DOCTYPE html>
2 <html>
3 <head>
4   <title>Exemplo Tag header no Main da Página</title>
5 </head>
6 <body>
7   <main>
8     <article>
9       <header>
10        <h1>Título do artigo</h1>
11        <p>Data da postagem</p>
12      </header>
13      <p>Texto do artigo</p>
14    </article>
15  </main>
16 </body>
17 </html>
    
```

Resultado no Navegador



No caso de um site de artigos fica um pouco mais fácil de entender, pois tudo o que vier antes do texto em um artigo, é o cabeçalho.

12.1.2 Tag footer

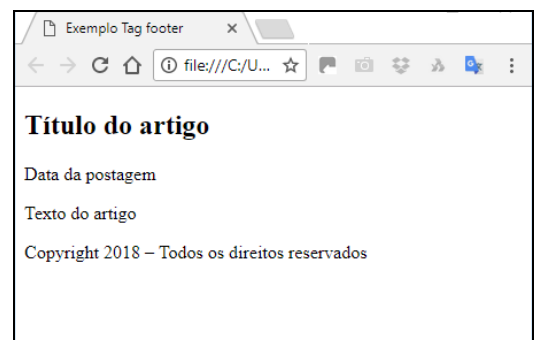
A **tag footer** é basicamente o rodapé, seja da página ou de uma seção. Em geral, contém informações sobre quem escreveu o conteúdo, links para conteúdos relacionados e informações sobre direitos autorais. Para a criação de rodapés mais robustos, com links por exemplo, é aconselhado utilizar a **tag section** e deixar a **tag footer** somente para informações mais sucintas.

Exemplo da tag footer

```

1 <!DOCTYPE html>
2 <html>
3 <head>
4   <title>Exemplo Tag footer</title>
5 </head>
6 <body>
7   <article>
8     <header>
9       <h1>Título do artigo</h1>
10      <p>Data da postagem</p>
11    </header>
12    <p>Texto do artigo</p>
13    <footer>
14      <p>Copyright 2018 - Todos os direitos reservados</p>
15    </footer>
16  </article>
17 </body>
18 </html>
    
```

Resultado no Navegador



12.1.3 Tag main

Essa *tag* é usada para identificar o **conteúdo principal da página** (só pode haver um **main** na página toda), ou seja, destina-se a marcar a seção do documento que contém o assunto principal da página. Não devem ser inclusos dentro desse elemento conteúdos tais como o topo da página, o rodapé da página e os links de navegação (menu).

12.1.4 Tag nav

A *tag nav* identifica os links para navegação seja para outra página ou para seções da mesma página. É muito utilizada para a criação de **menu de navegação**, ou seja, destina-se a marcar o mecanismo principal de navegação da página. Pequenos grupos de links, tais como os existentes no rodapé da página apontando para política do site, termos de serviços, homepage e página de direitos autorais, não precisam ser marcados com esse elemento.

Existem dois contextos distintos de uso desse elemento:

- Quando **dentro** de um elemento **article**, seu conteúdo deve estar relacionado ao conteúdo daquele elemento.
- Quando **fora** do elemento **article**, seu conteúdo deve estar relacionado ao conteúdo do site ou da página, como um banner de propaganda ou links de navegação.

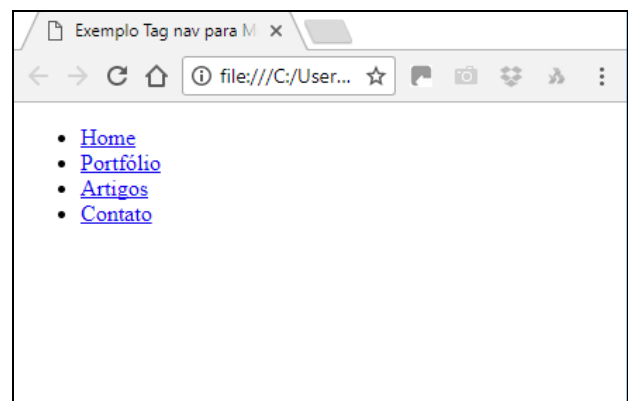
Veja a seguir um exemplo de utilização.

Exemplo da tag nav

```

1 <!DOCTYPE html>
2 <html>
3 <head>
4   <title>Exemplo Tag nav para Menu</title>
5 </head>
6 <body>
7   <nav>
8     <ul>
9       <li><a href="#">Home</a></li>
10      <li><a href="#">Portfólio</a></li>
11      <li><a href="#">Artigos</a></li>
12      <li><a href="#">Contato</a></li>
13    </ul>
14  </nav>
15 </body>
16 </html>
    
```

Resultado no Navegador



12.1.5 Tag section

Assim como antes da HTML5 costumava-se dividir a página em blocos de conteúdo com **divs** (não é mais tão utilizada), atualmente se faz o mesmo utilizando a **tag section**. A diferença entre a **section** e a **div** é a parte de semântica, pois toda seção precisa ter um significado como conteúdo, diferentemente da **div** no XHTML. A **section** serve para separar as seções de uma página e para identificar abordagens diferentes.

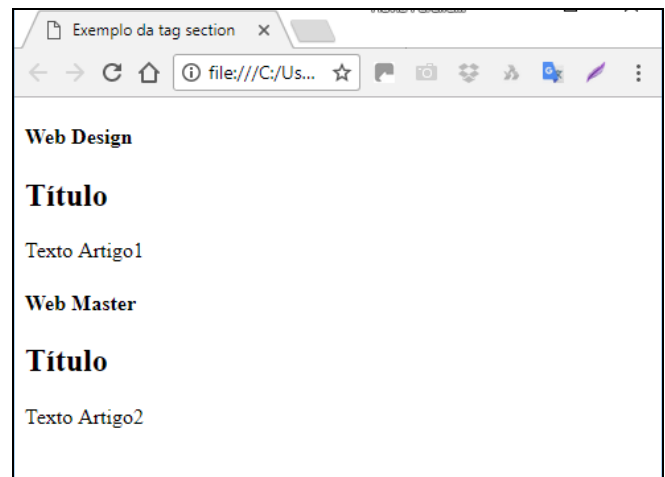
section é um elemento do tipo nível de bloco e destina-se a criar um container genérico para conteúdos. Veja um exemplo de utilização da **tag section**.

Exemplo da tag section

```

1 <!DOCTYPE html>
2 <html>
3 <head>
4   <title>Exemplo da tag section</title>
5 </head>
6 <body>
7   <div>
8     <section>
9       <h4>Web Design</h4>
10      <article>
11        <h2>Título</h2>
12        <p>Texto Artigo1</p>
13      </article>
14    </section>
15    <section>
16      <h4>Web Master</h4>
17      <article>
18        <h2>Título</h2>
19        <p>Texto Artigo2</p>
20      </article>
21    </section>
22  </div>
23 </body>
24 </html>
    
```

Resultado no Navegador



12.1.6 Tag `article`

A tag **article** é muito utilizada em sites de notícias, artigos, blogs e como o nome já é bem sugestivo, serve para delimitar um espaço para a inserção de um artigo, podendo ser referenciado via RSS² (sigla em inglês para *Rich Site Summary* ou *Really Simple Syndication*, ou seja, uma forma simplificada de apresentar o conteúdo de um site – esse conceito será visto com mais detalhes em capítulos adiante).

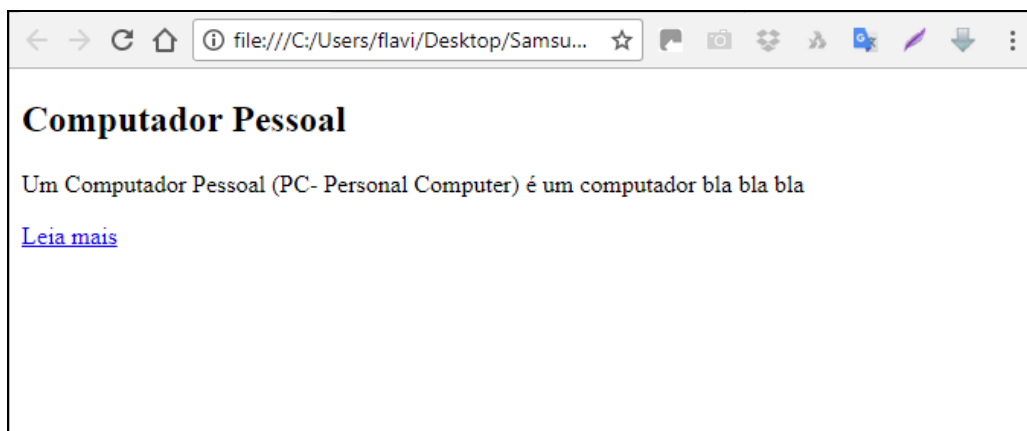
article é um elemento do tipo nível de bloco e destina-se a marcar conteúdos que possam ser distribuídos, reutilizados e entendidos isoladamente, como um post em um fórum, um comentário em um blog, ou seja, qualquer conteúdo independente de uma página. Exemplo de **article**:

Exemplo da tag `article`

```

1 <!DOCTYPE html>
2 <html>
3 <head>
4   <title>Exemplo da tag article</title>
5 </head>
6 <body>
7   <article>
8     <h1>Computador Pessoal</h1>
9     <p>Um Computador Pessoal (PC- Personal Computer) é um computador bla bla bla</p>
10    <a href="#" title="Leia mais">Leia mais</a>
11  </article>
12 </body>
13 </html>
    
```

Resultado no Navegador



Observação: Para saber quando utilizar **article**, isole o texto do resto da página. Se ele continuar fazendo sentido, estão use **article**.

article e **section** têm um relacionamento bem parecidos, o que pode confundir a princípio. Pois, assim como é possível colocar **article** dentro de **section**, o inverso também é possível.

DICA: Uma forma de ajudar a fixar a diferença entre essas *tags* é pensar que **section divide a página em blocos de conteúdo, enquanto article engloba o conteúdo em si**. Dessa forma talvez fique mais fácil de memorizar 😊

² Um documento RSS é feito na linguagem XML e geralmente exibe um grande volume de informações existente em uma página na internet de forma resumida. Pela característica de se alimentar de notícias, os documentos RSS também são chamados de *Feeds*.

12.1.7 Tag *aside*

Destina-se a marcar o conteúdo relacionado a outro conteúdo. Em geral, é usado para marcar barras laterais de conteúdos, blocos de conteúdos relacionados a outro conteúdo, mas visualmente separados. Essa *tag* é usada para a identificação de um conteúdo secundário que não seja parte da seção principal, ou seja, é utilizado quando precisamos criar um conteúdo de apoio/adicional ao conteúdo principal. Ela é bastante usada como uma *sidebar* (barra lateral), alocada em uma coluna esquerda ou direita de uma página.

Existem dois contextos distintos de uso desse elemento:

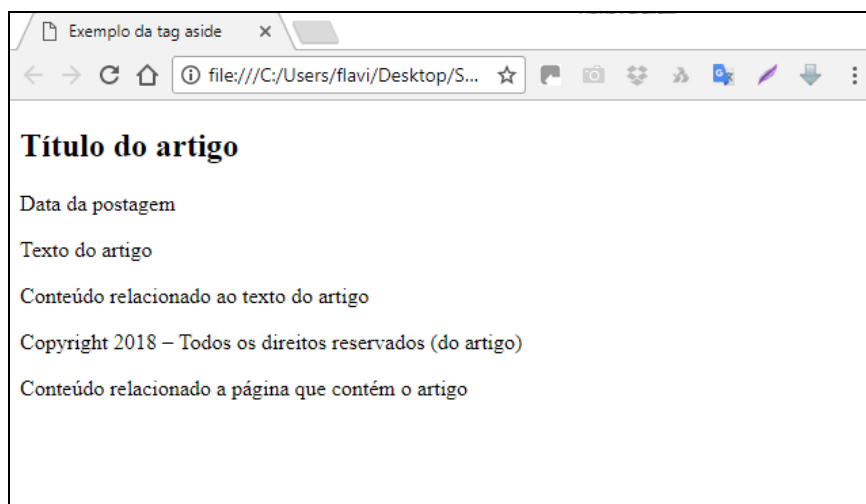
- **Dentro** de um elemento **article**: seu conteúdo deve estar relacionado ao conteúdo daquele elemento.
- **Fora** do elemento **article**: seu conteúdo deve estar relacionado ao conteúdo do site ou da página, como um banner de propaganda ou links de navegação.

Exemplo da tag *aside*

```

1  <!DOCTYPE html>
2  <html>
3  <head>
4  |   <title>Exemplo da tag aside</title>
5  </head>
6  <body>
7  |   <article>
8  |     <header>
9  |       <h1>Título do artigo</h1>
10 |       <p>Data da postagem</p>
11 |     </header>
12 |     <p>Texto do artigo</p>
13 |     <aside> <!--dentro do article-->
14 |       <p>Conteúdo relacionado ao texto do artigo</p>
15 |     </aside>
16 |     <footer>
17 |       <p>Copyright 2018 - Todos os direitos reservados (do artigo)</p>
18 |     </footer>
19 |   </article>
20 |   <aside> <!--fora do article-->
21 |     <p>Conteúdo relacionado a página que contém o artigo</p>
22 |   </aside>
23 </body>
24 </html>
--
    
```

Resultado no Navegador



13 Semântica

Semântica é um aspecto de alta relevância em marcação HTML e diz respeito ao correto uso do elemento para marcar o conteúdo. Por exemplo: se o conteúdo é um cabeçalho de nível 2, ele deverá ser marcado com o elemento **h2**; se é o item de uma lista não ordenada, deverá ser marcado com o elemento **li** contido em um elemento **ul**.

O conceito de semântica é importante que seja utilizado e faz parte das melhores práticas de desenvolvimento web, chamado de **HTML semântico**. Ou seja, usar os seus elementos ou *tags* para o propósito que foram projetados. Assim, deve-se ter cuidado especial em usar uma etiqueta **p** para parágrafos, **h1** para títulos principais de uma seção, **img** para imagens, **table** para tabelas e assim por diante.

Parece simples, não? Mas é possível ver **p** sendo usado para um título ou **table** para fazer o *layout* de um site inteiro (não faça isso!!!). **Semântica é significado, e um código HTML semântico utiliza as tags corretamente de acordo com o seu propósito**. Esta versão 5.0 da HTML traz mudanças significativas com a criação de novas *tags* que possibilitam unir as partes da página de forma lógica. **O que não significa necessariamente um código mais enxuto, pois o objetivo é torná-lo organizado**.

É importante notar que em documentos HTML o espaço em branco é quase que completamente ignorado. Por exemplo, tanto `<p>Texto</p>` quanto `<p> Texto </p>`, produzirão o mesmo resultado na página web. Ou seja, todo o espaço em branco no código HTML será reduzido a apenas um espaço visível no navegador, em cada lado da palavra (é possível mudar este comportamento utilizando CSS).

Tags de início e de fim devem estar corretamente aninhadas. O aninhamento apropriado é uma regra que deve ser obedecida para que se possa escrever códigos válidos, isto é, *tags* de fim devem ser posicionadas na ordem contrária das *tags* de início e, cuidando também da endentação do código (recuos). Veja os exemplos abaixo:

→ Este é um exemplo de código **válido**:

```
<em>Eu <strong>realmente</strong> quis dizer aquilo</em>.
```

→ Este é um exemplo de código **inválido**:

```
<em>Eu <strong>realmente</em> quis dizer aquilo</strong>.
```

Observe que, no exemplo válido, a *tag* de fim para cada elemento aninhado é sempre colocada antes da *tag* de fim do elemento no qual ele está aninhado.

Há algumas coisas necessárias de se compreender de imediato sobre as *tags*. Em primeiro lugar, repare que há uma hierarquia, de forma que as *tags* que abrem primeiro, fecham por último englobando as outras. Isso é feito para que se crie um esquema de nós que através do **W3C** é denominado **DOM** (*Document Object Model*) do HTML. As *tags* englobadoras são chamadas de **nós pais** e as *tags* englobadas são chamadas de **nós filhos**.

É possível validar uma página HTML que esteja na internet, para saber se está ou não no padrão W3C. Para isso basta acessar o site validator.w3.org, colocar a URL³ preenchendo o campo *address* com o endereço do site que deseja verificar e clicar no botão *check*.

Referências a caracteres com nome (também chamados de entidades) são usadas para imprimir caracteres que têm um significado especial em HTML. Por exemplo, a HTML interpreta os sinais de menor e maior como delimitadores de *tags*. Então, quando deseja-se exibir um sinal de maior em um texto de uma página web, pode-se usar uma referência ao **nome do caractere**. Há quatro referências comuns a caracteres que se deve conhecer, apresentadas na tabela abaixo:

Código HTML	Significado	O que aparece na página web
<code>&gt;</code>	sinal de maior	>
<code>&lt;</code>	sinal de menor	<
<code>&amp;</code>	E Comercial	&
<code>&quot;</code>	aspas duplas	"



#FicaDica: Não existem programas em HTML, pois HTML não é uma linguagem de programação, mas de marcação. Portanto, a rigor, não existem "programadores" de HTML



³ URL: *Uniform Resource Locator* (Localizador Uniforme de Recursos ou Localizador Padrão de Recursos) é o formato de atribuição universal para designar um recurso na rede, ou seja, é o endereço de um recurso (um arquivo, uma impressora etc.), disponível em uma rede; seja a Internet, ou uma rede corporativa, uma Intranet.

14 Outras Tags Importantes

Existem 108 elementos previstos nas especificações do W3C para a HTML5. Nesta apostila estamos estudando alguns dos mais importantes. Neste capítulo serão explicadas algumas *tags* que ainda não foram vistas nos capítulos anteriores.

14.1 *div*

Esse é um elemento do tipo **nível de bloco** e se destina a criar um container geral para outros elementos. É um elemento sem destinação específica, ou seja, diferentemente dos elementos **h1** e **p**, por exemplo, que se destinam a marcar cabeçalhos de nível 1 e parágrafos, respectivamente, o elemento **div** pode conter (marcar) qualquer outro elemento em qualquer quantidade e até mesmo outros elementos **div**.

Elementos com essas características são chamados de elementos **sem valor semântico**.

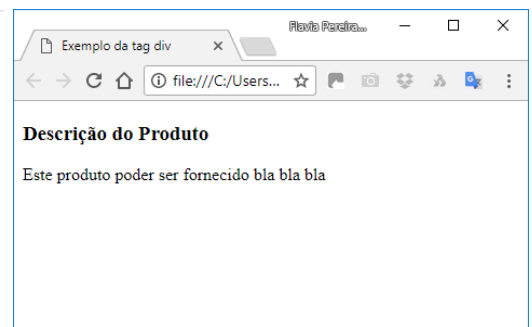
Exemplo da tag aside

```

1 <!DOCTYPE html>
2 <html>
3 <head>
4   <title>Exemplo da tag div</title>
5 </head>
6 <body>
7   <div>
8     <h3>Descrição do Produto</h3>
9     <p>Este produto poder ser fornecido bla bla bla</p>
10  </div>
11 </body>
12 </html>

```

Resultado no Navegador



Nesse exemplo, a presença do container **div** em nada altera a renderização desse trecho de código. Para efeito de renderização é como se o elemento **div** não estivesse presente na marcação.

Então para que serve o *div*? Conforme será abordado adiante, o **div** poderá servir, entre outras inúmeras coisas, como uma espécie de caixa com bordas e fundo colorido, para dar um formato visual atraente ao texto dentro dele. Mas isso deverá ser feito com o uso de **CSS**.

14.2 span

Esse é um elemento do tipo *inline* e destina-se a criar um container geral para outros elementos *inline*. É um elemento sem destinação específica, ou seja, da mesma forma que o `div`, esse elemento pode conter (marcar) qualquer outro elemento *inline* em qualquer quantidade e até mesmo outros elementos **span**.

Tal como o elemento `div`, esse elemento é **sem valor semântico**. A diferença do **span** para o **div**, é que se trata de um elemento *inline*, enquanto **div** é um elemento nível de bloco.

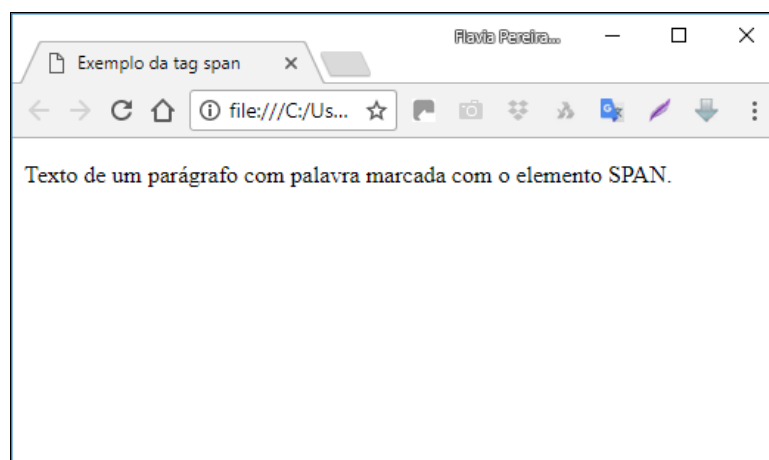
Exemplo da tag span

```

1 <!DOCTYPE html>
2 <html>
3 <head>
4   <title>Exemplo da tag span</title>
5 </head>
6 <body>
7   <p>Texto de um parágrafo com <span>palavra</span> marcada com o elemento SPAN.</p>
8 </body>
9 </html>

```

Resultado no Navegador



Nesse exemplo, a presença do container **span** em nada altera a renderização do texto “palavra” contido no parágrafo. Para efeito de renderização, é como se o elemento **span** não estivesse presente na marcação.

Então para que serve o span? Conforme será visto adiante, o **span** poderá servir, entre outras coisas, como uma espécie de caixa para que seu conteúdo seja renderizado em uma cor diferente, ou com um tamanho de letra diferente. Isso deverá ser feito com uso de **CSS**.

14.3 b e strong

Estes elementos são do tipo *inline* e os dois causam o mesmo efeito de renderização, mas têm **valor semântico diferente**. Quando aplicados a um pequeno trecho de texto ou palavra, ambos causam renderização em **negrito**.

Qual a diferença entre eles? Use o elemento **b** para dar o aspecto visual **negrito** e use o elemento **strong** para dar **forte ênfase**. Para melhor entender a diferença, suponha um internauta com visão normal e outro com deficiência visual navegando usando um **leitor de tela**. A marcação com o elemento **b** só terá efeito negrito para o internauta com visão normal e a marcação com o elemento **strong** terá efeito negrito e forte ênfase para os dois, pois além de negrito, o leitor de tela lerá o conteúdo de **strong** com voz em forte ênfase, e o internauta deficiente visual saberá que aquele trecho foi destacado pelo autor do documento.

14.4 i e em

Estes elementos são do tipo *inline* e os dois causam o mesmo efeito de renderização, mas têm **valor semântico diferente**. Quando aplicados a um pequeno trecho de texto ou palavra, ambos causam renderização em **itálico**.

Qual a diferença entre eles? Use o elemento **i** para dar o aspecto visual **itálico** e use o elemento **em** para dar **ênfase**. A justificativa de uso é semelhante àquela descrita para os elementos **b** e **strong** no item anterior.

14.5 small

Esse é um elemento do tipo *inline* e se destina a marcar pequenos trechos de texto relacionados com o conteúdo do texto no qual foram marcados. Notas sobre direitos autorais no rodapé do site e pequenas notas de advertência são conteúdos para se marcar com o elemento **small**. A renderização de textos marcados com esse elemento é feita com tamanho de fonte menor do que o do texto no qual foram marcados.

Exemplo da tag small

```

1  <!DOCTYPE html>
2  <html>
3  <head>
4  | <title>Exemplo da tag small</title>
5  </head>
6  <body>
7  | <h3>Valores das diárias</h3>
8  | <ul>
9  | | <li>Quarto de solteiro: R$245.00 <small>café da manhã incluso</small></li>
10 | | <li>Quarto de casal: R$320.00 <small>café da manhã incluso</small></li>
11 | </ul>
12 </body>
13 </html>

```

Resultado no Navegador



14.6 code

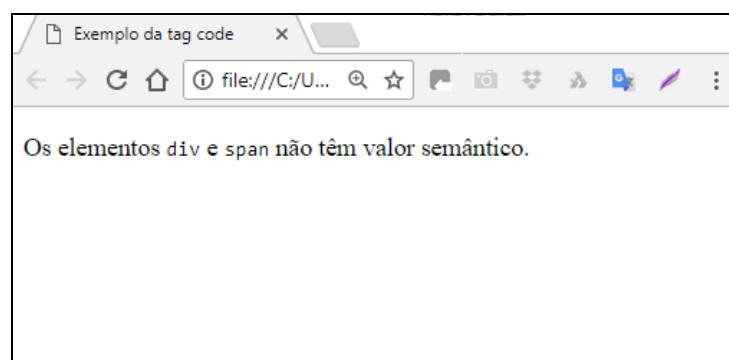
Esse é um elemento do tipo *inline* e se destina a marcar trechos de código de computador (código-fonte). Nomes de arquivos, nomes de softwares, nomes de elementos HTML, trechos de código, são conteúdos para se marcar com o elemento **code**. A renderização de textos marcados com esse elemento, normalmente é feita com tipo de fonte monoespaçada, diferente da fonte do texto no qual foram marcados.

Exemplo da tag code

```

1 <!DOCTYPE html>
2 <html>
3 <head>
4 | <title>Exemplo da tag code</title>
5 </head>
6 <body>
7 | <p>Os elementos <code>div</code> e <code>span</code> não têm valor semântico.</p>
8 </body>
9 </html>
    
```

Resultado no Navegador



14.7 audio

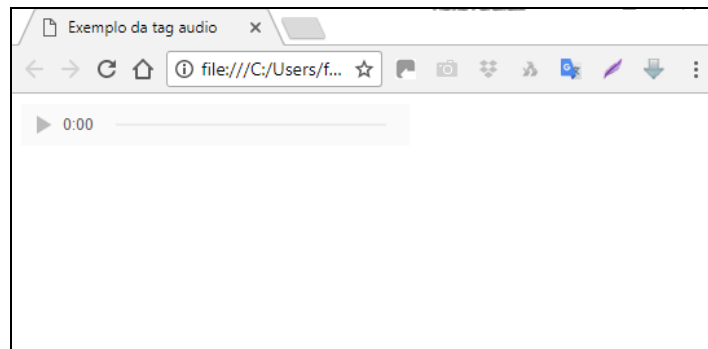
Esse é um elemento do tipo **nível de bloco** e destina-se a inserir um som ou *stream* de áudio.

Exemplo da tag audio

```

1  <!DOCTYPE html>
2  <html>
3  <head>
4  |   <title>Exemplo da tag audio</title>
5  </head>
6  <body>
7  |   <audio controls>
8  |   |   <source src="som.ogg" type="audio/ogg">
9  |   |   <source src="som.mp3" type="audio/mpeg">
10 |   |   <source src="som.wav" type="audio/wave">
11 |   |   <p>Seu navegador não suporta o elemento audio da HTML5. Faça <a href="som.zip"> download de som.zip</a></p>
12 |   </audio>
13 </body>
14 </html>
    
```

Resultado no Navegador



Para esse exemplo acima funcionar, é necessário ter os arquivos **som.ogg**, **som.mp3** e **som.wav**, dentro das respectivas pastas citadas.

14.8 video

Esse é um elemento do tipo **nível de bloco** e destina-se a inserir um vídeo.

Exemplo da tag video

```

1  <!DOCTYPE html>
2  <html>
3  <head>
4  |   <title>Exemplo tag video</title>
5  </head>
6  <body>
7  |   <video controls>
8  |   |   <source src="video.ogv" type="video/ogv">
9  |   |   <source src="video.mp4" type="video/mp4">
10 |   |   <source src="video.webm" type="video/webm">
11 |   |   <!-- Código (X)HTML para inserção de video com Flash -->
12 |   |   <p>Seu navegador não suporta o elemento video da HTML5. Faça <a href="video.mp4"> download do video</a></p>
13 |   </video>
14 </body>
15 </html>
    
```

14.9 canvas

Esse é um elemento do tipo **nível de bloco** e destina-se a criar uma área de desenho.

Exemplo da tag canvas

```

1 <!DOCTYPE html>
2 <html lang="pt-br">
3 <head>
4   <title>Elemento canvas da HTML5</title>
5   <meta charset="utf-8">
6   <style rel="stylesheet" /*Usando CSS incorporado*/
7     canvas{
8       width: 500px;
9       height: 250px;
10      border: 2px solid black;
11      background: #ffc;
12    }
13 </style>
14 </head>
15 <body>
16   <canvas>
17     <p>Conteúdo alternativo para navegadores que não suportam canvas.</p>
18   </canvas>
19 </body>
20 </html>
    
```

Resultado no Navegador



As técnicas para desenhar em **canvas** envolvem uma linguagem própria e JavaScript, e não serão abordadas nesta apostila.

15 Atributos

A seguir, serão explicados alguns atributos importantes e muito utilizados na criação de páginas web.

15.1 id

Este atributo **identifica uma tag**, ou seja, ele é um identificador e pode ser usado em várias *tags*, mas **ele é único e seu valor não pode ser repetido no mesmo documento**. Usa-se para selecionar um elemento específico, ou seja, o **id** de um elemento deve ser exclusivo dentro de uma página, então o seletor de identificação é usado para selecionar um elemento exclusivo!

Para selecionar um elemento com um **id** específico, usa-se um caractere *hash* (**#**), seguido do **id** do elemento. Visualmente não tem nenhum efeito para o usuário final. Mas vai ser muito útil quando estivermos estudando **CSS**.

15.2 class

O seletor de classe seleciona elementos com um atributo de classe específico. Para selecionar elementos com uma classe específica, escreva um caractere de ponto (**.**), seguido do nome da classe. É outro atributo destinado ao uso de **CSS** que também estudaremos adiante.

15.3 style

O atributo de estilo é usado para especificar o estilo de um elemento, como cor, fonte, tamanho etc. E também é uma das maneiras de importar **CSS** nas páginas **HTML**. É um atributo que se utiliza bastante, mas ele deve conter instruções de **CSS** dentro para que tenha efeito, então utilizaremos este atributo mais adiante.

16 Tabelas

A manipulação de tabelas, mesmo em editores, é trabalhosa. A maior diferença entre tabelas em HTML e em editores como o MS-Word, entretanto, é o fato das tabelas em HTML serem definidas apenas **em termos de linhas e não de colunas**. Mas isso será percebido no decorrer deste capítulo.

As tabelas por muito tempo foram utilizadas para fazer os layouts das páginas web. Hoje em dia **não se deve usar tabelas para layouts**, ou seja, usa-se tabelas em HTML apenas se for necessário ter uma tabela na página. Pesquise sobre o conceito de *tableless* para saber mais.

16.1 Elementos básicos de tabelas

`<table>...</table>` delimita uma tabela. Um atributo básico é **border**, que indica a apresentação da borda.

```
<table border="borda">
...
</table>
```

16.2 Títulos, linhas e elementos

`<caption>...</caption>`: define o título da tabela

`<tr>...</tr>`: delimita uma linha

`<th>...</th>`: define um cabeçalho para colunas ou linhas (dentro de `<tr>`)

`<td>...</td>`: delimita um elemento ou célula (dentro de `<tr>`)

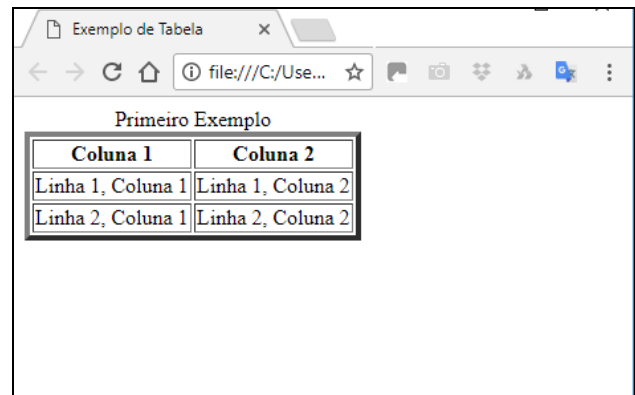
16.3 Exemplo de uma tabela simples

Exemplo da tag table

```

1 <!DOCTYPE html>
2 <html lang="pt-br">
3 <head>
4   <title>Exemplo de Tabela</title>
5   <meta charset="utf-8">
6 </head>
7 <body>
8   <table border=4>
9     <caption>Primeiro Exemplo</caption>
10    <tr>
11      <th>Coluna 1</th>
12      <th>Coluna 2</th>
13    </tr>
14    <tr>
15      <td>Linha 1, Coluna 1</td>
16      <td>Linha 1, Coluna 2</td>
17    </tr>
18    <tr>
19      <td>Linha 2, Coluna 1</td>
20      <td>Linha 2, Coluna 2</td>
21    </tr>
22  </table>
23 </body>
24 </html>
    
```

Resultado no Navegador



Coluna 1	Coluna 2
Linha 1, Coluna 1	Linha 1, Coluna 2
Linha 2, Coluna 1	Linha 2, Coluna 2

16.4 Títulos compreendendo mais de uma coluna ou linha

É possível englobar colunas e linhas, através dos atributos **colspan** (para colunas) e **rowspan** (para linhas):

```

<table border=1>
  <tr><th colspan=2>colunas 1 e 2</th></tr>
  <tr><td>linha1, coluna1</td><td>linha1, coluna2</td></tr>
  <tr><td>linha2, coluna1</td><td>linha2, coluna2</td></tr>
  <tr><th rowspan=3>3 linhas</th><td>uma linha</td></tr>
  <tr><td>duas linhas</td></tr>
  <tr><td>tres linhas</td></tr>
</table>
    
```

colunas 1 e 2	
linha1, coluna1	linha1, coluna2
linha2, coluna1	linha2, coluna2
3 linhas	uma linha
	duas linhas
	tres linhas

Neste exemplo, vemos que o cabeçalho **colunas 1 e 2** compreende duas colunas (**colspan=2**) e o cabeçalho **3 linhas** compreende, por sua vez, 3 linhas (**rowspan=3**).

16.5 Tabelas sem borda

```
<table border="0">  
...  
</table>
```



Dica: A formatação de tabelas é complicada, e o código fonte chega a ser quase ininteligível quando montamos tabelas complexas e fazemos uso de seus diversos atributos. A melhor opção, nesses casos, sem dúvida, é usar os editores WYSIWYG. Alguns editores de modo texto têm uma interface gráfica que ajuda na criação de tabelas, mas a edição de tabelas já existentes precisa ser feita à mão.

17 Formulários HTML

Os formulários HTML são estruturas que permitem que usuários submetam dados a uma página. Esses dados podem ser tratados e/ou armazenados dependendo da aplicação.

Formulários estão presentes na Internet para possibilitar cadastros, pesquisas, envio de comentários, aumentando o poder de interação com os visitantes dos sites. Um formulário HTML é uma página web que contém, além de texto, elementos especiais chamados **campos**, representados por caixas de checagem, botões, caixas de seleção, campos de textos etc.

Um formulário funciona da seguinte maneira: os usuários preenchem os campos do formulário, submetendo-o em seguida, a algum agente de processamento (clicando em um botão do tipo **submit** para enviar os dados do formulário). Neste momento, todas as informações fornecidas são enviadas a um programa escrito especialmente para processar esses dados de acordo com alguma necessidade e especificação. Em alguns casos, os dados são gravados em um Banco de Dados (BD), em outros casos uma nova página é construída, em outros ainda, as informações são encaminhadas via e-mail. A *tag* de formulários é a **<form>** que possui alguns atributos essenciais para seu funcionamento como apresentados a seguir.

Vale salientar que a *tag* **<form>** sozinha não tem nenhum efeito visual para o usuário final, ela apenas agrupa todos os campos de **input** para que sejam enviados juntos.

17.1 Criando um form

Os **forms** fazem parte do código HTML, portanto devem ser definidos como as *tags* desta linguagem. Sempre com uma *tag* de abertura e outra de fechamento, inserindo o conteúdo entre estas duas.

Exemplo:

```
<form action="pagina.php" method="post"> <!-- tag de abertura -->
  <!--conteúdo do formulário (campos) -->
</form> <!-- tag de fechamento -->
```

17.2 Parâmetros do form

action: é o script ou página para onde os dados serão submetidos. Neste script que, normalmente, os dados são tratados, ou seja, indica para onde os dados do formulário estão sendo enviados.

method: é o método de envio dos dados. Pode ter dois valores, apresentados a seguir.

17.2.1 get

Passa os valores pela URL, ou seja, é possível ver as variáveis passadas na URL da página destino definida no campo **action**. Na Figura 14 abaixo, é apresentada uma URL com um exemplo de uso do método **get**. Ao digitar a palavra “pizza” no campo de busca e clicar no botão Buscar, na barra de endereço aparece a informação que está sendo passada, no caso, a palavra “pizza”.



Figura 14: Exemplo de Uso do Método Get

17.2.2 post

Passa as variáveis de maneira transparente para o usuário, ou seja, usando este método, não serão visíveis as informações que estão sendo passadas, como no método get.

Observação: A escolha entre os dois métodos depende de como foi implementado o lado do servidor que vai receber as informações. Veja mais diferenças entre **get** e **post** em:

https://developer.mozilla.org/en-US/docs/Web/Guide/HTML/Forms/Sending_and_retrieving_form_data

17.3 Elementos do form

Pode-se inserir vários tipos de entrada de dados em um formulário, a maioria delas definida pela *tag* **input**. Todo elemento possui um parâmetro **name** que é utilizado para identificar a variável onde o dado está contido no script destino. Lembrando que estes elementos devem ser inseridos entre as *tags* **<form>** **</form>**.

O campo **input** é um dos campos utilizados para atribuir valores aos formulários e ele tem um atributo chamado **type** que aceita vários valores diferentes. Este atributo define o **tipo de campo** que será apresentado ao usuário. Detalhes sobre os campos e atributos, são explicados a seguir.

17.3.1 Campo Texto

Campo para entrada de textos comuns: **input type="text"**

Declaração: `<input type="text" name="" value="" size="" maxlength="">`

Parâmetros:

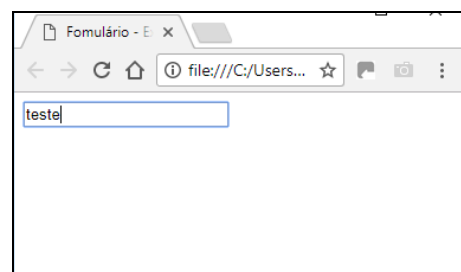
- **name:** usado para identificar os campos dos formulários. Ao realizar um “submit” de um formulário, o atributo **name** é o identificador dentro de uma requisição GET ou POST no servidor, ou seja, é o nome que será usado para referenciar cada campo do formulário.
- **value:** utilizado quando há necessidade de se pré-definir um valor para o elemento. Este valor pode ser normalmente alterado pelo usuário.
- **size:** tamanho do elemento em caracteres na página HTML (que será exibido na tela).
- **maxlength:** tamanho máximo do texto que pode ser inserido no elemento.

Exemplo campo tipo texto

```

1 <!DOCTYPE html>
2 <html lang="pt-br">
3 <head>
4   <title>Fomulário - Exemplo Campo Text</title>
5   <meta charset="utf-8">
6 </head>
7 <body>
8   <form>
9     <input type="text" name="" value="" size="" maxlength="">
10  </form>
11 </body>
12 </html>
    
```

Resultado no Navegador



17.3.2 Campo Senha

Tipo de campo idêntico ao anterior, mas quando o usuário digita, os caracteres são substituídos por "*" (asteriscos): `input type="password"`

Atenção: o campo senha não possui nenhum tipo de criptografia, apenas coloca uma máscara no texto inserido, trocando os caracteres digitados por asteriscos.

Declaração: `<input type="password" name="" value="" size="" maxlength="">`

Parâmetros:

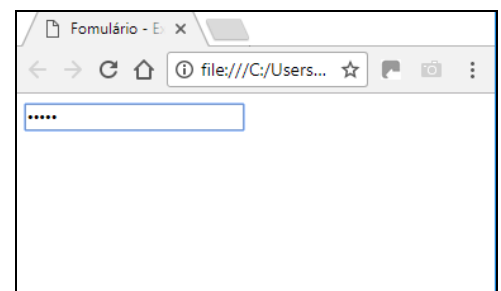
- **value:** utilizado quando há necessidade de se pré-definir um valor para o elemento. Este valor pode ser normalmente alterado pelo usuário.
- **size:** tamanho do elemento em caracteres na página HTML (que será exibido na tela).
- **maxlength:** tamanho máximo do texto que pode ser inserido no elemento.

Exemplo campo tipo senha

```

1 <!DOCTYPE html>
2 <html lang="pt-br">
3 <head>
4   <title>Fomulário - Exemplo Campo Password</title>
5   <meta charset="utf-8">
6 </head>
7 <body>
8   <form>
9     <input type="password" name="" value="" size="" maxlength="">
10  </form>
11 </body>
12 </html>
    
```

Resultado no Navegador



17.3.3 Botão Rádio (radio button)

Utilizado para entradas de múltipla escolha, onde o usuário só pode **escolher uma única opção** (ou seja, é um campo de escolha única): `input type="radio"`

Para que o interpretador saiba que as opções fazem parte do mesmo grupo, e permita que só uma seja selecionada, basta *colocar o mesmo nome no parâmetro **name*** dos botões rádio.

Declaração: `<input type="radio" name="" value="" checked="">`

Parâmetros:

- **value:** valor que será passado à página destino quando o formulário é submetido, se este elemento estiver selecionado.
- **checked:** se for declarado, o elemento terá seu estado inicial como selecionado.

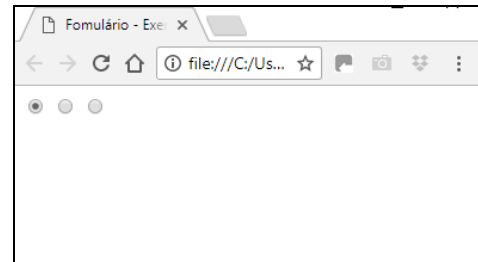
Exemplo a seguir.

Exemplo campo tipo radio

```

1 <!DOCTYPE html>
2 <html lang="pt-br">
3 <head>
4   <title>Fomulário - Exemplo Campo Radio</title>
5   <meta charset="utf-8">
6 </head>
7 <body>
8   <form>
9     <input type="radio" name="" value="" checked="">
10    <input type="radio" name="" value="">
11    <input type="radio" name="" value="">
12  </form>
13 </body>
14 </html>
    
```

Resultado no Navegador



17.3.4 Botão de Checagem (check box)

Utilizado para entradas de **múltipla escolha** onde o usuário *pode escolher várias opções*. Cada opção deve ter um nome independente: `input type="checkbox"`

Declaração: `<input type="checkbox" name="" value="" checked="">`

Parâmetros:

- **value:** valor que será passado à página destino quando o formulário é submetido se este elemento estiver marcado.
- **checked:** se for declarado, o elemento terá seu estado inicial como marcado.

Exemplo campo tipo checkbox

```

1 <!DOCTYPE html>
2 <html lang="pt-br">
3 <head>
4   <title>Fomulário - Exemplo Campo CheckBox</title>
5   <meta charset="utf-8">
6 </head>
7 <body>
8   <form>
9     <input type="checkbox" name="" value="" checked="">
10    <input type="checkbox" name="" value="">
11    <input type="checkbox" name="" value="">
12  </form>
13 </body>
14 </html>
    
```

Resultado no Navegador



17.3.5 Botão Submeter (*submit*)

Botão que submete o formulário à página destino especificada no parâmetro **action** do **form**, ou seja, ao clicar no botão **submit**, envia os dados do formulário: `input type="submit"`

Declaração: `<input type="submit" name="" value="">`

Parâmetros:

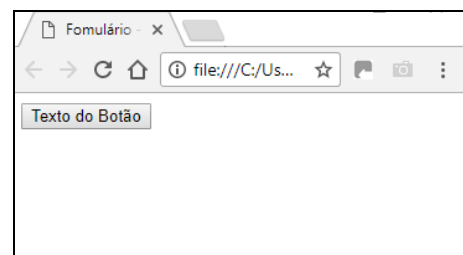
- **value:** texto que aparecerá no botão.

Exemplo botão submit

```

1 <!DOCTYPE html>
2 <html lang="pt-br">
3 <head>
4   <title>Formulário - Exemplo Botão Submit</title>
5   <meta charset="utf-8">
6 </head>
7 <body>
8   <form>
9     <input type="Submit" name="" value="Texto do Botão">
10  </form>
11 </body>
12 </html>
    
```

Resultado no Navegador



17.3.6 Botão Reset

Reseta o conteúdo dos campos do formulário, ou seja, volta todos os campos do formulário para os valores iniciais: `input type="reset"`

Ao clicar nesse botão, os campos do formulário voltam aos valores especificados nos parâmetros **value** de cada um dos campos.

Declaração: `<input type="reset" name="" value="">`

Parâmetros:

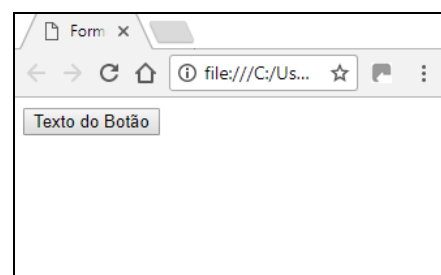
- **value:** texto que aparecerá no botão.

Exemplo botão reset

```

1 <!DOCTYPE html>
2 <html lang="pt-br">
3 <head>
4   <title>Formulário - Exemplo Botão Reset</title>
5   <meta charset="utf-8">
6 </head>
7 <body>
8   <form>
9     <input type="Reset" name="" value="Texto do Botão">
10  </form>
11 </body>
12 </html>
    
```

Resultado no Navegador



17.3.7 Select

select é um componente muito utilizado para apresentar listas onde um ou mais itens serão selecionados. Esses itens são representados pela *tag* **<option>**, ou seja, é utilizado para selecionar uma ou mais opções de uma lista predefinida.

Declaração:

```
<select name="" size="" multiple="">
  <option value="">
    texto da opção1
  </option>

  <option value="">
    texto da opção2
  </option>
</select>
```

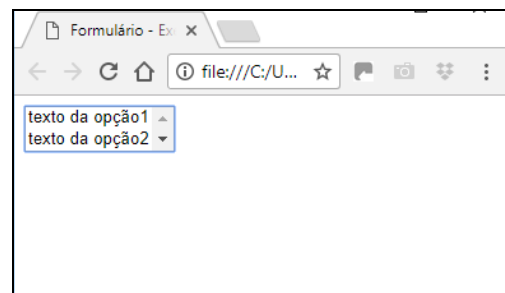
Parâmetros:

- **multiple:** se existir, permite que sejam selecionadas múltiplas opções através das teclas **Ctrl** ou **Shift**.
 - **size:** número de opções exibidas por vez na tela. Se o **size** estiver setado para “1”, que é o valor *default* (padrão) e não existir o parâmetro **multiple**, o elemento é exibido como um **Combo Box**. Caso contrário, é exibido como um **Select List**.
 - **option:** cada subtag **option** adiciona uma opção ao elemento.
 - **value:** o **value** de cada **option** é o valor passado caso aquela opção seja selecionada.
- Observação:** O texto da opção deve ser especificado entre as *tags* **<option>** **</option>**

Exemplo campo select

```
1 <!DOCTYPE html>
2 <html lang="pt-br">
3 <head>
4   <title>Formulário - Exemplo Campo Select</title>
5   <meta charset="utf-8">
6 </head>
7 <body>
8   <form>
9     <select name="" size="2" multiple="">
10      <option>
11        texto da opção1
12      </option>
13      <option>
14        texto da opção2
15      </option>
16      <option>
17        texto da opção3
18      </option>
19    </select>
20  </form>
21 </body>
22 </html>
```

Resultado no Navegador



17.3.8 Área de Texto

Permite a entrada de um texto onde é possível definir o tamanho através das linhas (**rows**) e colunas (**cols**).

Declaração: `<textarea cols="" rows="">`
`<!--texto inicial, se desejar-->`
`</textarea>`

Parâmetros:

- **cols**: número de caracteres por linha (colunas).
- **rows**: número de linhas do campo.

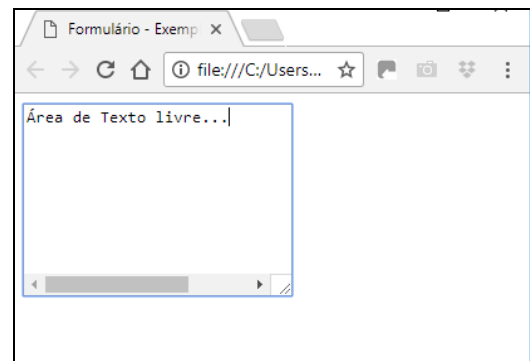
Observação: *não existe o parâmetro **value** para área de texto*, o texto inicial deve ser definido entre as tags `<textarea>` `</textarea>`.

Exemplo campo textarea

```

1 <!DOCTYPE html>
2 <html lang="pt-br">
3 <head>
4     <title>Formulário - Exemplo Campo Text Area</title>
5     <meta charset="utf-8">
6 </head>
7 <body>
8     <form>
9         <textarea cols="20" rows="10">
10
11         </textarea>
12     </form>
13 </body>
14 </html>
    
```

Resultado no Navegador



17.4 Trabalhando com os dados enviados pelo Form

Como visto anteriormente, os **forms** submetem os dados à um script ou página especificada no parâmetro **action**. Este script ou página pode ser ou não o mesmo onde o **form** está contido. No script destino, são criadas automaticamente variáveis com os mesmos nomes dos elementos do **form**, contendo os valores submetidos.

Por exemplo:

Em uma página qualquer, tem-se o seguinte código:

```
<form action="teste.php" method="post">
  Digite seu nome <input type="text" name="nome"> <br>
  <input type="submit" name="enviar" value="Clique aqui">
</form>
```

Que resulta no seguinte formulário na página web:

Digite seu nome

Quando este **form** for enviado (clicando no botão **Clique aqui**), na página **teste.php** será criada uma *variável* com o nome **\$nome** contendo o texto que foi digitado no campo.

17.5 Colocando Formulários em Prática

Antes de continuar, para poder praticar os formulários HTML, será necessário **instalar um pacote chamado WampServer**. Nos computadores dos laboratórios esse pacote já está instalado, mas quem estiver usando o seu notebook, deverá instalar. Veja o passo-a-passo no Capítulo 18 desta apostila (**Página 55**).

Após instalado o Wamp, vamos criar duas páginas (dois arquivos) no nosso servidor de teste (**dentro da pasta www do Wamp**):

- **tutorial.html**: será o formulário com alguns campos para o usuário preencher.
- **resultado.php**: arquivo onde serão exibidas as respostas do formulário de uma maneira amigável no *browser*.

No primeiro arquivo, **tutorial.html**, você deve criar uma estrutura HTML normal, simples, e dentro do **body** criar um formulário que submete (envia) os dados do formulário para a página **resultado.php**, onde essas informações serão trabalhadas e exibidas na tela. Siga o passo-a-passo da seguinte maneira:

Passo1) Criar o arquivo **tutorial.html** com o formulário, que vai conter um **campo texto** para entrada do nome do usuário, um **checkbox** perguntando se o usuário leu o tutorial (marcar a opção) e um campo **select** para entrada da nota dada ao tutorial. Salvar o arquivo dentro da pasta **www** do Wamp, com o nome **tutorial.html**, conforme código abaixo:

```

1  <html>
2  <head>
3  <meta charset="utf-8">
4  <title>Formulário HTML e PHP</title>
5  </head>
6  <body>
7  <h3 align=center>Meu Primeiro Formulário HTML</h3>
8  <form action="resultado.php" method="post">
9  Nome <input type="text" name="nome"><br>
10 Eu li o tutorial todinho <input type="checkbox" name="leu"><br>
11 Qual nota você dá para o tutorial?
12 <select name="nota">
13 <option value="0">Nota 0</option>
14 <option value="1">Nota 1</option>
15 <option value="2">Nota 2</option>
16 <option value="3">Nota 3</option>
17 </select>
18 <input type="submit" value="Enviar">
19 </form>
20 </body>
21 </html>
    
```

Passo2) Ao abrir o arquivo **tutorial.html** no *browser*, deve aparecer da seguinte maneira:

Meu Primeiro Formulário HTML

Nome

Eu li o tutorial todinho

Qual nota você dá para o tutorial?

Passo3) Agora você deve criar o arquivo **resultado.php**, que receberá os dados enviados pelo formulário quando o usuário clicar no botão Enviar. Veja os comandos seguir.

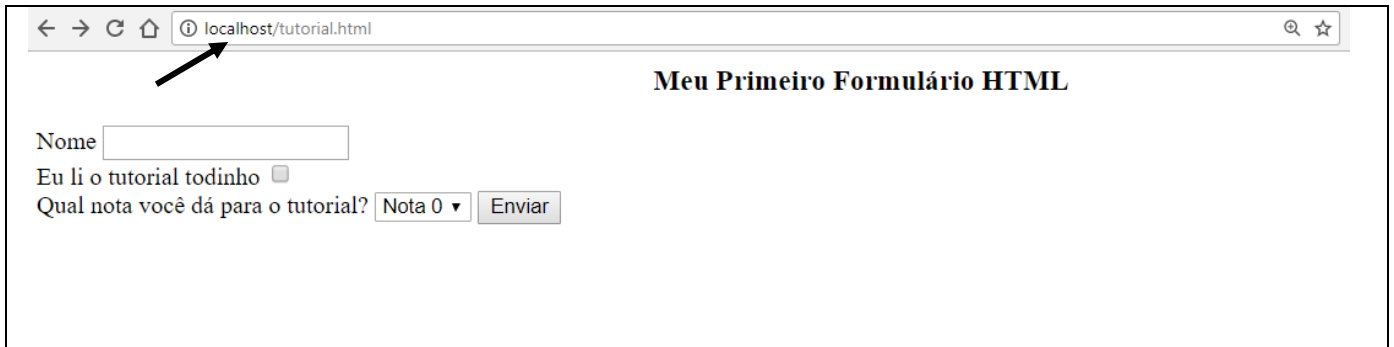
```

resultado.php
<html>
  <head>
    <meta charset="utf-8">
    <title>Resultado</title>
  </head>
  <body>
    <h3 align=center>Dados Preenchidos no Formulário</h3>
    Usuário: <?php echo $_POST['nome']; ?><br>
    Leu o tutorial:
    <?php
      if (isset($_POST['leu'])) /*isset testa se a variável existe*/
      {
        echo "Sim, leu!";
      }
      else
      {
        echo "Não leu.";
      }
    ?><br>
    Nota: <?php echo $_POST['nota']; ?>
  </body>
</html>
    
```

Passo4) Salve o arquivo **resultado.php** também dentro da pasta www do Wamp.

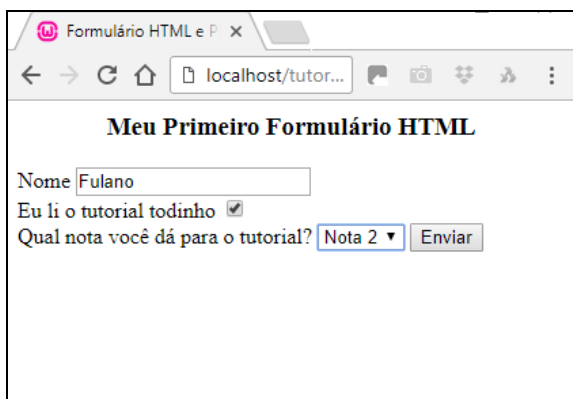
Passo5) Para testar o formulário é necessário ter instalado o **Wamp** e ter salvo os dois arquivos (**tutorial.html** e **resultado.php**) **dentro da pasta www do Wamp**. Também verifique se os serviços do Wamp estão **online** (o ícone deve estar verde).

Passo6) Você deve abrir o arquivo **tutorial.html** em um navegador da seguinte forma: **localhost/tutorial.html**, pois só assim os comandos PHP irão funcionar. No Chrome, aparecerá da seguinte maneira:

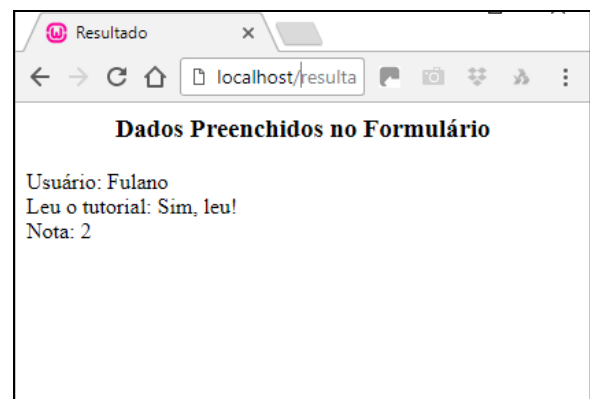


Passo7) Preencha o formulário, faça uma oração e clique no botão **Enviar** para testar se o seu formulário está funcionando! 😊

Exemplo Formulário Prático



Resultado no Navegador



Conclusão: este exemplo mostrou como utilizar **forms** HTML, explicando detalhadamente seus elementos e respectivos parâmetros e como capturar e utilizar os dados enviados pelo **form**, usando a linguagem PHP e o pacote WampServer.

18 Pacote WampServer

Para colocar em prática os formulários HTML, é necessário instalar o pacote WampServer. Nas seções a seguir será apresentado um passo-a-passo para instalar no Windows.

18.1 O que é WampServer

WampServer é um ambiente de desenvolvimento web para Windows que permite criar aplicações com Apache, PHP e Banco de Dados MySQL. Possui o phpMyAdmin que permite gerenciar facilmente os bancos de dados.

Wamp significa:

- **W**indows: Sistema Operacional (SO)
- **A**pache: Servidor Web
- **M**ySQL: Sistema Gerenciador de Banco de Dados (SGBD)
- **P**HP – Perl – Python: Linguagens de Programação para desenvolvimento web.

Assim como o pacote Wamp, que é para Windows, tem o Lamp para Linux e o Mamp para Mac OS X. WampServer é um pacote que instala automaticamente tudo que é necessário para desenvolver aplicações web e é muito intuitivo de usar.

18.2 Instalando WampServer

O site oficial do Wamp é www.wampserver.com. WampServer é um projeto de código aberto, livre para usar (licença GPL). E é gratuito! Na página do WampServer você escolhe baixar a versão para 32 ou 64 bits (Windows: Iniciar, Configurações, Sistema, Sobre), de acordo com seu Sistema Operacional, conforme imagem a seguir:

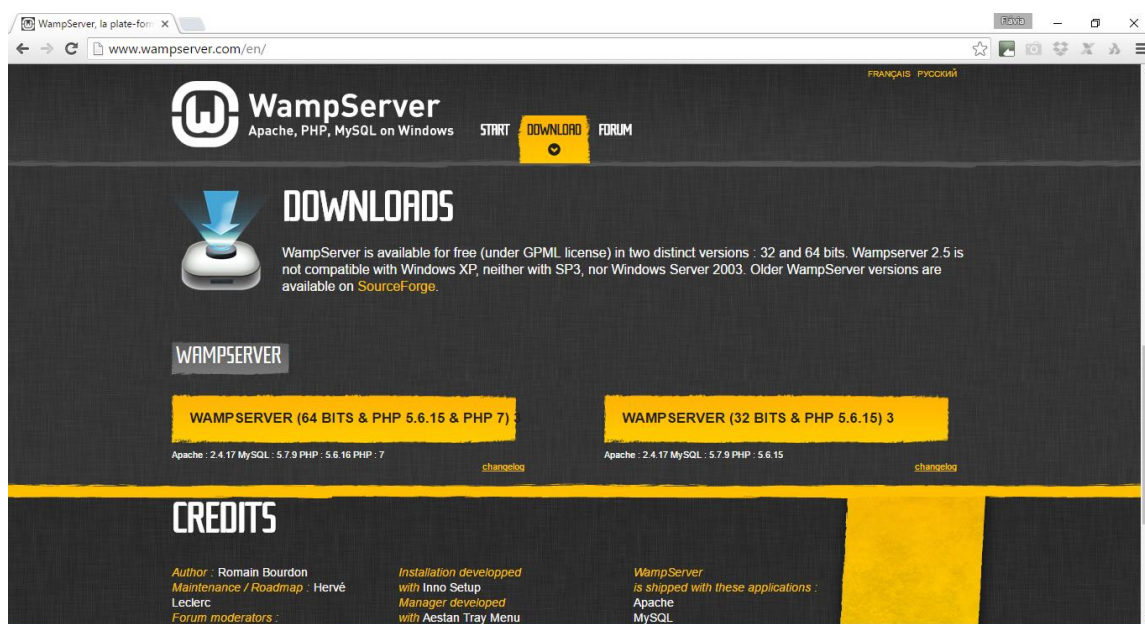


Figura 15: Site Oficial WampServer

Após, fazer o *download*, basta dar um duplo clique no arquivo baixado e seguir as instruções. Tudo é automático. O pacote WampServer sempre vem com as últimas versões do Apache, MySQL e PHP. Uma vez instalado, é possível atualizar as versões apenas clicando com o mouse. Cada versão do Apache, MySQL e PHP tem suas próprias configurações e seus próprios arquivos.

18.3 Usando WampServer

- a) O diretório "**www**" será criado automaticamente (normalmente **c : \wamp\www**)
- b) Caso não tenha sido criado automaticamente, crie um subdiretório no "www" e coloque seus arquivos PHP e HTML dentro dele.
- c) Clique no link "**localhost**" no menu WampServer ou acesse pelo navegador: **http://localhost**

Perceba que, quando o WampServer estiver aberto, aparecerá no canto inferior direito, na Barra de Tarefas (do Windows) um ícone, conforme imagem abaixo (canto direito inferior):



Figura 16: Ícone do WampServer na Barra de Tarefas do Windows

Clicando no **ícone do Wamp** (se estiver **verde**, como na Figura 16 acima, os serviços estão rodando), abrirá o seguinte menu, apresentado da Figura 17 a seguir.

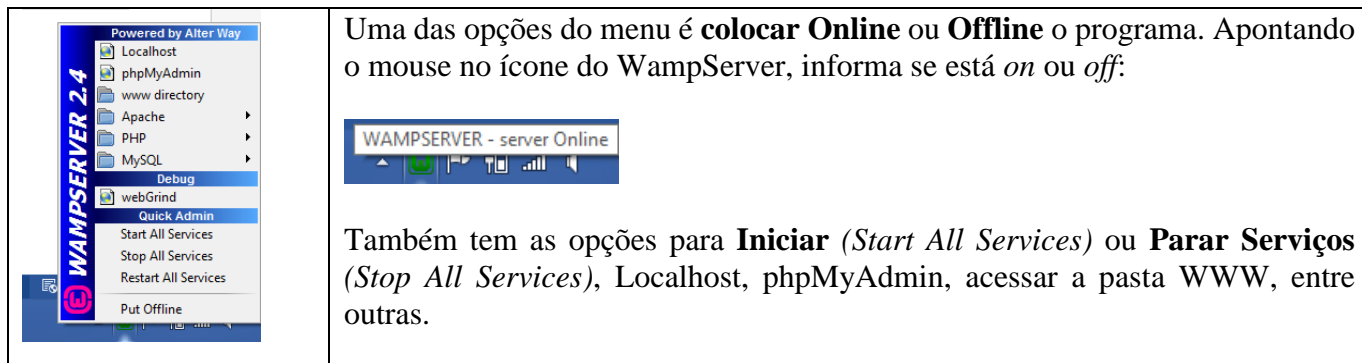


Figura 17: Menu do WampServer

Agora que você já tem o pacote Wamp instalado no seu computador, pode retornar para o Capítulo 17.5 (Página 53), para fazer os exemplos práticos de Formulários em HTML, usando a linguagem PHP.

19 Introdução às Regras CSS

CSS é a sigla em inglês para *Cascading Style Sheets*, que traduzindo significa **Folhas de Estilo em Cascata**. Consiste num conjunto de regras **responsáveis pela formatação** de um documento web, como organizar a página, posicionar e expor o texto e, dependendo de onde é aplicado, como organizar uma coleção de documentos. **Visa remover a formatação dos documentos HTML, separando o conteúdo da formatação**.

CSS formata a informação entregue pelo HTML. Essa informação pode ser qualquer coisa: imagem, texto, vídeo, áudio ou qualquer outro elemento criado. **Atenção: CSS formata a informação!**

HTML5 trouxe poucas novidades para os desenvolvedores *client-side*. Basicamente foram criadas novas *tags*, o significado de algumas foi modificado e outras *tags* foram descontinuadas. As novidades interessantes mesmo ficaram para o pessoal que conhece JavaScript. As APIs que o HTML5 disponibilizou são, sem dúvida, uma das *features* mais aguardadas por todos estes desenvolvedores. Diferentemente do CSS3 que trouxe mudanças drásticas para a manipulação visual dos elementos do HTML. **Resumindo:** CSS é uma linguagem de estilo utilizada para definir a apresentação de documentos na web!

Por que utilizar CSS? A HTML foi criada para ser uma linguagem exclusivamente de marcação e estruturação de conteúdos na web. Isso significa que não cabe à HTML fornecer informações sobre a apresentação dos elementos da página web. Por exemplo: cores de fontes, tamanhos de textos, posicionamentos e todo aspecto visual de um documento web **não** são funções da HTML. Cabem às CSS todas as funções de apresentação de um documento, e essa é a sua finalidade maior. Daí a já consagrada frase que resume a dobradinha HTML + CSS: **“HTML para Estruturar os Conteúdos e CSS para Apresentá-los!”**, conforme imagem abaixo:

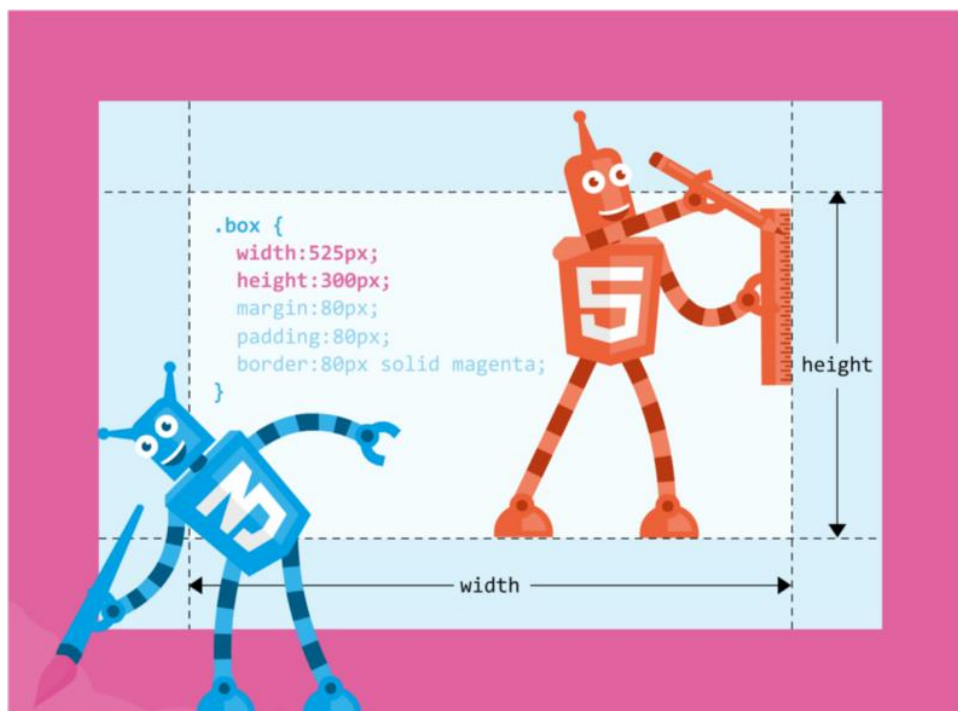


Figura 18: HTML para Estruturar os Conteúdos e CSS para Apresentá-los

Fonte: <https://medium.com/@AmJustSam/whats-the-difference-between-background-clip-vs-background-origin-b9a43b4a973>

19.1 Algumas considerações sobre CSS

- Os estilos definem para o browser como devem ser exibidos os elementos HTML.
- Os estilos são geralmente definidos em folhas de estilos.
- O CSS foi implementado na versão do HTML 4.0 para resolver o problema de separação entre conteúdo e formatação.
- Os estilos quando armazenados em folhas de estilo externas ao documento HTML (arquivo.css) e compartilhados entre páginas web do mesmo site, poupam muito trabalho e simplificam enormemente a manutenção do site.

19.2 Alguns benefícios, que justificam o uso de CSS

- Produtividade aumentada.
- Facilidade em criar sites com identidade visual unificada e coerente.
- Facilidade em fazer alterações em todo site, pois basta alterar um arquivo CSS em vez de ter que mudar várias páginas HTML.
- Arquivos mais leves = download mais rápido = experiência do usuário melhorada.
- Menos código na página, tornando mais fácil codificar.
- Sites mais acessíveis para uma ampla variedade de aparelhos.
- Mais controle sobre o código: interpretação do código na ordem correta para os “leitores de tela” (acessibilidade).
- Permitir que visitantes alterem suas preferências definindo estilos dinamicamente.
- Disponibiliza versão para impressão sem duplicação de conteúdo, somente alternando o CSS.
- Permite formatar elementos HTML, como formulários e barras de rolagem, impossível via atributos HTML.
- Controle do layout de vários documentos a partir de uma única folha de estilos (apenas um arquivo .css).

Uma das metas ao se usar os conceitos de *Web Standards* (padrões da internet) é remover toda a apresentação do código HTML, deixando-o limpo e semanticamente correto. Enquanto HTML define a estrutura, o CSS fica responsável pela formatação visual e posicionamento dos elementos dentro da página web.

Os estilos podem e devem, na maioria dos casos, ser definidos em um arquivo separado do HTML, com extensão .css e compartilhado entre todos ou com um grupo de documentos (páginas web) relacionados ao mesmo site.

Outra grande vantagem na separação do conteúdo da formatação, é a possibilidade de tornar disponível o mesmo conteúdo para múltiplos *devices* (aparelhos) sem necessidade de duplicar conteúdos, alterando somente a formatação, ou seja, o arquivo `.css`. Isto inclusive pode ser feito de forma dinâmica ou utilizando JavaScript, por exemplo.

19.3 Sintaxe do CSS

A sintaxe dos estilos é chamada de **Regra CSS** e é uma sintaxe de formatação simples, definindo o elemento a ser formatado (chamado de **seletor**), a **propriedade** desse elemento e o **valor** que essa propriedade irá assumir. O seletor pode ser uma *tag* ou um grupo de *tags* HTML, uma Classe um ou Id, de acordo com o que deseja-se estilizar (mais detalhes em capítulos adiante). Veja o exemplo a seguir:

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

Uma regra CSS escrita conforme a sintaxe mostrada, poderá conter várias declarações separadas por ponto e vírgula, conforme exemplo abaixo:

```
p{
    color: red;
    text-align: center;
    font-size: 14px;
}
```

19.3.1 Agrupar Seletores

Na prática, ocorre com frequência a necessidade de estilizar vários seletores com as mesmas declarações de estilos. Nesses casos, a sintaxe CSS prevê que os seletores podem ser agrupados. Se tiver elementos com as mesmas definições de estilos, como a seguir:

```
h1 {
    text-align: center;
    color: red;
}

h2 {
    text-align: center;
    color: red;
}

p {
    text-align: center;
    color: red;
}
```

Conforme o exemplo anterior, será melhor agrupar os seletores, para minimizar o código, separando cada seletor com uma vírgula. No exemplo abaixo, agruparemos os seletores do código acima:

```
h1, h2, p {
    text-align: center;
    color: red;
}
```

Atenção: Não confunda seletores encadeados e seletores agrupados! Veja as diferenças a seguir.

Seletores encadeados e seletores agrupados são a base do CSS. Você os aprende por osmose durante o dia a dia ☺ Para você lembrar o que são seletores encadeados e agrupados segue um exemplo abaixo.

Exemplo de seletor encadeado:

```
div p strong a {
    color: red;
}
```

Este seletor formata o link **a**, que está dentro de um **strong**, que está dentro de **p** e que por sua vez está dentro de um **div**.

Exemplo de seletor agrupado:

```
strong, em, span {
    color: red;
}
```

Você agrupa elementos separados por vírgula para que herdem a mesma formatação.

19.4 Site que altera seu design com CSS

Para obter uma demonstração do que pode ser conseguido visualmente através de um design baseado em CSS, acesse o site Zen Garden (<http://www.csszengarden.com/>), onde diferentes desenvolvedores criaram para o mesmo conteúdo, ou seja, para o mesmo HTML, diferentes arquivos CSS, resultando em documentos com design 100% diferentes entre si, mas sempre apresentando o mesmo conteúdo.

19.5 Formas de usar CSS

Existem três tipos de Folhas de Estilo, ou seja, três maneiras diferentes de usar CSS nos arquivos HTML são elas: **Inline**, **Interna** e **Externa**. Veja a seguir cada uma delas detalhadamente.

19.5.1 Inline

Também chamada **Em Linha**, é quando a informação do estilo é aplicada direto em uma tag específica dentro do documento HTML. Este tipo de estilo *raramente é utilizado*, pois esta técnica seria necessária apenas quando há uma única ocorrência de um estilo específico em todo site.

Sintaxe:

```
<p style="font-family: sans-serif;">CSS Inline</p>
```

19.5.2 Interna

Também chamada de **Incorporada**, *também não é muito utilizada*, pois seria quando uma página do site tem a sua própria folha de estilo, ou seja, coloca-se as regras CSS dentro do arquivo HTML usando a **tag <style>** que deve estar dentro da **tag <head>**. É útil apenas quando deseja-se uma formatação diferenciada em uma única página de todo site.

Sintaxe:

```
<head>
  <title>CSS Interno</title>
  <style type="text/css">
    body{
      font-family: verdana, arial, sans-serif;
    }
    h1{
      font-size: 120%;
    }
  </style>
</head>
```

19.5.3 Externa

Este tipo de folha de estilo é **o mais usado**, pois com ele é possível vincular várias páginas web aos estilos que estiverem presentes no arquivo, ou seja, é criado um **arquivo .css separado do arquivo .html**, contendo todas as regras CSS desejadas. Nos arquivos HTML que deseja-se usar tais regras, simplesmente se “linka” o arquivo .css, ou seja, se chama, se importa o CSS dentro do HTML.

Sintaxe:

Arquivo HTML	Arquivo CSS
<pre> 1 <!DOCTYPE html> 2 <html> 3 <head> 4 <link rel="stylesheet" type="text/css" href="cssexterno.css"> 5 <title>Exemplo CSS Externo</title> 6 </head> 7 <body> 8 <h1>Exemplo CSS Externo</h1> 9 <p>O arquivo CSS Externo é o mais usado.</p> 10 <p>Tem-se um arquivo separado do HTML com a extensão .css</p> 11 <p>E temos que importar o arquivo .css dentro do arquivo .html</p> 12 </body> 13 </html> </pre>	<pre> 1 h1 { 2 text-align: center; 3 border: 5px; 4 border-style: solid; 5 border-color: red; 6 } 7 p { 8 color: blue; 9 } </pre>

19.6 Exemplos Práticos

Nos próximos capítulos serão apresentados exemplos para serem praticados usando as três formas de Folhas de Estilos: Inline, Interna e Externa.

19.6.1 Exemplo1 usando CSS Inline

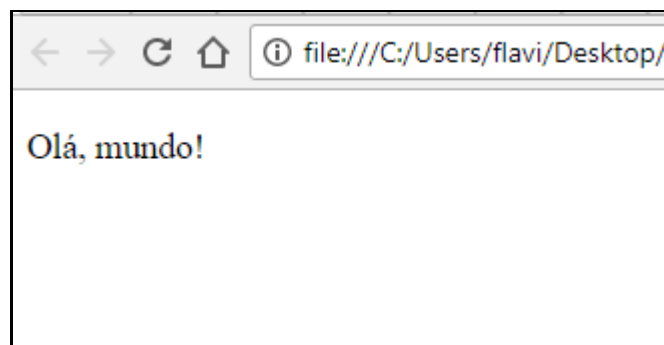
Vamos fazer um exercício prático com o tradicional exemplo "Olá, mundo!", usando **CSS Inline**. Siga os passos a seguir.

Passo1) Abra o Editor HTML que pretende usar para desenvolver suas páginas (pode ser o Sublime) e digite o seguinte código:

```

Exemplo1-CSSInline.html
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <title>Exemplo1-CSS Inline</title>
5 </head>
6 <body>
7   <p>Olá, mundo!</p>
8 </body>
9 </html>
    
```

Passo2) Salve o arquivo com o nome **Exemplo1-CSSInline.html** e abra no navegador. Você verá que não há nada de especial sobre essa linha, por enquanto.

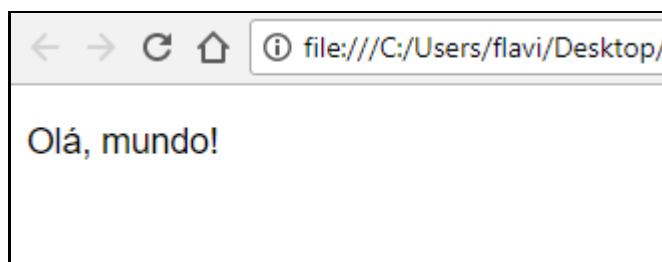


Passo3) Para mudar o estilo da letra para *sans-serif* (sem serifa), vamos adicionar o seguinte CSS diretamente no código HTML, na tag `<p>`, pois estamos usando CSS Inline:

```

1  <!DOCTYPE html>
2  <html>
3  <head>
4  |   <title>Exemplo1-CSS Inline</title>
5  </head>
6  <body>
7  |   <p style="font-family: sans-serif;">Olá, mundo!</p>
8  </body>
9  </html>
    
```

Passo4) Salve o arquivo novamente (Ctrl+S) e atualize sua página no navegador (F5) para ver o novo estilo de letra adicionado ao parágrafo.



19.6.2 Exemplo2 usando CSS Incorporado

Agora, neste segundo exemplo, vamos fazer um exercício prático usando **CSS Incorporado**, ou seja, dentro do arquivo HTML, mas no **<head>** da página. Siga os passos a seguir.

Passo1) Crie um novo arquivo e salve com o nome **Exemplo2-CSSIncorporado.html**, contendo o seguinte código HTML:

```

1 <!DOCTYPE html>
2 <html>
3 <head>
4   <title>Exemplo2-CSS Incorporado</title>
5 </head>
6 <body>
7   <h1>Exemplo CSS Incorporado</h1>
8   <p>Este é um simples parágrafo com um <a href="http://www.google.com">link</a>.</p>
9 </body>
10 </html>
    
```

Passo2) Salve e abra esse arquivo no navegador:

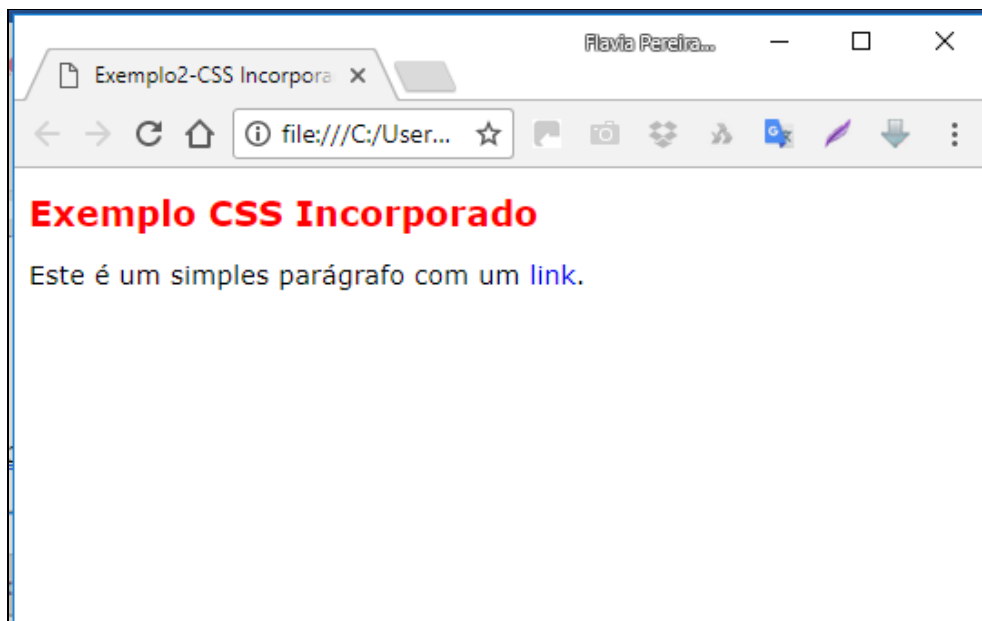


Passo3) Acrescente o seguinte código CSS dentro da tag `<head>` usando a tag `<style>`:

```

1  <!DOCTYPE html>
2  <html>
3  <head>
4      <title>Exemplo2-CSS Incorporado</title>
5      <style type="text/css">
6          body {
7              font-family: verdana, arial, sans-serif;
8          }
9          h1 {
10             font-size: 120%;
11             color: red;
12         }
13         a {
14             text-decoration: none;
15         }
16
17         p {
18             font-size: 90%;
19         }
20
21     </style>
22 </head>
23 <body>
24     <h1>Exemplo CSS Incorporado</h1>
25     <p>Este é um simples parágrafo com um <a href="http://www.google.com">link</a>.</p>
26 </body>
27 </html>
    
```

Passo4) Salve e abra o arquivo no navegador:



19.6.3 Exemplo3 usando CSS Externo

Neste terceiro exemplo, vamos fazer um exercício prático usando **CSS Externo**, que é o método mais utilizado. Siga os passos a seguir.

Passo1) Crie um novo arquivo e salve com o nome **Exemplo3-CSSExterno.html**, contendo o seguinte código HTML:

```

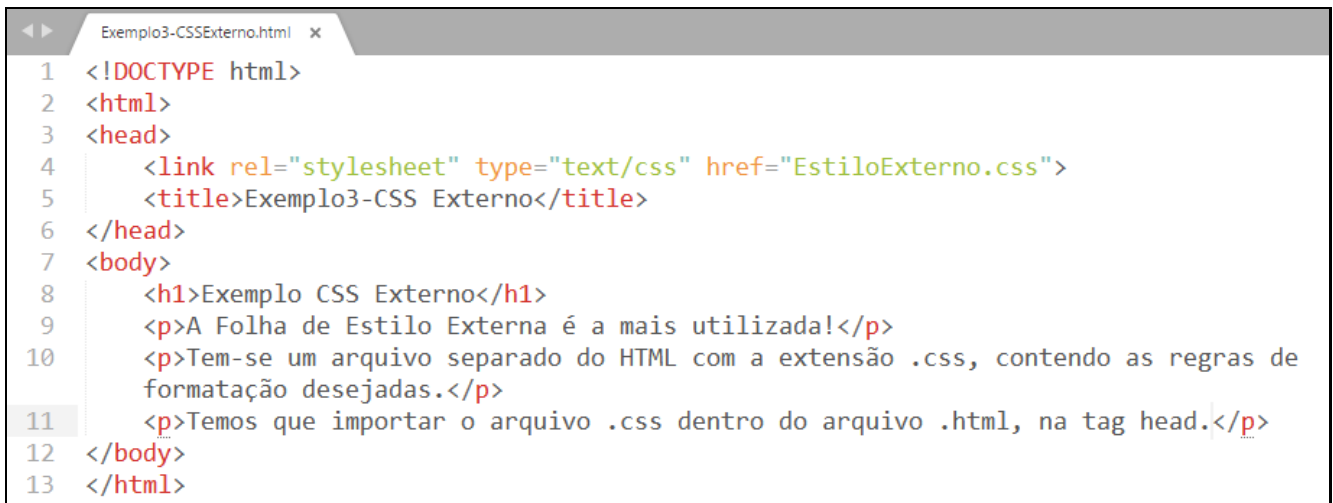
1  <!DOCTYPE html>
2  <html>
3  <head>
4      <title>Exemplo3-CSS Externo</title>
5  </head>
6  <body>
7      <h1>Exemplo CSS Externo</h1>
8      <p>A Folha de Estilo Externa é a mais utilizada!</p>
9      <p>Tem-se um arquivo separado do HTML com a extensão .css, contendo as regras de
10     formatação desejadas.</p>
11     <p>Temos que importar o arquivo .css dentro do arquivo .html, na tag head.</p>
12 </body>
13 </html>
    
```

Passo2) Salve e abra esse arquivo no navegador:



Passo3) Acrescente a seguinte linha **dentro da tag <head>** no arquivo .html, para importar o arquivo externo chamado **EstiloExterno.css**, o qual vai conter as regras CSS que serão aplicadas ao HTML:

```
<link rel="stylesheet" type="text/css" href="EstiloExterno.css">
```



```

1  <!DOCTYPE html>
2  <html>
3  <head>
4      <link rel="stylesheet" type="text/css" href="EstiloExterno.css">
5      <title>Exemplo3-CSS Externo</title>
6  </head>
7  <body>
8      <h1>Exemplo CSS Externo</h1>
9      <p>A Folha de Estilo Externa é a mais utilizada!</p>
10     <p>Tem-se um arquivo separado do HTML com a extensão .css, contendo as regras de
11     formatação desejadas.</p>
12     <p>Temos que importar o arquivo .css dentro do arquivo .html, na tag head.</p>
13 </body>
14 </html>
    
```

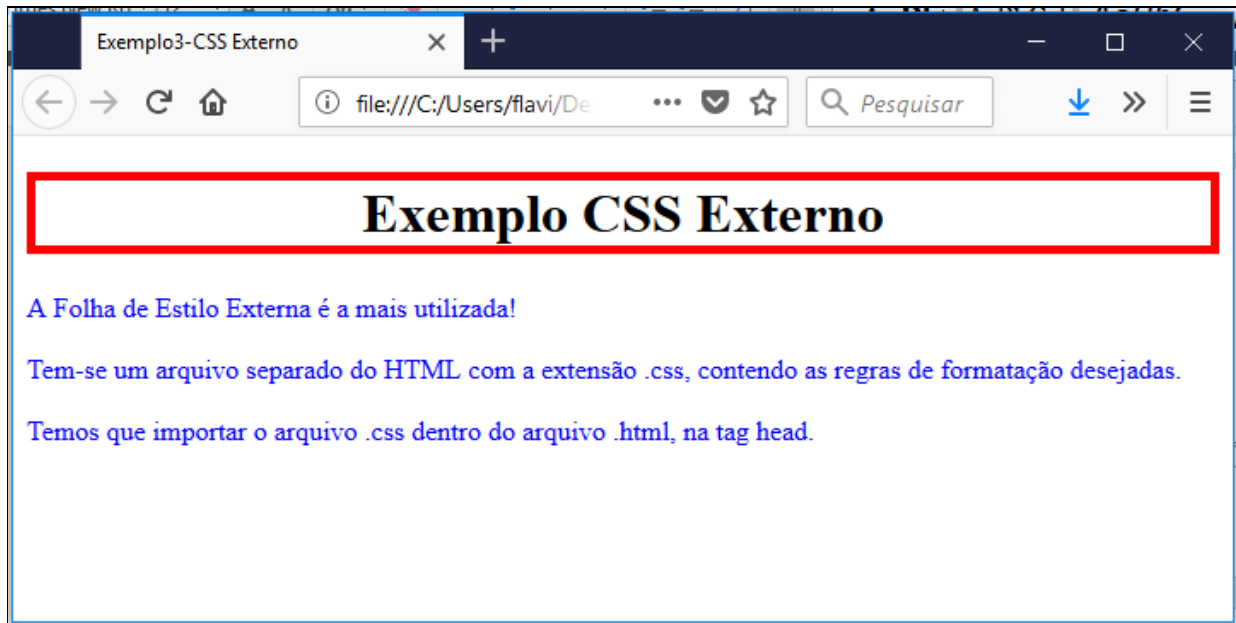
Passo4) Salve novamente o arquivo **Exemplo3-CSSExterno.html**. Se você abrir esse arquivo no navegador agora, não verá nada de diferente, pois ainda não criamos o arquivo **EstiloExterno.css** que estamos linkando dentro do HTML. Então, agora, o próximo passo, é criarmos esse arquivo .css (o nome desse arquivo é determinado por nós, ou seja, o desenvolvedor que escolhe o nome, mas a extensão deve ser .css). Crie um arquivo novo no editor e salve com esse nome: **EstiloExterno.css**. O conteúdo do arquivo CSS será o seguinte:



```

1  h1 {
2      text-align: center;
3      border: 5px;
4      border-style: solid;
5      border-color: red;
6  }
7
8  p {
9      color: blue;
10 }
11
    
```

Passo5) Atualize sua página no navegador (**F5**), para ver o efeito das regras CSS aplicadas ao arquivo HTML:



19.7 Efeito Cascata

Pode acontecer de ocorrer conflito entre regras CSS, por exemplo:

```

1  p {
2      font-family: sans-serif;
3  }
4
5  p {
6      font-family: Times;
7  }
```

A primeira regra declarada acima, determina os parágrafos com fonte sem serifa e a segunda regra diz que os parágrafos devem ser da família Times que é com serifa. Como é resolvido esse tipo de conflito nas formatações? Qual formatação será aplicada aos parágrafos?

A regra básica para resolver esse tipo de conflito é: vale a formatação que estiver mais próxima do texto! Ou seja, nesse exemplo acima, iria prevalecer a fonte Times.

O mesmo ocorre se tivermos uma regra CSS em um arquivo externo e uma outra aplicada ao mesmo elemento, mas de forma interna, por exemplo. Então, nesse caso, a regra que irá prevalecer será a Interna. Isso se chama **efeito cascata**, que nada mais é, do que o estabelecimento de uma **prioridade** para aplicação da regra de estilo a um elemento.

Para determinar a prioridade, são considerados diversos fatores, entre eles, o tipo de folha de estilo, o local físico da folha de estilo no seu todo, o local físico da regra de estilo na folha de estilo e a especificidade da regra de estilo (pesquise mais sobre especificidade das regras CSS).

19.8 Comentários em CSS

Podemos fazer comentários no código fonte CSS em uma linha ou em várias linhas, ou seja, um bloco de comentário. A sintaxe é a mesma para as duas formas: `/*` e `*/`, deve-se abrir e fechar o comentário. Tudo que estiver comentado, aparecerá em tom cinza claro, conforme os exemplos abaixo:

```
/* Este é um comentário CSS em uma linha */
```

```
/* Este é um bloco de comentário em CSS, em  
linhas diferentes contendo várias informações  
sobre um trecho de folha de estilo */
```

19.9 Seletores Classe e Id

Em capítulos anteriores já tínhamos visto um pouco sobre *class* e *id*, mas agora veremos exemplos mais detalhados. Às vezes será necessário **personalizar um grupo de tags** para isso usa-se a **Classe**. As classes são uma forma de identificar um grupo de elementos. Através delas, pode-se atribuir formatação a vários elementos de uma só vez. Outras vezes será necessário **personalizar apenas um elemento**, para isso utiliza-se o **Identificador**.

Resumindo: a principal diferença entre o uso de *class* e *id* é que quando se deseja **agrupar elementos**, usa-se *class* e quando se deseja **identificar um único elemento**, usa-se *id*.

19.9.1 Quando e Como usar Id

O seletor de **Identificação** usa o atributo **id** de um elemento HTML para selecionar **um elemento específico**. O **id** de um elemento deve ser exclusivo dentro de uma página, então o seletor de identificação é usado para selecionar um único elemento! Para selecionar um elemento com um **id** específico, usa-se o caractere *hash* (**#**), seguido do **Nome do Id** (que você determina) do elemento.

O que há de especial no atributo **id** é que não poderá existir dois ou mais elementos com o mesmo **id**, ou seja em um documento apenas um e somente um elemento poderá ter um determinado **id**. Cada **id** é único. Para casos em que haja necessidade de mais de um elemento com o mesmo identificador usam-se o atributo **class**. A seguir um exemplo de possível uso de **id**:

Não esqueça: **id**'s são únicos!

- Cada elemento pode ter **apenas um id**
- Cada página pode ter apenas **um elemento** com aquele **id**

O que pode acontecer se usarmos se não usarmos da forma correta? *Seu código não vai passar pelo validador se você usar o mesmo ID em mais de um elemento*. Adiante, veremos mais razões pelas quais um ID deve ser único.

Exemplo: a regra de estilo abaixo será aplicada ao elemento HTML com **id = "para1"**:

```
#para1 {
    text-align: center;
    color: red;
}
```

Nota: O nome de um **id** não pode iniciar com número!

Outro Exemplo:

Arquivo HTML	Arquivo CSS
<pre><h1 id="titVermelho">Titulo Vermelho</h1></pre>	<pre>#titVermelho { color: red; }</pre>

19.9.2 Exemplo usando Class

Para agrupar elementos deve-se utilizar uma **Classe**, que é definida dentro do CSS e no arquivo HTML. Dentro da *tag* identifica-se a que classe, determinado elemento faz parte, utilizando o atributo **class**. Para selecionar elementos com uma classe específica, escreva um **ponto (.)**, seguido do **Nome da Classe** (que o programador determina).

Exemplo:

Arquivo HTML	Arquivo CSS
<pre><p class="textoSite">Texto do site</p></pre>	<pre>.textoSite { font-size: 15px; }</pre>

Não esqueça: Classes **não** são únicas!

- É possível usar a **mesma class** para vários elementos.
- É possível usar **várias classes para um mesmo elemento**.

Qualquer informação de estilo que precise ser aplicada a múltiplos elementos em uma página deve ser feita com uma **class**.

No exemplo abaixo, todos os elementos HTML com **class = "center"** serão vermelhos e alinhados no centro:

```
.center {
    text-align: center;
    color: red;
}
```

Também é possível determinar que apenas elementos HTML específicos serão afetados por uma classe. No exemplo abaixo, apenas os elementos **<p>** com **class = "center"** serão alinhados no centro:

```
p.center {
    text-align: center;
    color: red;
}
```

Os elementos HTML também podem se referir a mais de uma classe. No exemplo abaixo, o elemento `<p>` será formatado de acordo com `class = "center"` e com `class = "large"`:

```
<p class="center large">This paragraph refers to two classes.</p>
```

Nota: O nome de uma `class` não pode iniciar com número!

19.10 Box Model

Todos os elementos HTML podem ser considerados como **caixas**. Em CSS, o termo "modelo de caixa" (*box model*) é usado quando se fala sobre design e layout. O modelo de caixa CSS é essencialmente **uma caixa que envolve todos os elementos HTML**. Consiste em: margens (*margin*), bordas (*border*), preenchimento (*padding*) e o conteúdo realmente (*content*).

Sempre quando vamos estilizar algum elemento via CSS, é comum, e precisamos estar cientes, que alguma alteração que iremos fazer possa impactar em outros elementos. Isso fica fácil de compreender quando entendemos o conceito de **box model**.

Basicamente, a ideia do **box model** é composta por essas quatro partes citadas acima (na ordem, de dentro para fora):

- conteúdo (*content*)
- preenchimento (*padding*)
- bordas (*border*)
- margens (*margin*)

Resumindo, pode-se dizer que o **box model** trata-se de como as 4 propriedades acima se relacionam para compor a dimensão dos elementos. O modelo de caixa permite adicionar uma borda em torno de elementos e definir espaço entre elementos. A imagem abaixo ilustra o modelo de caixa:

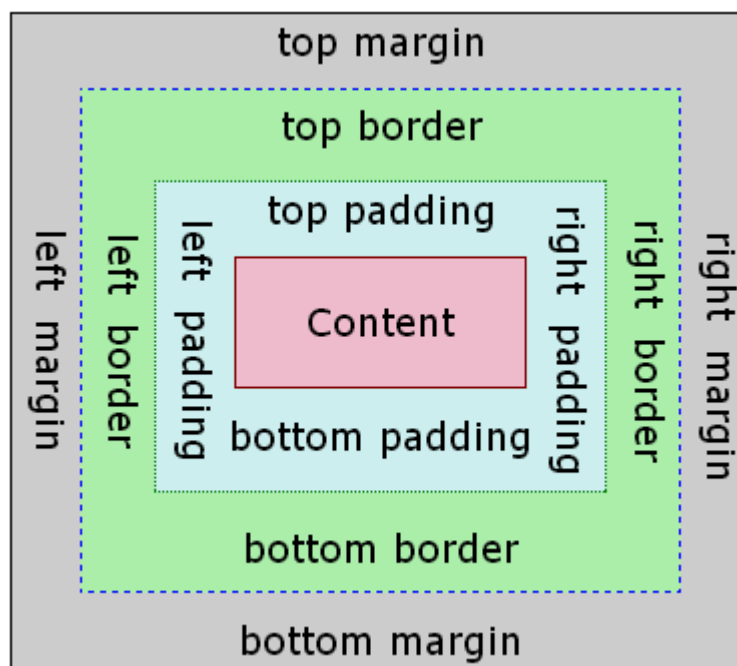


Figura 19: Box Model (Modelo de Caixa)

Explicação das diferentes partes do *Box Model*:

- **Content:** conteúdo da caixa, onde texto e imagens aparecem.
- **Padding:** área ao redor do conteúdo (espaçamento interno), ou seja, é o espaçamento entre o conteúdo e a sua borda. O preenchimento é transparente.
- **Border:** borda que circunda o preenchimento e o conteúdo.
- **Margin:** área fora da borda (as margens são externas, ou seja, distância entre uma caixa e outra). Margem é a distância da borda de um elemento até o *Box Model* de outro elemento ou até a margem do documento HTML. A propriedade *margin* adiciona um espaço transparente e não pode ser preenchido com nenhuma cor. Além disso, ela não pode receber bordas.

Abaixo, uma outra imagem para ajudar a entender o *Box Model*:



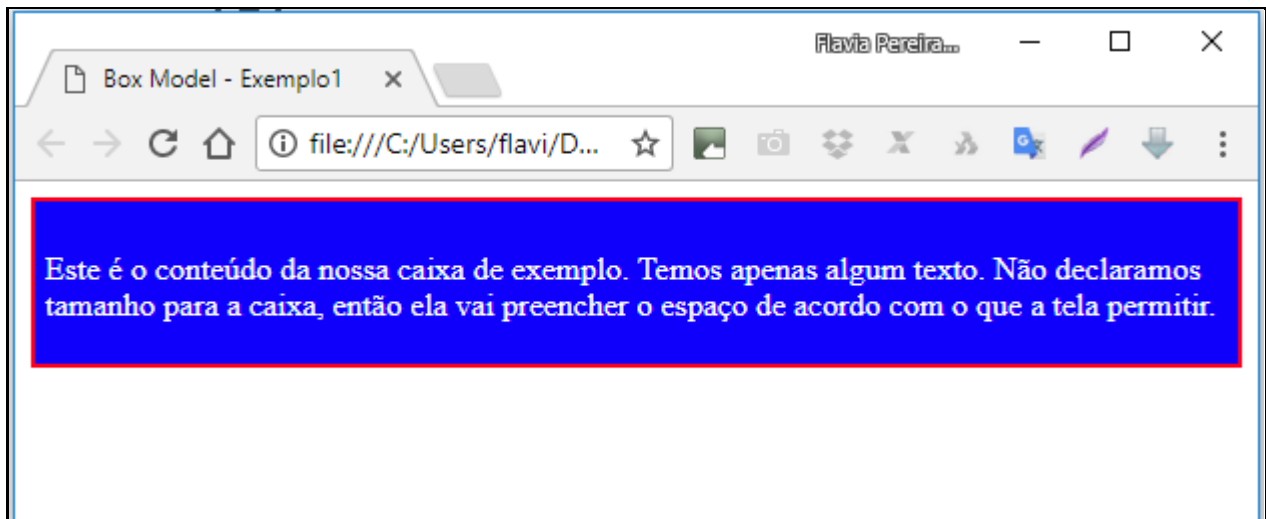
Figura 20: Outro Exemplo de *Box Model* (Modelo de Caixa)

A seguir, um código de exemplo de uma caixa com a largura não especificada:

```

1  <!DOCTYPE html>
2  <head>
3      <meta charset="utf-8">
4      <title>Box Model - Exemplo1</title>
5      <style type="text/css">
6          .box {
7              color: white;
8              background-color: blue;
9              border: 2px solid red;
10             padding: 25px 10px 20px 5px;
11         }
12     </style>
13 </head>
14 <body>
15     <div class="box">
16         Este é o conteúdo da nossa caixa de exemplo. Temos apenas algum texto. Não declaramos tamanho para a
17         caixa, então ela vai preencher o espaço de acordo com o que a tela permitir.
18     </div>
19 </body>
20 </html>
    
```

Resultado do código acima:



Esta página tem algumas margens à esquerda e à direita, mas, além disso, nossa caixa se esticará para caber na página. Em CSS, a largura da caixa é apenas a largura do conteúdo. Para calcular a quantidade real de espaço horizontal que a caixa irá ocupar, devemos adicionar os montantes de margem, borda e preenchimento para cada lado.

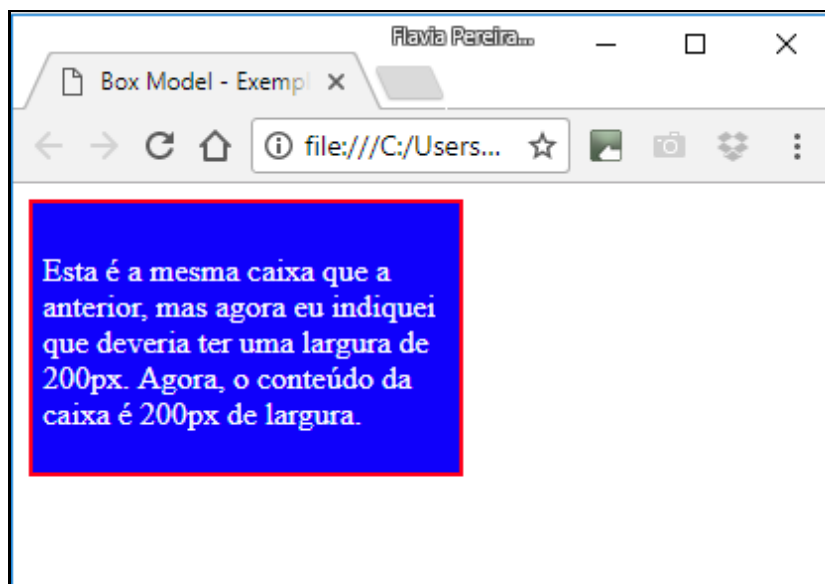
No próximo exemplo, temos a mesma que a anterior, mas agora indicando uma largura de 200px, ou seja, agora o conteúdo da caixa terá 200px de largura.

Abaixo, um código de exemplo de uma caixa com a largura 200px:

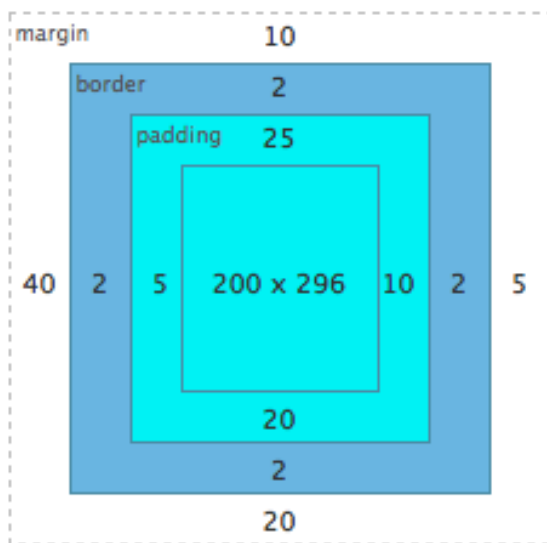
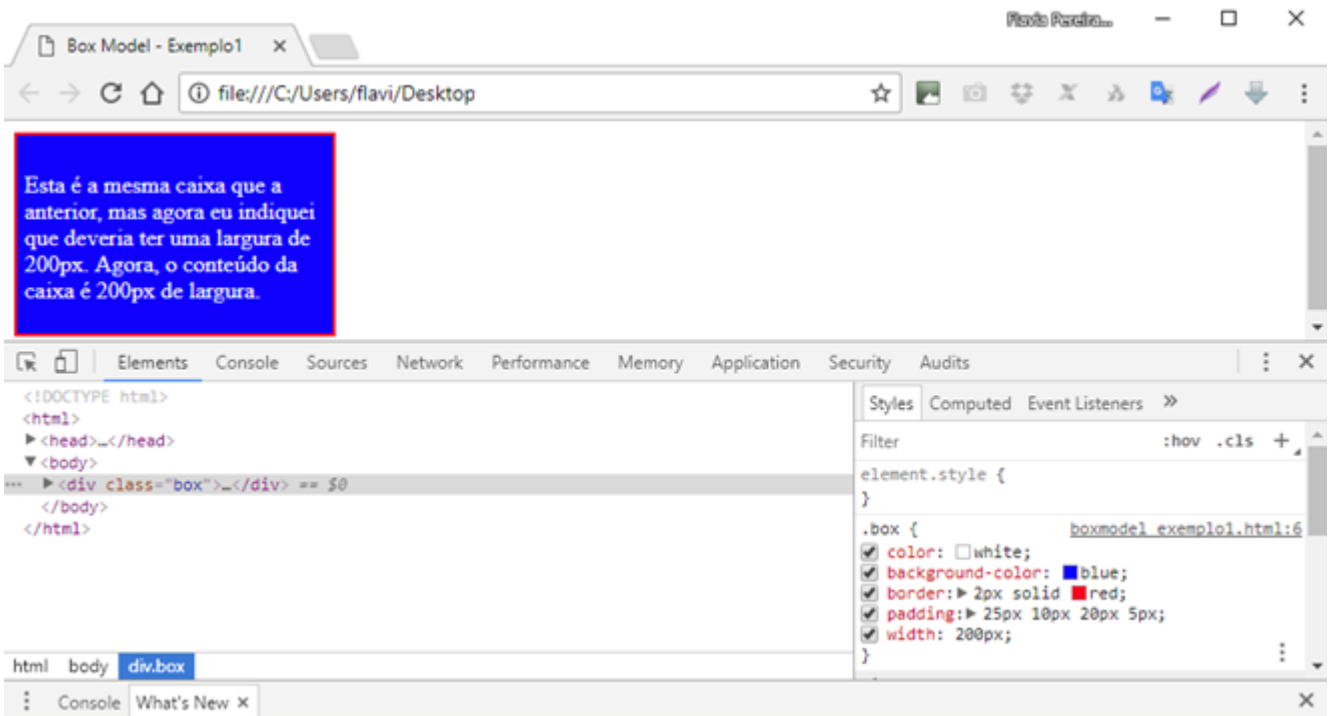
```

1 <!DOCTYPE html>
2 <head>
3     <meta charset="utf-8">
4     <title>Box Model - Exemplo2</title>
5     <style type="text/css">
6         .box {
7             color: white;
8             background-color: blue;
9             border: 2px solid red;
10            padding: 25px 10px 20px 5px;
11            width: 200px; /*largura do conteúdo*/
12        }
13    </style>
14 </head>
15 <body>
16     <div class="box">
17         Esta é a mesma caixa que a anterior, mas agora eu indiquei que deveria ter uma largura de 200px.
18         Agora, o conteúdo da caixa é 200px de largura.
19     </div>
20 </body>
21 </html>
    
```

Resultado do código acima:



O conteúdo da caixa acima tem 200px de largura. No entanto, temos margens, bordas e preenchimento à esquerda e à direita da caixa. Usando o **Inspetor da Web** é facilmente possível verificar isso. Clicando com o botão direito do mouse sobre a caixa e escolhendo a opção **Inspecionar** (ou Inspecionar Elemento), aparecem as opções conforme a imagem a seguir:



Ao adicionar toda a margem, bordas e preenchimento à esquerda e à direita da caixa juntamente com a largura especificada de 200px, obtemos 264px. A largura da caixa acima é 200px, mas a quantidade real de espaço horizontal que ocupa é 264px. Se quiséssemos que essa mesma caixa se encaixasse em um espaço de 400px, teríamos que calcular a **largura** da caixa da seguinte forma:

- 400 total pixels**
- 5 left padding
 - 10 right padding
 - 2 left border
 - 2 right border
 - 40 left margin
 - 5 right margin
-
- = 336 pixel width**

Mais exemplos:

```

1. .classe {
2.     width: 50px;
3.     height: 50px;
4.     border: 1px solid gray;
5.     padding: 10px 20px;
6. }
```

Se aplicarmos a classe acima em um elemento, ele vai ter as dimensões que setamos (50 pixels de altura e 50 pixels de largura), certo? Errado. O elemento vai ser renderizado com 72 pixels de altura e 92 pixels de largura.

Isso acontece devido ao fato das propriedades **padding** e **border** serem somadas à largura e altura já definidas, aumentando as dimensões do elemento. Isso nos leva a ver que as propriedades **width** e **height** definem as dimensões do seu conteúdo e não do elemento como um todo.

Largura - width

50 (largura definida) +
20 (padding left) +
20 (padding right) +
1 (border left) +
1 (border right) = **92 pixels de largura**

Altura - height

50 (altura definida) +
10 (padding top) +
10 (padding bottom) +
1 (border top) +
1 (border bottom) = **72 pixels de altura**

Um exemplo prático para vermos a dor de cabeça que você pode ter no seu dia a dia. Imagine que você precise ter um elemento que ocupe 100% da largura disponível. Mas também precisa que esse elemento tenha **10 pixels de padding** e uma **borda de 1 pixel**.

```

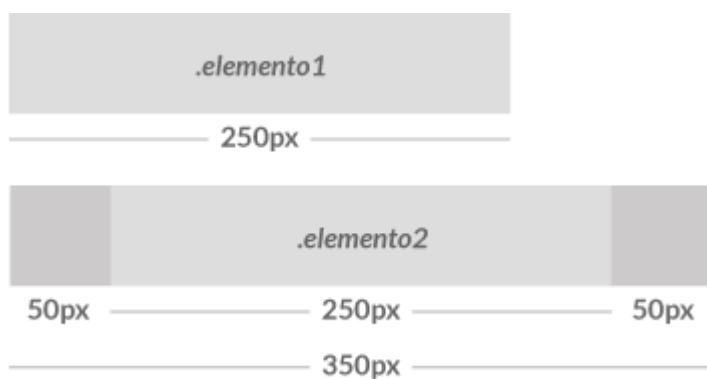
1. .dor-de-cabeca {
2.     width: 100%;
3.     padding: 10px;
4.     border: solid 1px gray;
5. }
```

Seu elemento com a **classe dor-de-cabeca** ultrapassa o limite de 100% que você tinha definido, provavelmente “quebrando” o layout. A nova largura dele é resultante da soma: 100% (largura definida) + 20 pixels (padding left e right) + 2 pixels (border left e right).

Um outro exemplo para explicar o *Box Model*, considere os elementos abaixo:

```

1. .elemento1 {
2.     background: #ddd;
3.     width: 250px;
4.     height: 50px;
5. }
6.
7. .elemento2 {
8.     background: #ddd;
9.     width: 250px;
10.    height: 50px;
11.    padding: 0 50px;
12. }
    
```



Os elementos são definidos com a mesma largura, mas como podemos ver na imagem acima, o `elemento2` é renderizado maior, devido ao **padding**.

19.10.1 Propriedades CSS para o Box Model

Veremos a seguir as regras para aplicação das propriedades CSS *margin*, *border* e *padding*. O box criado no modelo de caixa é um quadrilátero em que cada um dos lados é identificado por um termo em inglês de acordo com a posição do lado. Os lados são designados da seguinte forma:

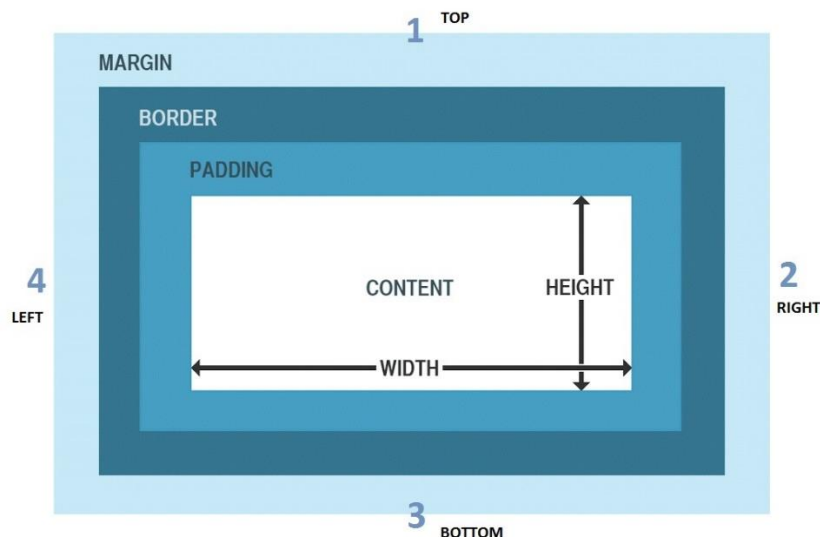
top: superior

right: direito

bottom: inferior

left: esquerdo

É possível definir cada uma dessas quatro dimensões para: **margin**, **border**, **padding** entre outros elementos. Conforme já vimos, mas vamos reforçar com outra imagem semelhante abaixo:



Exemplo de valores para as dimensões da margin:

```
#elemento {
    margin-top: 15px;
    margin-right: 10px;
    margin-bottom: 25px;
    margin-left: 35px;
}
```

Essa é a declaração completa, porém existe uma forma mais prática para declarar a **margin** quando você deseja adicionar margens em todos os lados do elemento.

Exemplo simplificado de valores para as dimensões da *margin*:

```
#elemento {margin: 15px 10px 25px 35px;}
```

Utilizei os mesmos valores justamente para fazer a relação, na maneira curta (*shorthand*) o primeiro valor é responsável pela margem do **topo**, a segunda pela **direita**, a terceira pela da **base** e a quarta pela **esquerda**.

Exemplo: `#elemento {margin: top right bottom left};`

Para não esquecer: Inicia no topo e desce em sentido horário!

Caso o valor da margem do topo seja igual ao da base e o valor da direita seja igual ao da esquerda, você também pode declarar da seguinte forma:

```
#elemento {margin: 25px 10px} /*topo-base direita-esquerda*/
```

Ou seja, o *margin*, *top* e *bottom* do elemento terá de 25px, *left* e *right* terão 10px.

Exemplo: `#elemento {margin: topo/base esquerda/direita}`

E se todos os valores forem iguais você ainda pode declarar da seguinte maneira:

```
#elemento {margin: 10px;}
```

Todas as propriedades da margem podem ter os seguintes valores:

- **auto**: o navegador calcula a margem.
- **length**: comprimento, especifica uma margem em px, pt, cm, etc.
- **%**: especifica uma margem em % da largura do elemento.
- **inherit**: herança, especifica que a margem deve ser herdada do elemento pai.

20 Introdução à Linguagem JavaScript

Nos capítulos anteriores, foi explicado como estruturar os conteúdos de uma página web através do uso de HTML e estilizá-los para que tenham uma aparência interessante, usando CSS. Neste capítulo, será iniciado o estudo da interatividade nos sites, ou seja, além do desenvolvimento de páginas web que não se alteram, será possível desenvolver aplicativos dinâmicos que respondam às entradas dos usuários.

Para isso, será apresentada uma introdução ao uso da Linguagem JavaScript, uma linguagem de *scripts* executada em todos navegadores web.

20.1 Qual a diferença entre JavaScript e Java?

A linguagem de programação JavaScript, desenvolvida pela Netscape Inc., não faz parte da plataforma Java. JavaScript não cria *applets* ou aplicações independentes. Na sua forma mais comum, JavaScript fica embutido nos documentos HTML e pode fornecer níveis de interatividade para páginas web que não são acessíveis com um HTML simples.

Algumas diferenças-chave entre Java e JavaScript:

- Java cria aplicações executadas em uma máquina virtual ou em um browser, ao passo que o código JavaScript é executado apenas em um browser.
- O código Java precisa ser compilado, ao passo que os códigos JavaScript estão totalmente em texto.

Java é uma linguagem pensada para realizar qualquer tipo de programação em qualquer âmbito, desde a programação de aplicações até páginas de servidor, sistemas distribuídos etc. Enquanto que JavaScript serve somente para criar *scripts* que se executem no navegador do usuário quando visita páginas web.

20.2 Programando com JavaScript

Os *scripts* em linguagem JavaScript (JS) devem ser escritos entre os delimitadores `<script></script>`, desta forma é possível escrever scripts JS junto com outras linguagens como HTML, PHP, e os delimitadores irão definir o que JS e o que é PHP, por exemplo. É usual os delimitadores conterem algumas informações, em JS é usada a opção **type**, na qual pode ser informado o tipo de *script* que será criado, conforme exemplo abaixo:

```
<script type="text/javascript"></script>
```

Os *scripts* criados em JS possibilitam que informações possam ser processadas por navegadores e posteriormente apresentadas ao visitante da página. Podem ser usadas caixas de diálogo ou formulários escritos em HTML como forma de interação para que informações sejam enviadas ao *script* ou apresentadas no corpo da página (Nieradka, 2014).

20.3 Entrada e Saída de Dados: Caixas de Diálogo

As caixas de diálogo permitem que haja comunicação e interação com o visitante da página de três formas: **alert**, **prompt** e **confirm**. Nas seções a seguir, serão apresentadas essas três opções com exemplos.

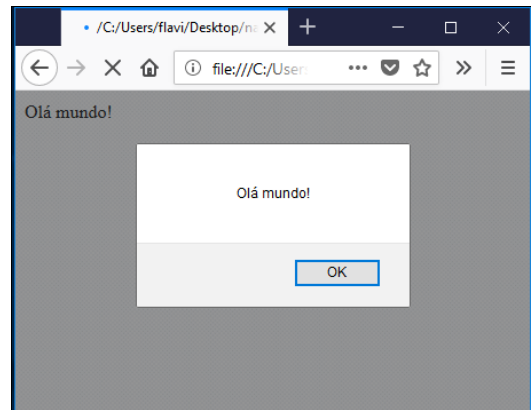
20.3.1 alert

O **alert** apresenta uma janela com um texto para o visitante apenas ler (é uma caixa de texto). O exemplo a seguir mostra a frase “Olá mundo” de duas formas diferentes: no corpo da página e em uma caixa de texto.

Código em JS

```
<script type="text/javascript">
  document.write("Olá mundo");
  alert("Olá mundo")
</script>
```

Resultado no Browser



JS também possibilita ao visitante da página passar informações para o *script* processar usando uma caixa de mensagens com um campo de texto. Além disso, é possível fazer com que as informações sejam armazenadas em variáveis para posterior apresentação.

Variáveis são locais onde as informações podem ficar armazenadas temporariamente. Em algumas linguagens de programação é necessário informar o tipo de dado que será armazenado nas variáveis, mas isso não ocorre na linguagem JS. Em JS, quando uma variável é criada, ela recebe o valor *undefined* (indefinido) até que um valor seja atribuído a ela. Ou seja, o tipo da variável é definido no momento em que ela receber o valor. Os valores possíveis de serem armazenados por variáveis em JS são: números reais e inteiros, objetos, caracteres e strings, e valores booleanos (V ou F).

Existem duas maneiras de declarar variáveis em JS: usando **var** seguido do nome da variável ou declarar com atribuição de valor, onde não é necessário o uso da palavra **var**.

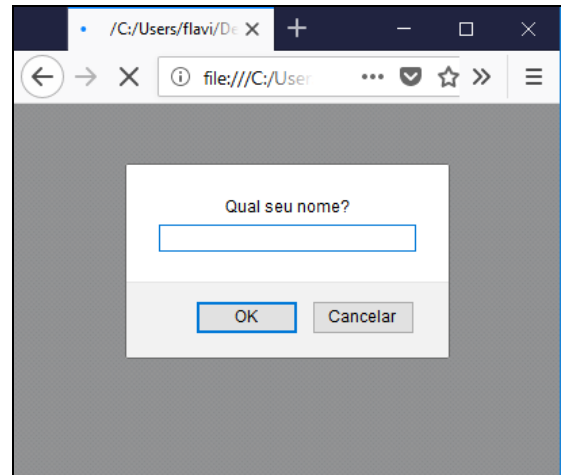
20.3.2 *prompt*

O *prompt* apresenta uma janela com um campo de texto para que o visitante possa inserir textos. O exemplo a seguir apresenta um código que permite a visitante informar seu nome e mostra como armazenar informações em uma variável para apresentação no navegador.

Código em JS

```
<script type="text/javascript">
  var nome;
  nome=prompt("Qual seu nome?");
  document.write("Olá "+nome);
</script>
```

Resultado no Browser



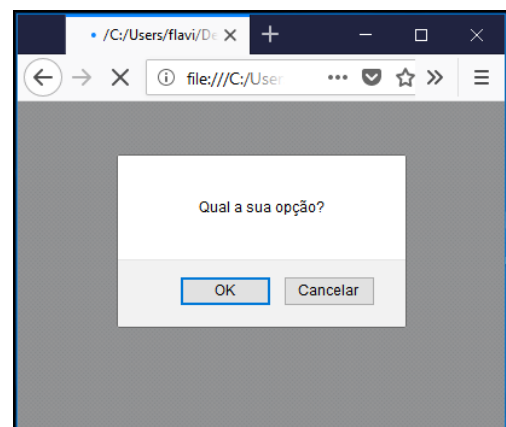
20.3.3 *confirm*

O *confirm* apresenta uma janela de decisão com dois botões para que o visitante escolha sua opção. Além da caixa de diálogo com campo texto editável, JS apresenta outros recursos para interatividade, como a caixa de diálogo de confirmação, nela o visitante pode optar entre confirmar ou cancelar a operação. O código a seguir apresenta um exemplo de caixa de diálogo de confirmação.

Código em JS

```
<script type="text/javascript">
  opcao=confirm("Qual a sua opção?");
  document.write("Sua opção foi "+opcao);
</script>
```

Resultado no Browser



Neste exemplo existe uma variável **opcao** que armazenará o valor retornado pela função **confirm()**. É possível notar que não há a palavra **var** na declaração da variável, pois sua declaração é feita justamente com a atribuição do valor do retorno da função **confirm()**, que será **true** para OK ou **false** para Cancelar.

20.4 Entrada e Saída de Dados: Formulários

Como visto no capítulo sobre formulários em HTML, eles permitem a criação de componentes para entrada e saída de dados. Formulários têm amplo uso na web como preenchimento de dados para cadastro e envio de mensagens. Usando JS é possível fazer a validação de dados, realizar cálculos, criar máscara de entrada ou simplesmente receber dados para algum tipo de processamento. Neste capítulo, veremos a forma como JS pode trabalhar informações provenientes de formulários.

A linguagem JS trata os componentes de um formulário de formas distintas em razão de alguns fatores como estrutura e propriedades. Para compreender melhor como trabalhar com componentes, serão criados exemplos separados detalhando como JS manipula cada um.

20.4.1 Caixa de Texto (textField)

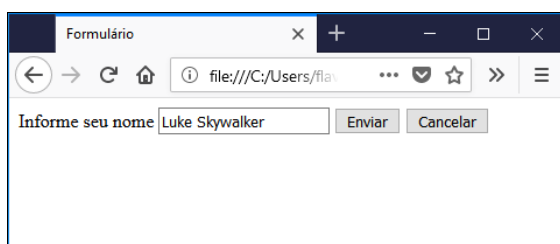
O exemplo a seguir, apresenta a criação do formulário e o *script* com a função que irá mostrar o valor do componente caixa de texto (campo texto ou *textField*).

```

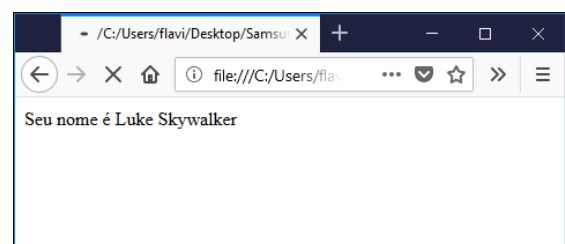
Exemplo4-JS-TextField.html x
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <meta charset="utf-8">
5   <title>Formulário com JS-Campo Texto</title>
6   <script type="text/javascript">
7     function escreveNome(){
8       var texto;
9       texto=document.fExemplo.nome.value;
10      document.write("Seu nome é "+texto);
11    }
12  </script>
13 </head>
14 <body>
15   <form name="fExemplo">
16     Informe seu nome <input type="text" size="20" name="nome">
17     <input type="button" value="Enviar" onclick="escreveNome()">
18     <input type="reset" value="Cancelar">
19   </form>
20 </body>
21 </html>
    
```

Para que os dados do formulário sejam enviados para a função, um evento deve ocorrer. Eventos são ações como um **clique de mouse**, uma tecla pressionada, arrastar um elemento etc. Neste exemplo acima, o evento é o clique do mouse, representado por **onclick** que indica qual função a ser utilizada.

Aparência do Formulário no Browser



Resultado ao clicar em Enviar



20.4.2 Caixa de Seleção (*select*)

A maneira como o JS trata o componente caixa de seleção (*select*) é semelhante a como trata o componente campo texto, alterando apenas a quantidade e formato da informação apresentada.

Existem duas maneiras de trabalhar JS juntamente com HTML. Uma delas é como visto no exemplo anterior, quando é usado o **script embutido no cabeçalho** do documento HTML, e a outra é trabalhar com **scripts externos**, que será apresentada nos exemplos a seguir.

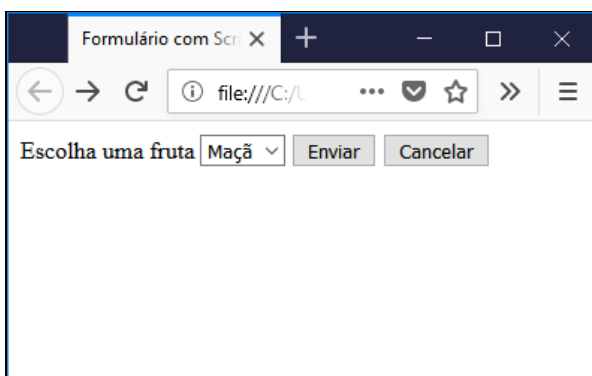
```

1 <!DOCTYPE html>
2 <html>
3 <head>
4   <meta charset="utf-8">
5   <title>Formulário com Script Externo-Select</title>
6   <script type="text/javascript" src="scriptExemplo5.js"></script>
7 </head>
8 <body>
9   <form name="fExemplo">
10    Escolha uma fruta
11    <select name="fruta">
12      <option value="Maçã">Maçã</option>
13      <option value="Pera">Pera</option>
14      <option value="Uva">Uva</option>
15    </select>
16    <input type="button" value="Enviar" onclick="mostraFruta()">
17    <input type="reset" value="Cancelar">
18  </form>
19 </body>
20 </html>
    
```

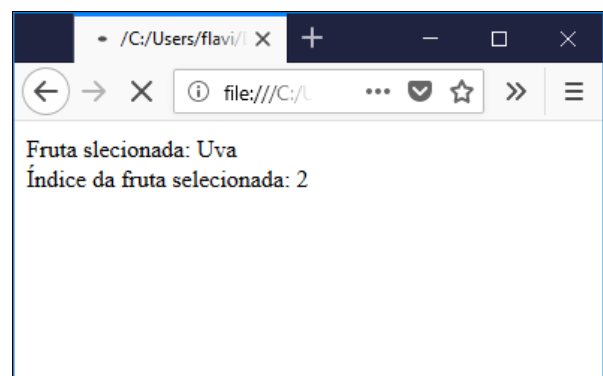
```

1 function mostraFruta(){
2   var indice,valor;
3   indice=document.fExemplo.fruta.selectedIndex;
4   valor=document.fExemplo.fruta.value;
5   document.write("Fruta selecionada: "+valor);
6   document.write("<br>Índice da fruta selecionada: "+indice);
7 }
    
```

Aparência do Formulário no Browser



Resultado ao Escolher Uva e clicar em Enviar



O **índice** é obtido através da propriedade **selectedIndex** e o **valor** com a propriedade **value**. Perceba que, qualquer que seja a informação que se deseja obter de um elemento de um formulário, o caminho é sempre o mesmo:

document.nome-do-form.nome-do-elemento.propriedade-desejada

20.5 Funções

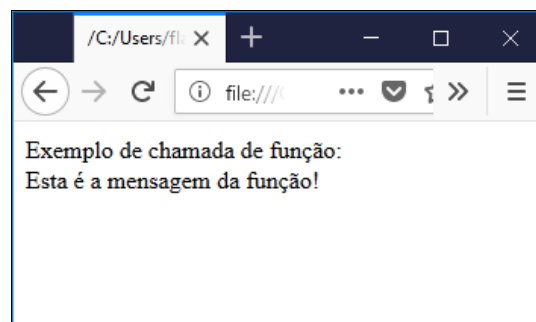
Em alguns dos exemplos anteriores, foram usadas funções que realizavam determinadas tarefas, porém sem muitos detalhes. As funções apresentadas nos exemplos a seguir mostrarão mais detalhes e novos conceitos, como parâmetros e retorno de valores.

As funções são sub-rotinas (subprogramas) usadas para modularizar um programa, fazendo com que cada função realize uma das tarefas necessárias, tornando o programa mais fácil de ser compreendido e também de mais fácil manutenção.

O exemplo abaixo apresenta uma função que executa uma instrução que escreve uma mensagem na tela, o importante deste exemplo é a forma como a função é chamada.

```

Exemplo1-FuncoesJS.html
1 <meta charset="utf-8">
2 <script type="text/javascript">
3     function EscreverMsg(){
4         document.write("Esta é a mensagem da função!");
5     }
6     document.write("Exemplo de chamada de função: <br>");
7     EscreverMsg();
8 </script>
    
```



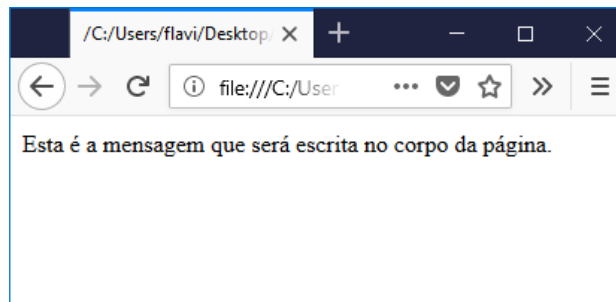
20.5.1 Parâmetros

Uma função pode também realizar tarefas com dados enviados a ela, ou seja, é possível enviar informações para as funções e essas informações são chamadas de parâmetros e são escritas entre os parênteses da função.

O exemplo a seguir apresenta uma função que recebe uma frase por parâmetro e a escreve no corpo da página.

```

Exemplo2-FuncoesJS-Parametros.html
1 <meta charset="utf-8">
2 <script type="text/javascript">
3     function EscreverMsg(msg){
4         document.write(msg);
5     }
6     texto="Esta é a mensagem que será escrita no corpo da página.";
7     EscreverMsg(texto);
8 </script>
    
```



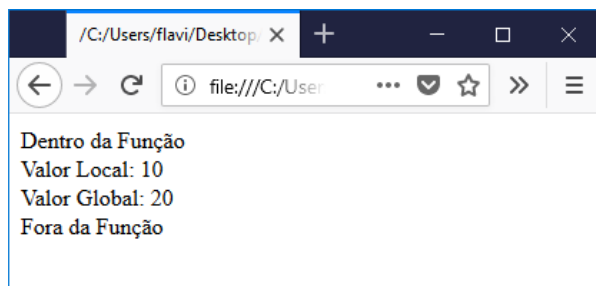
20.5.2 Variáveis Globais e Locais

Ao falarmos de funções, não podemos deixar de falar de variáveis globais e locais. A diferença entre elas é que uma **variável global** pode ser acessada de qualquer parte do programa (dentro de funções ou fora), já uma **variável local** existe apenas dentro da função onde ela foi criada (não pode ser acessada fora da função). O exemplo a seguir apresenta um programa usando variáveis globais e locais.

```

1 <meta charset="utf-8">
2 <script type="text/javascript">
3     var global;
4     function teste(){
5         var local=10;
6         document.write("Dentro da Função");
7         document.write("<br>Valor Local: "+local);
8         document.write("<br>Valor Global: "+global);
9     }
10    global=20;
11    teste();
12    document.write("<br>Fora da Função");
13    document.write("<br>Valor Local: "+local);
14    document.write("<br>Valor Global: "+global);
15 </script>
    
```

Este exemplo acima produzirá o resultado abaixo, com um erro proposital. Como pode ser observado na imagem abaixo, o programa executou apenas até a linha 12. As linhas 13 e 14 não foram executadas, pois na linha 13 está sendo feita uma referência à variável **local** fora da função, sendo que ela só existe dentro da função `teste()`. A linguagem JS interpreta linha por linha e processa em tempo real, então quando há algum erro, a execução é interrompida e, a partir dali, nada mais é executado.

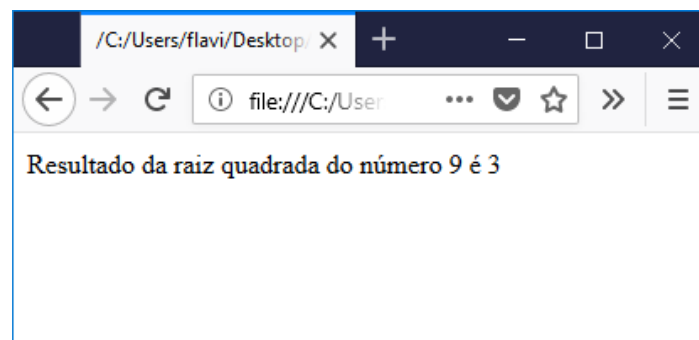


20.5.3 Retorno de Valores

Outra característica importante das funções, é o fato de que elas podem retornar valores, ou seja, uma função pode realizar sua tarefa (cálculos, repetições, testes etc.) e, em vez de apresentar o resultado final para o usuário, pode **retornar o resultado da sua tarefa para o local onde ela foi chamada**. O próximo exemplo apresenta uma função que realiza o cálculo da raiz quadrada de um número e retorna o resultado (para o local onde ela foi chamada).

```

Exemplo4-FuncoesJS-RetornaVal.html x
1 <meta charset="utf-8">
2 <script type="text/javascript">
3     var raiz,numero;
4     function CalculaRaiz(valor){
5         var resultado;
6         resultado=Math.sqrt(valor);
7         return resultado;
8     }
9     numero=prompt("Informe um número: ");
10    raiz=CalculaRaiz(numero);
11    document.write("Resultado da raiz quadrada do número "+numero+" é "+raiz);
12 </script>
    
```



20.6 Validação de Dados em Formulários

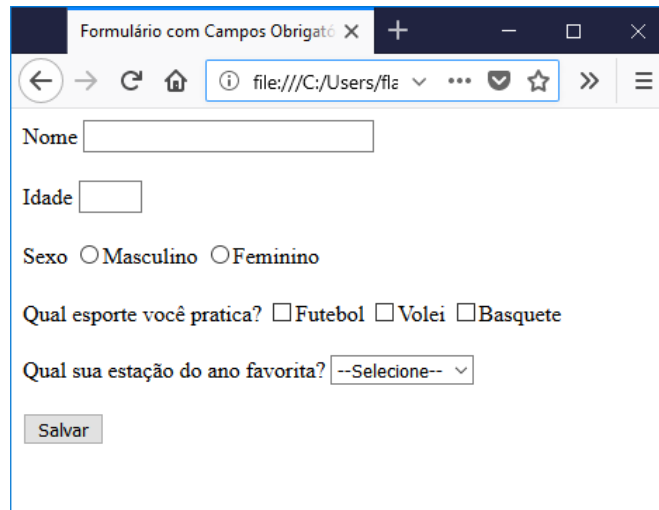
Formulários disponíveis na internet em geral são utilizados para realizar cadastros e, para garantir que os dados sejam preenchidos corretamente, é necessário que sejam feitas validações nas informações preenchidas. Os campos dos formulários devem ser validados tanto na forma como são preenchidos, como datas, endereços etc. como também na obrigatoriedade de preencher alguns campos importantes.

20.6.1 Validação de Dados em Entrada em Formulários

Inicialmente, será apresentada uma validação de obrigatoriedade de preenchimento, na qual serão disparadas mensagens ao usuário da página informando que faltou preencher alguns campos do formulário. O exemplo a seguir apresenta um formulário em HTML e o script que irá fazer a validação dos campos do formulário.

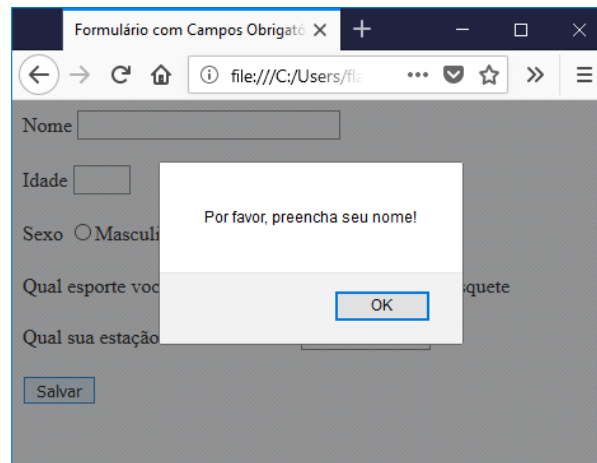
```

Exemplo1-ValidarFormJS-CamposObrigatorios.html
1 <!DOCTYPE html>
2 <html>
3 <head>
4     <meta charset="utf-8">
5     <script src="ValidarCamposObrigatorios.js" type="text/javascript"></script>
6     <title>Formulário com Campos Obrigatórios</title>
7 </head>
8 <body>
9     <form name="fCadastro" action="javascript:DadosRecebidos()">
10        Nome <input type="text" name="nome" size="30"><br><br>
11        Idade <input type="text" name="idade" size="3"><br><br>
12        Sexo <input type="radio" name="sexo" value="masculino">Masculino
13             <input type="radio" name="sexo" value="feminino">Feminino<br><br>
14        Qual esporte você pratica?
15             <input type="checkbox" name="esporte" value="futebol">Futebol
16             <input type="checkbox" name="esporte" value="volei">Volei
17             <input type="checkbox" name="esporte" value="basquete">Basquete<br><br>
18        Qual sua estação do ano favorita?
19             <select name="estacao">
20                 <option value="">--Selecione--</option>
21                 <option value="verao">Verão</option>
22                 <option value="outono">Outono</option>
23                 <option value="primavera">Primavera</option>
24                 <option value="inverno">Inverno</option>
25             </select><br><br>
26        <input type="button" value="Salvar" onclick="validacao()">
27    </form>
28 </body>
29 </html>
    
```



```

ValidarCamposObrigatorios.js x
1 function validacao(){
2     var x,caixa=false, erro=false;
3     if(document.fCadastro.nome.value==""){
4         alert("Por favor, preencha seu nome!");
5         document.fCadastro.nome.focus();
6     }
7     if(document.fCadastro.idade.value==""){
8         alert("Por favor, preencha sua idade!");
9         document.fCadastro.idade.focus();
10    }
11    if(!document.fCadastro.sexo[0].checked && !document.fCadastro.sexo[1].checked){
12        alert("Por favor, informe seu sexo!");
13    }
14    for(x=0;x<3;x++){
15        if(document.fCadastro.esporte[x].checked){
16            caixa=true;
17        }
18    }
19    if(caixa==false){
20        alert("Por favor, informe pelo menos um esporte!");
21    }
22    if(document.fCadastro.estacao.selectedIndex==0){
23        alert("Por favor, informe a estação do ano favorita!");
24    }
25    if(erro==false){
26        document.fCadastro.submit();
27    }
28 }
    
```



20.6.2 Função para Apresentar Dados do Formulário

Observando a imagem abaixo, a linha 9 contém o código que chama a função **DadosRecebidos()**, através da propriedade **action**, sempre que o formulário for enviado. Nessa mesma linha 9, pode ser observado que tem a palavra **javascript**: antes do nome da função que será chamada, isto é necessário para indicar que o que está sendo chamado é uma função JS.

Essa função chamada **dadosRecebidos()** deve ser criada no mesmo arquivo **ValidarCamposObrigatorios.js**, onde já existe a função **validacao()**. Esta é uma prática comum, onde cria-se um arquivo com todas funções necessárias.

```

Exemplo1-ValidarFormJS-CamposObrigatorios.html • ValidarCamposObrigatorios.js x
1  <!DOCTYPE html>
2  <html>
3  <head>
4      <meta charset="utf-8">
5      <script src="ValidarCamposObrigatorios.js" type="text/javascript"></script>
6      <title>Formulário com Campos Obrigatórios</title>
7  </head>
8  <body>
9      <form name="fCadastro" action="javascript:dadosRecebidos()">
10         Nome <input type="text" name="nome" size="30"><br><br>
11         Idade <input type="text" name="idade" size="3"><br><br>
12         Sexo <input type="radio" name="sexo" value="masculino">Masculino
13              <input type="radio" name="sexo" value="feminino">Feminino<br><br>
14         Qual esporte você pratica?
15             <input type="checkbox" name="esporte" value="futebol">Futebol
16             <input type="checkbox" name="esporte" value="volei">Volei
17             <input type="checkbox" name="esporte" value="basquete">Basquete<br><br>
18         Qual sua estação do ano favorita?
19             <select name="estacao">
20                 <option value="">--Selecione--</option>
21                 <option value="verao">Verão</option>
22                 <option value="outono">Outono</option>
23                 <option value="primavera">Primavera</option>
24                 <option value="inverno">Inverno</option>
25             </select><br><br>
26             <input type="button" value="Salvar" onclick="validacao()">
27         </form>
28 </body>
29 </html>
    
```

```

Exemplo1-ValidarFormJS-CamposObrigatorios.html  ValidarCamposObrigatorios.js x
1  function validacao(){
2      var x,caixa=0, erro=false;
3      if(document.fCadastro.nome.value==""){
4          alert("Por favor, preencha seu nome!");
5          document.fCadastro.nome.focus();
6          erro=true;
7      }
8      if(document.fCadastro.idade.value==""){
9          alert("Por favor, preencha sua idade!");
10         document.fCadastro.idade.focus();
11         erro=true;
12     }
13     if(!document.fCadastro.sexo[0].checked && !document.fCadastro.sexo[1].checked){
14         alert("Por favor, informe seu sexo!");
15         erro=true;
16     }
17     for(x=0;x<3;x++){
18         if(document.fCadastro.esporte[x].checked){
19             caixa=1;
20         }
21     }
22     if(caixa==0){
23         alert("Por favor, informe pelo menos um esporte!");
24         erro=true;
25     }
26     if(document.fCadastro.estacao.selectedIndex==0){
27         alert("Por favor, informe a estação do ano favorita!");
28         erro=true;
29     }
30     if(erro==false){
31         document.fCadastro.submit();
32     }
33 }

```

```

34
35 function dadosRecebidos(){
36     var n,i,s,e ;
37     n=document.fCadastro.nome.value;
38     i=document.fCadastro.idade.value;
39     if(document.fCadastro.sexo[0].checked){
40         s=document.fCadastro.sexo[0].value;
41     }
42     else{
43         s=document.fCadastro.sexo[1].value;
44     }
45     for(x=0;x<3;x++){
46         if(document.fCadastro.esporte[x].checked){
47             e=document.fCadastro.esporte[x].value;
48         }
49     }
50     est=document.fCadastro.estacao.value;
51     document.write("Nome= "+n);
52     document.write("<br>Idade= "+i);
53     document.write("<br>Sexo= "+s);
54     document.write("<br>Esporte= "+e);
55     document.write("<br>Estação= "+est);

```




```

57  /*
58  ----tentando escrever direto, sem guardar nas variaveis antes, nao dá, pois o write
59  apaga o html, e entao os dados do form sao perdidos!----
60
61  document.write("Nome= "+document.fCadastro.nome.value);
62  document.write("Idade= "+document.fCadastro.idade.value);
63  if(document.fCadastro.sexo[0].checked){
64      document.write("<br>Sexo= "+document.fCadastro.sexo[0].value);
65  }
66  else{
67      document.write("<br>Sexo= "+document.fCadastro.sexo[1].value);
68  }
69  for(x=0;x<3;x++){
70      if(document.fCadastro.esporte[x].checked){
71          document.write("<br>Esporte= "+document.fCadastro.esporte[x].value);
72      }
73  }
74  document.write("<br>Estação= "+document.fCadastro.estacao.value);
75  */
76  }
    
```

20.6.3 Forma mais Elegante de Apresentar Dados do Formulário

Uma forma mais elegante de apresentar mensagens é usar elementos HTML em vez de caixas de diálogo. Para fazer referência a elementos HTML é possível fazer uso da propriedade ID para criar uma identificação para o elemento e então usar o método `getElementById()`. Vamos usar o mesmo exemplo anterior, mas algumas alterações.

No arquivo HTML, a única alteração que faremos é criar uma área para as mensagens de erro, usando a tag `<div>`, conforme na linha 9 do exemplo abaixo:



```

Exemplo2-ValidarFormJS-MsgDivGetElementById.html x ValidarCamposObrigatorios.js x
1  <!DOCTYPE html>
2  <html>
3  <head>
4      <meta charset="utf-8">
5      <script src="ValidarCamposObrigatorios.js" type="text/javascript"></script>
6      <title>Formulário com Campos Obrigatórios</title>
7  </head>
8  <body>
9      <div id="mensagem"></div> <!--linha acrescentada-->
10     <form name="fCadastro" action="javascript:dadosRecebidos()">
11         Nome <input type="text" name="nome" size="30"><br><br>
12         Idade <input type="text" name="idade" size="3"><br><br>
13         Sexo <input type="radio" name="sexo" value="masculino">Masculino
14             <input type="radio" name="sexo" value="feminino">Feminino<br><br>
    
```

Caso haja algum erro a ser apresentado, não serão usadas caixas de diálogo, pois as mensagens serão mostradas nessa área criada com a tag `<div>`. O script com as modificações para que as mensagens não sejam mais informadas em caixas de diálogo, mas sim na área identificada como mensagem, está apresentado abaixo:

```

Exemplo2-ValidarFormJS-MsgDivGetElementById.html x ValidarCamposObrigatoriosMensagens.js
1 function validacao(){
2     var x,caixa=0, erro=false;
3     document.getElementById("mensagem").innerHTML="";
4     if(document.fCadastro.nome.value==""){
5         document.getElementById("mensagem").innerHTML="<br>Preencha seu nome";
6         document.fCadastro.nome.focus();
7         erro=true;
8     }
9     if(document.fCadastro.idade.value==""){
10        document.getElementById("mensagem").innerHTML="<br>Preencha sua idade";
11        document.fCadastro.idade.focus();
12        erro=true;
13    }
14    if(!document.fCadastro.sexo[0].checked && !document.fCadastro.sexo[1].checked){
15        document.getElementById("mensagem").innerHTML="<br>Informe seu sexo";
16        erro=true;
17    }
18    for(x=0;x<3;x++){
19        if(document.fCadastro.esporte[x].checked){
20            caixa=1;
21        }
22    }
23    if(caixa==0){
24        document.getElementById("mensagem").innerHTML+="<br>Informe pelo menos um esporte";
25        erro=true;
26    }
27    if(document.fCadastro.estacao.selectedIndex==0){
28        document.getElementById("mensagem").innerHTML+="<br>Informe sua estação favorita";
29        erro=true;
30    }
31    if(erro==false){
32        document.fCadastro.submit();
33    }
34 }

```

21 Referências

Neste capítulo são listadas as referências utilizadas para o desenvolvimento desta apostila.

Livros da Biblioteca:

Livro do Maujor: HTML5 – A Linguagem de Marcação que Revolucionou a Web. Mauricio Samy Silva. Novatec, 2011.

JavaScript + CSS + DOM – Desenvolvimento para Web. Itamar Pena Nieradka. Novaterra, 2014.

Livros na Web:

https://issuu.com/kiraculsteve/docs/12_-_design_responsivo_para_web

http://guilhermemuller.com.br/pt/elearning/html_css_basico/licao/1/introducao-html

<https://developer.mozilla.org/pt-BR/docs/HTML/Introduction>

Livro do Maujor: Fundamentos de HTML5 e CSS3. Mauricio Samy Silva. Novatec, 2015.
www.livrosdomaujor.com.br

<http://www.infowester.com/introhtml5.php>

<http://maujor.com/>

LIVRO oreilly
Aprendendo a Desenvolver Aplicações Web. Semmy Purewal. Novatec, 2014.

https://www.java.com/pt_BR/download/faq/java_javascript.xml

<http://www.w3c.br/divulgacao/guiasreferencia/css2/>

<https://www.vivaolinux.com.br/artigo/MVC-Conceito-e-exemplo-em-PHP>

<http://livrosdomaujor.com.br/html5css3/download.html>

<http://pt-br.html.net/tutorials/css/lesson9.php>

<http://www.icmc.usp.br/ensino/material/html/intro.html>

<http://tableless.github.io/iniciantes/manual/obasico/o-que-front-back.html>

<http://tableless.github.io/iniciantes/manual/css/box-model.html>

<http://www.icmc.usp.br/ensino/material/html/intro.html> Autora: Maria Alice Soares de Castro

<http://www.julioabattisti.com.br/tutoriais/juliosabreu/formularioshtml001.asp>

<http://javascript-coder.com/html-form/html-form-tutorial-p1.phtml>

Engenharia de Software Uma abordagem profissional - 8ª Edição Bookman 2016. By Roger Pressman, Bruce Maxim.